Oracle® Cloud

Understanding Technology Adapters

12 *c* (12.1.3)

E60648–04

October 2016

Documentation for Oracle SOA (Service-Oriented Architecture) developers that describes underlying concepts, context within SOA, and developing and deploying SOA JCA technology adapters.

ORACLE

Oracle Cloud Understanding Technology Adapters, 12 *c* (12.1.3)

E60648–04

Primary Author: Oracle Corporation

Contributors: Amandeep Mahajan, Bo Stern, Srimant Misra, Deepak Agarwal, Raghavendra Chandrashekar, Stephen Mcritchie, Michael Chiocca, Rod Fernandez, Sunil Gopal, Manas Panda, Sagar Shiruguppi, Vikas Anand, Sujay Bandyopadhyay, Syed Zarina, Anuj Kaushal, Ashish Mathur, Prateek Maheshwari, Dhaval B Shah, Sandeep Jain

Contents

5 Oracle JCA Adapter for Sockets

6 Native Format Builder Wizard

8 Oracle JCA Adapter for JMS

9 Oracle JCA Adapter for Database

10 Oracle JCA Adapter for MQ Series

A Oracle JCA Adapter Properties

B Oracle JCA Adapter Valves

C Oracle MQ Series Adapter Supported Encodings

Index

Preface

Audience

Oracle Fusion Middleware Understanding Technology Adapters is intended for anyone who is interested in using these adapters.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

For more information, see the following documents in the *Oracle Fusion Middleware* documentation set:

- System Requirements and Specifications
- Planning an Installation of Oracle Fusion Middleware
- Developing Resource Adapters for Oracle WebLogic Server
- High Availability Guide
- Developing SOA Applications with Oracle SOA Suite.
- Administering Oracle Fusion Middleware.
- Administering Oracle SOA Suite and Oracle Business Process Management Suite.
- Oracle E-Business Suite Adapter User's Guide

- Administering Oracle Service Bus
- Securing Web Services and Managing Policies with Oracle Web Services Manager
- *Oracle® Application Server Installation Guide for Legacy Adapters*

Conventions

The following text conventions are used in this document:

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

What's New in This Guide for Release 12.1.3

This preface introduces the new and changed features of Oracle Integration Adapters that are described in this guide, and provides pointers to additional information.

New and Changed Features for Release 12c

The following topics introduce the new and changed features of Oracle Integration Adapters and other significant changes that are described or referred to in this guide, and provides pointers to additional information.

- Retitled Section 2, "Life Cycle" to "Adapter Framework."
- Added a description of the use and implementation of encryption and decryption to support PII.
- Added chapter about new Adapter: LDAP Adapter. Oracle JCA Adapter for LDAP.
- Added chapter about new Adapter: MSMQ Adapter, including use cases. Oracle JCA Adapter for Microsoft Message Queueing
- Added chapter about new Adapter: Coherence Adapter. Oracle JCA Adapter for Coherence
- Added chapter about new Adapter: JDE World Adapter.
- Added description of user interface which exposes Synchronous Dequeue by Correlation ID in the AQ adapter. Synchronous Dequeue
- Added description of FTP Adapter Extensibility. Features
- Added information on configuring FTP Adapter to Handle Response from MLSD Command Configuring FTP Adapter to Handle Response from MLSD Command
- Extended the Listing Operation to Send MLSD Commands Rather than the List Commands.Extending MLSD
- Extended the Store Operation to Send Additional Proprietary FTP Commands to FTP Server Running on the MVS Platform Extend the Store Operation to Send Additional Proprietary FTP Commands to FTP Server Running on the MVS Platform
- Added information on the FTP Adapter Extension of FTP Client Login FTP Adapter Extension of FTP Client Login
- Added description of support for attachments in the MQ Series Adapter. Processing Messages as Attachment

- Added description of additional ability for MQ Series to read RFH2 Headers.Reading Single or Multiple RFH2 Rules and Formatting Header Version 2 Headers.
- Added description of MQ Series Adapter feature of selectively dequeueing messages from a queue using message selectors and/or filters. Message Selector in the MQ Adapter Configuration Wizard.
- Added description and examples of MQ Series Adapter to directly consume/parse a CCDT configuration file. CCDT Use Cases
- Added description of NIO Support for Socket Adapters. Socket Adapter NIO Support.
- Added description and example of JavaScripting support for Socket Adapters.Java Script Support .
- Added IM and SMS functionality to UMS Adapter. See information in place in Chapter, for example, Figure 11-12, where the process for specifying IM and SMS functionality is documented.
- Added information on Coherence High Availability Support in File/FTP Adapter Chapter. Inbound Operations
- Added description of Database Adapter support for Query By Example.Combining Query by Example with a Regular Query
- Added description of Database Adapter support for ROWID for DELETE/UPDATE. Using ROWID as the Primary Key
- Added discussion and examples of cluster support for Socket Adapter. Cluster Support for Socket Adapter.
- Added description of support for complex lookahead strategies. Defining Complex Look Ahead Strategies for Conditional Processing of Record Using Regular Expressions
- Added description of support for pluggable MFL translator within NXSD. Converting an MFL Format File to Schema Format
- Added description of File/FTP adapter support for attachments in Synchronous Read. Synchronous Read Concepts
- Added description for support for File/FTP Adapters UI for chunked interaction.Chunked Interaction File Adapter Processing.
- Added information about support for Multibyte Streaming in Native Format Builder. Multi-Character Streaming Support
- Added additional information about Skip Locking for the Database Adapter.Configuring PollingInterval_ MaxTransactionSize_ and ActivationInstances in Depth
- Added information about Shared Delimiters to the Native Format Builder. Shared Delimiters
- Moved some MQ Series Adapter code examples to a separate, new Appendix. Oracle MQ Series Adapter Supported Encodings
- Added Chapter on Tuning Adapters. Oracle JCA Adapter Tuning Guide

- Revised and updated the Adapters Property Appendix, including information about properties for new Adapters.Oracle JCA Adapter Properties
- Revised and updated certification-related information on databases and database drivers in chapter on Database Adapters. Supported Third-Party Databases

Part I

Introduction and Concepts

Part I contains the following chapters:

1

Introduction to Oracle JCA Adapters

This chapter provides an introduction to Oracle JCA compliant adapters, which enable you to integrate your business applications, and which provide a robust, lightweight, highly-scalable and standards-based integration framework for disparate applications to communicate with each other. The chapter provides context for the JCA Adapters within today's business application processing.

With the growing need for business process optimization, efficient integration with existing back-end applications has become the key to success. To optimize business processes, you can integrate applications by using JCA 1.5 compliant resource adapters. Adapters support a robust, light weight, highly scalable, and standards-based integration framework, which enables disparate applications to communicate with each other. For example, adapters enable you to integrate packaged applications, legacy applications, databases, and Web services. Using Oracle JCA Adapters, you can ensure interoperability by integrating applications that are heterogeneous, provided by different vendors, based on different technologies, and run on different platforms.

For more information on further configurations, other adapters, refer to the guides in Related Documents.

Note:

A certification matrix relevant for systems requirements for Adapters can be found on the Adapters Home page on the Oracle Technology Network, at http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html

For a list of current adapters, see *http://www.oracle.com/technetwork/middleware/adapters/overview/index-083423.html*

This chapter includes the following sections:

- Features of Oracle JCA Adapters
- Types of Oracle JCA Adapters
- Types of Oracle JCA Adapters Services

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

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

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

Features of Oracle JCA Adapters

This topic discusses the features of Oracle JCA Adapters.

Oracle JCA Adapters provide the following benefits:

- Provide a connectivity platform for integrating complex business processes: Adapters integrate mainframe and legacy applications with enterprise resource planning (ERP), customer relationship management (CRM), databases, and messaging systems. Oracle provides adapters to connect various packaged applications, such as SAP and Siebel, and databases. In addition, adapters integrate middleware messaging systems, such as MQSeries and Oracle Advanced Queuing, and legacy applications, such as CICS and Tuxedo, to provide a complete solution.
- Support open standards: Adapters are based on a set of standards such as J2EE Connector Architecture (JCA) version 1.5, Extensible Markup Language (XML), and Web Service Definition Language (WSDL). The support for standards reduces the learning curve of a user and eliminates the dependency of users on a single vendor.
- Service Component Architecture (SCA) assembly model: Provides the service details and their interdependencies to form composite applications. SCA enables you to represent business logic as reusable service components that can be easily integrated into any SCA-compliant application. The resulting application is known as an SOA composite application. The specification for the SCA standard is maintained by the Organization for the Advancement of Structured Information Standards (OASIS).
- Implement a Service-Oriented Architecture (SOA): The support for open standards enables adapters to implement an SOA, which facilitates loose coupling, flexibility, and extensibility.
- Use native APIs: Adapters support multiple ways of interfacing with the back-end system and provide various deployment options. Using native APIs, adapters communicate with the back-end application and also translate the native data to standard XML, which is provided to the client.
- Model data: Adapters convert native APIs to standard XML and back, based on the adapter metadata configured during design time. Adapter configurations are defined during design time, which is used by runtime components.
- Facilitate real-time and bidirectional connectivity: Adapters offer bidirectional communication with various back-end systems. This includes sending requests to back-end systems and receiving a response. Adapters also support the real-time event notification service. This service notifies about the back-end events associated with successful back-end transactions for creating, deleting, and updating back-end data. This two-way connectivity ensures faster, flexible, efficient integration, and reduces the cost of integration.
- Maximize availability: Oracle JCA Adapters are based on the J2CA 1.5 specification. Adapters can, therefore, fully leverage the scalability and high availability of the underlying Oracle Application Server platform.

  For more information, see *Developing Resource Adapters for Oracle WebLogic Server*.
- Provide easy-to-use design-time tools: Adapters use design-time tools that provide a graphical user interface (GUI) to configure and administer adapters for fast

implementation and deployment. In addition, the tools let you to browse, download, and configure back-end schemas.

- Support seamless integration with Oracle Application Server components: Adapters integrate with Oracle Fusion Middleware. Adapters integrate with the JCA Binding Component of the Oracle Fusion Middleware platform, thereby seamlessly integrating with other service engines and binding components.

Types of Oracle JCA Adapters

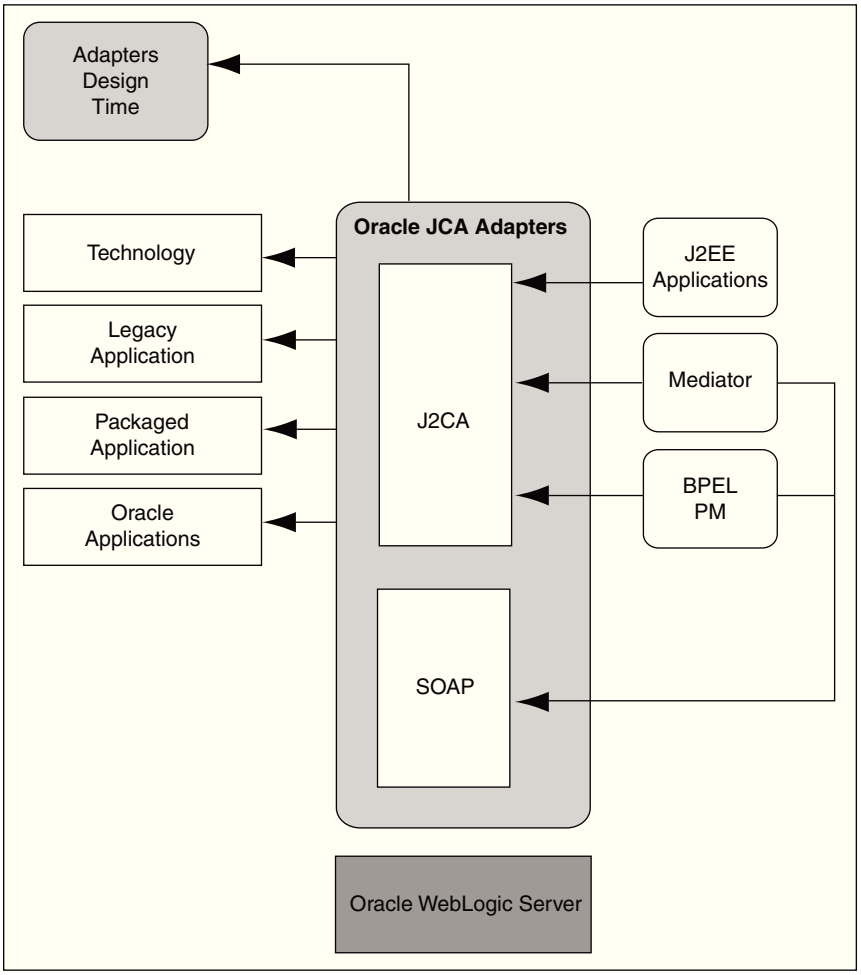
This topic provides information about types of Oracle JCA Adapters.

Oracle JCA Adapters include:

- Oracle Technology Adapters
- Legacy Adapters
- Packaged-Application Adapters
- Oracle E-Business Suite Adapter

Figure 1-1 illustrates the different types of adapters.

Figure 1-1 Types of Oracle JCA Adapters



Oracle Technology Adapters

Oracle technology adapters integrate Oracle Application Server and Oracle Fusion Middleware components such as Oracle BPEL Process Manager (Oracle BPEL PM) or Oracle Mediator components to file systems, FTP servers, database queues (advanced queues, or AQ), Java Message Services (JMS), database tables, and message queues (MQ Series).

These adapters include:

- Oracle JCA Adapter for Files/FTP
- Oracle JCA Adapter for Sockets
- Oracle JCA Adapter for AQ
- Oracle JCA Adapter for JMS
- Oracle JCA Adapter for Database
- Oracle JCA Adapter for MQ Series
- Oracle JCA Adapter for UMS
- Oracle JCA Adapter for LDAP
- Oracle JCA Adapter for Microsoft Message Queueing
- Oracle JCA Adapter for Coherence
- Oracle JCA Adapter for JDE Edwards World

Oracle technology adapters are installed as part of Oracle Fusion Middleware.

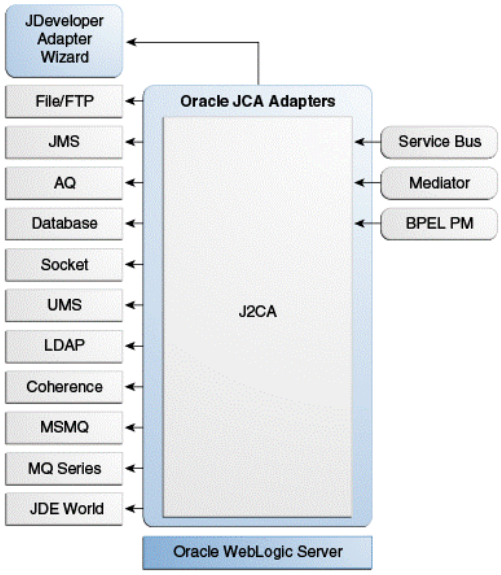
This section includes the following topics:

- Architecture
- Design-Time Components
- Runtime Components
- Deployment

Architecture

Oracle technology adapters are based on J2EE Connector Architecture (JCA) 1.5 standards and deployed as a resource adapter in the same Oracle WebLogic Server as Oracle Fusion Middleware. Oracle Adapter for Oracle Applications has a similar architecture as that of the Oracle technology adapters. Figure 1-2 illustrates the architecture of Oracle technology adapters.

Figure 1-2 Oracle Technology Adapters Architecture



Design-Time Components

During design time, Oracle technology adapters use Oracle JDeveloper (JDeveloper) to generate the adapter metadata. Binding configuration files consist of J2CA-centric XML markup. The J2CA binding configuration files are used by the JCA Binding Component to seamlessly integrate the J2CA 1.5 resource adapter with Oracle Fusion Middleware.

For more information about integration of Oracle technology adapters with Oracle Fusion Middleware, see Adapter Integration with Oracle Fusion Middleware.

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

Example of generating WSDL and binding configuration files for Oracle JCA Adapter for Database:

By using JDeveloper, you can configure Oracle JCA Adapter for Database. This adapter helps you to perform data manipulation operations, call stored procedures or functions, and publish database events in real time. To configure adapter definitions, drag and drop **Database Adapter** from the Components window to the External References swim lane.

Figure 1-3 shows how to browse through the Import Tables window to select the required tables for the adapter.

Figure 1-3 Browsing for Required Tables

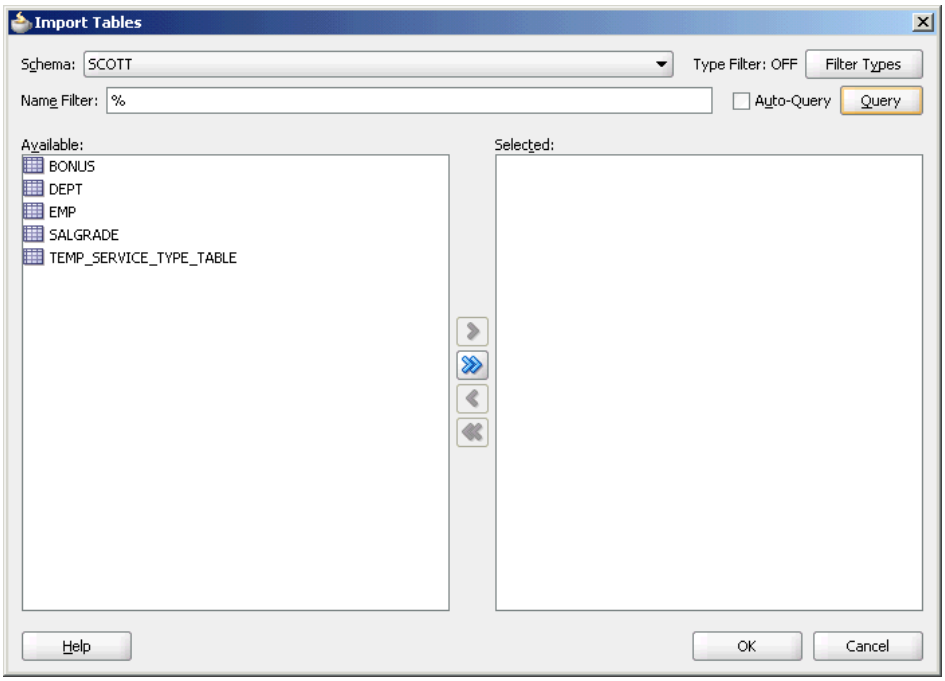
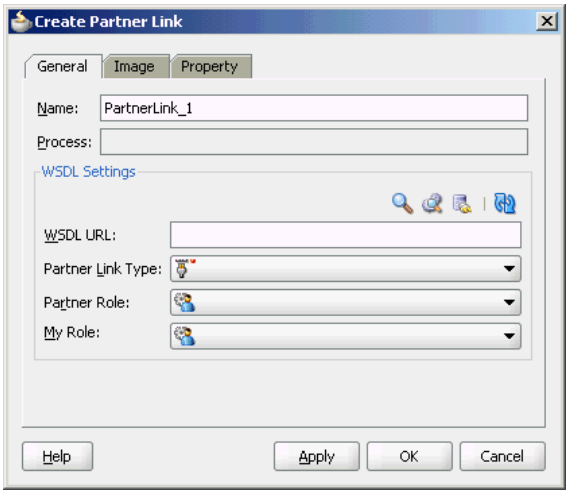


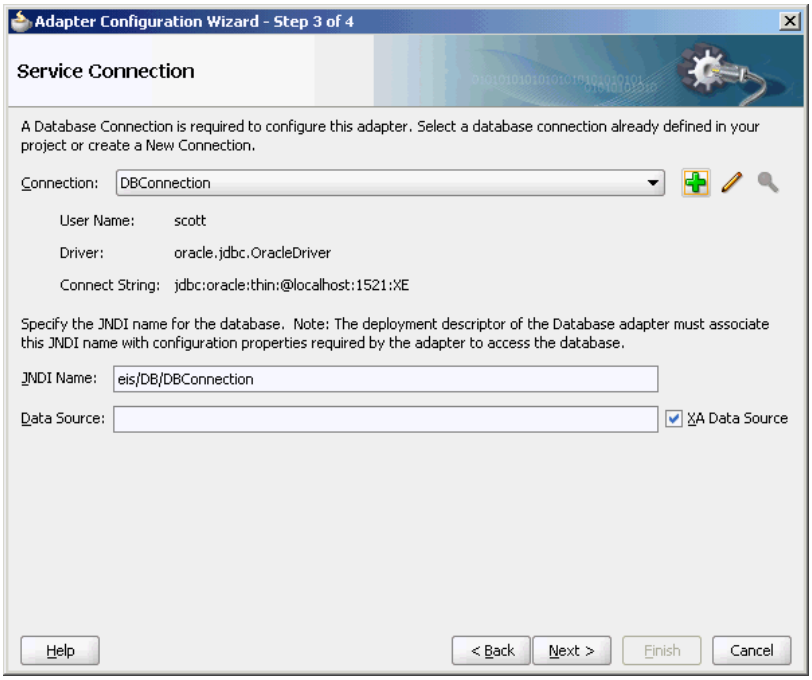
Figure 1-4 shows how to specify the WSDL settings for Oracle JCA Adapter for Database.

Figure 1-4 Specifying WSDL Settings



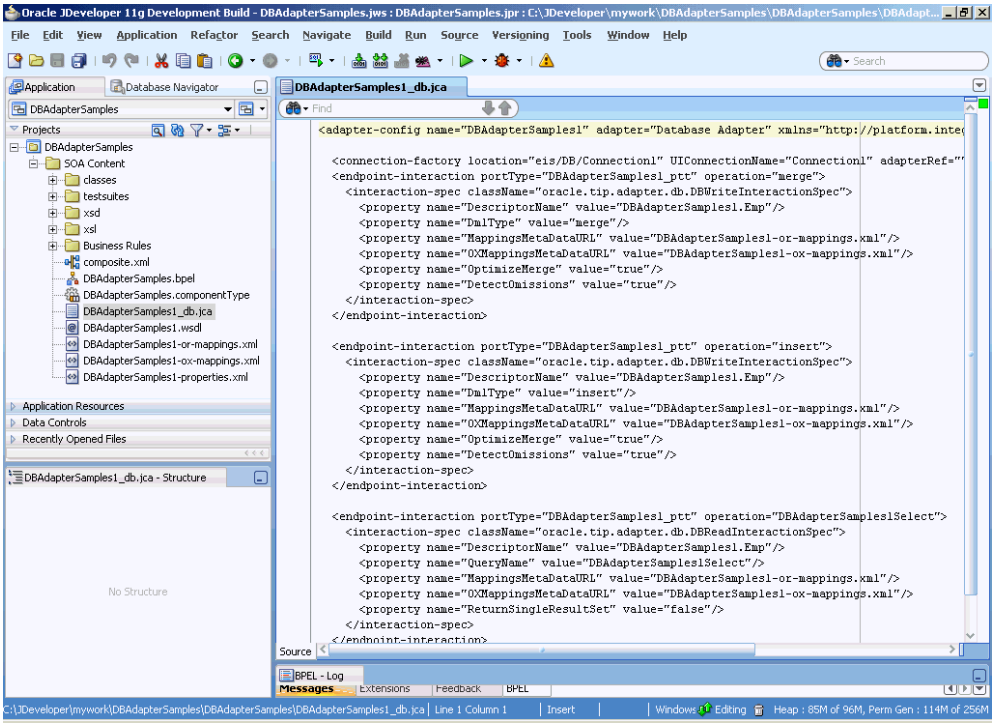
Next, you must establish a database connection, select an operation type, and select the required tables. The runtime connection parameters are specified in the `weblogic-ra.xml` file and linked to a Java Naming and Directory Interface (JNDI) name, which is specified during design time. Figure 1-5 shows the creation of a new database connection.

Figure 1-5 Creating a New Database Connection



Finally, JDeveloper generates a WSDL file and a binding configuration file with the J2CA binding for the Oracle JCA Adapter for Database, as shown in Figure 1-6.

Figure 1-6 Structure of a JCA File



Runtime Components

The runtime component of Oracle technology adapters is the J2CA 1.5 resource adapter for the specific back-end application. Oracle technology adapters are deployed

in the J2CA container of the Oracle WebLogic Server. Oracle Fusion Middleware integrates with these J2CA 1.5 adapters through the JCA Binding Component, which converts Web service messages into J2CA interactions and back.

Oracle Fusion Middleware uses the JCA Binding Component to integrate the request-response service (J2CA outbound interaction) with a SCA composite reference and publish the adapter events to a SCA composite service.

For more information about integration with Oracle Fusion Middleware, see Adapter Integration with Components .

Fusion Middleware Control Accessibility and Technology Adapters

Fusion Middleware Control provides accessibility options for the pages on which you monitor and manage applications. Fusion Middleware Control supports screen readers and provides standard shortcut keys to support keyboard navigation. You can also view the console pages in high contrast or with large fonts for better readability. For information and instructions on configuring accessibility in Fusion Middleware Control, see "Using Oracle Fusion Middleware Accessibility Options" in *Administering Oracle Fusion Middleware*.

Deployment

Oracle technology adapters are deployed as J2CA 1.5 resource adapters within the same Oracle WebServer container as that of Oracle Fusion Middleware during installation. Although Oracle technology adapters are physically deployed as J2CA 1.5 resource adapters, their logical deployment involves creating the connection factory entries for the J2CA 1.5 resource adapter by editing the `weblogic-ra.xml` file and using JDeveloper during design time. By using JDeveloper, you specify the JNDI name, which acts as a placeholder for the connection used when your composite is deployed to the Oracle WebLogic Server. This placeholder enables you to use different databases for development and later production. However, for the logical deployment changes (that is, only if you are creating a new outbound connection) to take effect, the WebLogic Server container process should be updated. However, if you are updating any outbound connection property for an existing JNDI, then you must restart the Oracle WebLogic Server. To avoid a server restart when you update an outbound connection property for an existing JNDI, refer to Adding or Updating an Adapter Connection Factory.

Legacy Adapters

Legacy adapters integrate Oracle Application Server with legacy and mainframe applications using legacy communication protocols.

These adapters include:

- OracleAS Adapter for Tuxedo
- OracleAS Adapter for CICS
- OracleAS Adapter for VSAM
- OracleAS Adapter for IMS/TM
- OracleAS Adapter for IMS/DB

Legacy adapters are available as part of the OracleAS Adapters CD.

This section includes the following topics:

- Architecture
- Design-Time Components
- Runtime Components
- Deployment

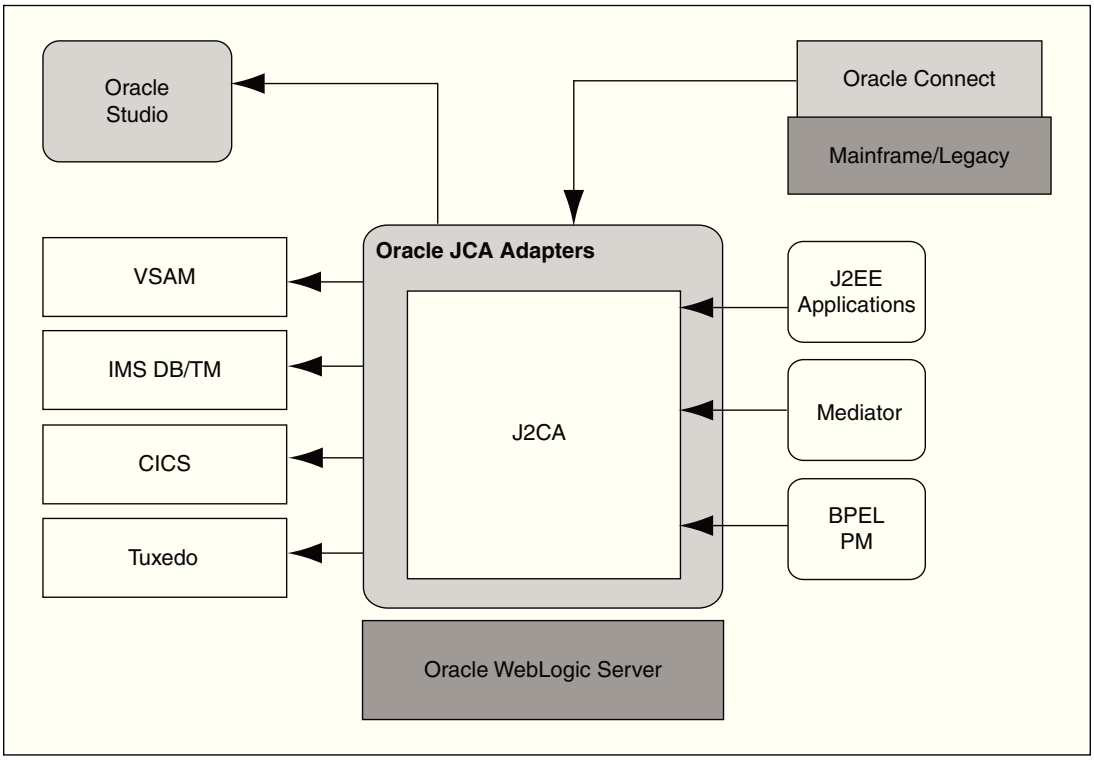
Architecture

Legacy adapters include the following components in the architecture

- Oracle Connect
- Oracle Studio
- J2CA Adapter

Figure 1-7 illustrates the architecture of legacy adapters.

Figure 1-7 Legacy Adapter Architecture



Changed Data Capture (CDC) adapters also have the same architecture.

Oracle Connect

Oracle Connect is a component that resides on the legacy and mainframe platforms. It consists of native adapters for communicating with the mainframe application and data stores. Oracle Connect consists of the following components:

- **Server Processes**
- **Native Adapters**
- **Daemon**

- **Repository**

Server Processes: Oracle Connect consists of multiple servers to process client requests.

Native Adapters: Oracle Connect consists of various embedded native adapters to communicate with Tuxedo and IMS-TM transaction systems, and database drivers to communicate with various databases and file systems on mainframe systems such as VSAM and IMS-DB. The native adapters convert application structures, such as the legacy COBOL applications data, to and from XML. The XSD schema is used for precise mapping between mainframe data and standard XML data.

Daemon: Daemon is an RPC-based listener that manages and maintains multiple server configurations. It runs on every computer running Oracle Connect and handles user authentication and authorization, connection allocation, and server process management.

When a client requests for a connection, the daemon allocates a server process to handle this connection. The allocated server process may be a new process or any process that has been running. Further communication between the client session and the server process is direct and does not involve the daemon. However, the daemon is notified when the connection ends and the server process is either killed or being used by another client.

The daemon supports multiple server configurations called workspaces. Each workspace defines accessible data sources, applications, environment settings, security requirements, and server allocation rules. The daemon authenticates clients, authorizes requests for a server process within a certain server workspace, and provides clients with the required servers. The allocation of servers by the daemon is based on the workspace that the client uses. Thus, a client can access a data source using one workspace, where a server process is allocated from an existing pool of servers, or the client can access a data source using a different workspace, where a new server process is allocated for each client request. A fail-safe mechanism enables the specification of alternate daemons, which function as a standby for high availability.

Repository: Oracle Connect supports a repository for storing the XML-based schema and configuration information. There is a single repository for each Oracle Connect instance. The repository stores the following information:

- Oracle Connect configuration settings (including the Daemon settings to control client/server communication)
- User profiles to enable single sign-on to multiple back-end applications and data sources
- Adapter metadata for each adapter, which includes adapter request-response and event services

Oracle Studio

Oracle Studio is the design-time tool for configuring the Oracle AS Adapters for mainframes. It enables you to configure the services, events, and connection information for native adapters. The configuration information is stored in the Oracle Connect repository on the legacy or mainframe application. In addition, it enables you to do management and monitoring of Oracle Connect. The Oracle Studio is available only on the Windows platform. The Oracle Studio is based on the Eclipse GUI framework.

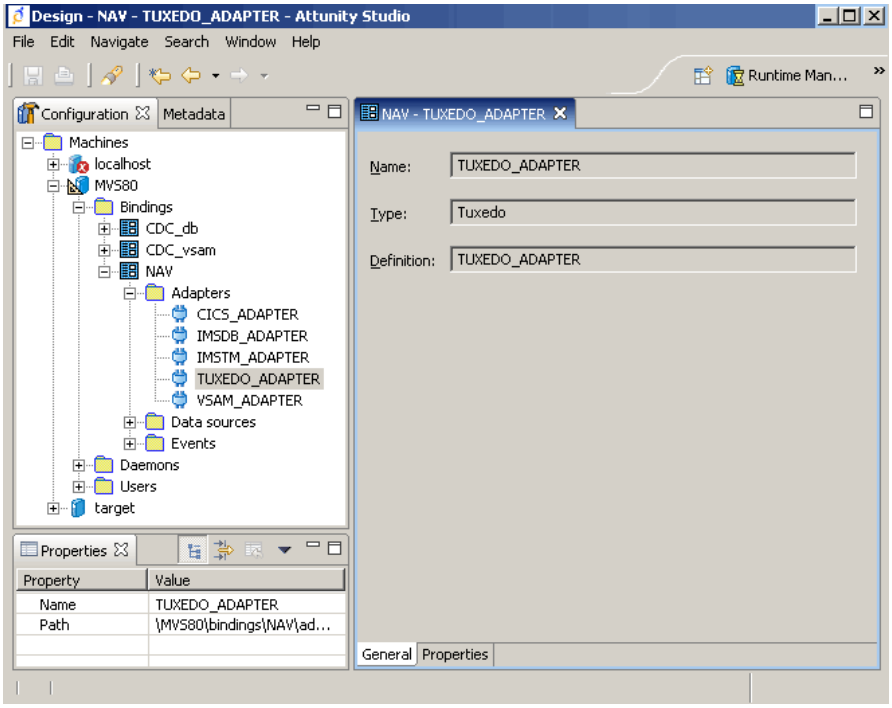
J2CA Adapter

The J2EE Connector Architecture (J2CA) adapter forwards the WebLogic Server application client requests to the Oracle Connect application. Oracle Connect communicates with the mainframe application and forwards the response back to the J2CA adapter. The response might contain the transaction data or might contain the exception data if the request generated an error. Oracle Fusion Middleware integrates with Oracle Connect through the J2CA Legacy adapter.

Design-Time Components

To configure legacy adapters during design time, use Oracle Studio, as shown in Figure 1-8.

Figure 1-8 Oracle Studio



Example of configuring OraleAS Adapter for Tuxedo:

Using Oracle Studio, you can configure OracleAS adapter for Tuxedo, as shown in Figure 1-9 and Figure 1-10.

Figure 1-9 Configuring OracleAS Adapter for Tuxedo

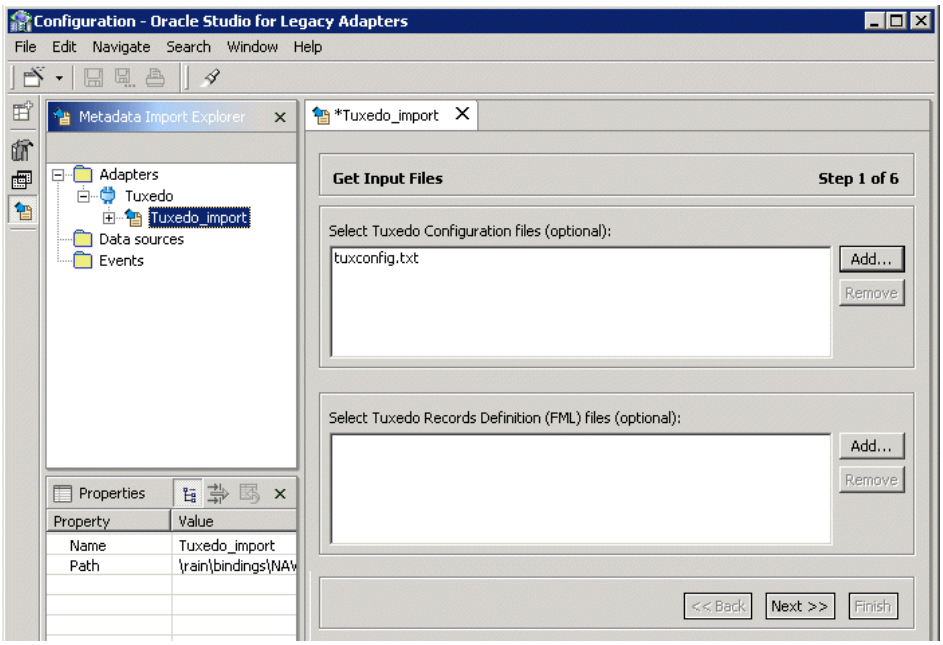
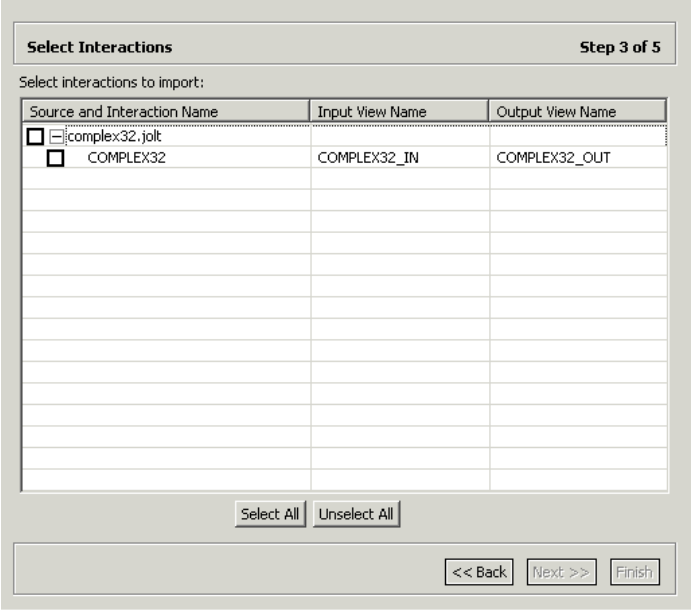


Figure 1-10 Selecting the Types of Interactions for OracleAS Adapter for Tuxedo



Runtime Components

During runtime, WSDL files generated during design time are consumed by the integrating components. For example, Oracle Fusion Middleware uses the JCA Binding Component to integrate the request-response service (J2CA outbound interaction) with a BPEL Invoke activity and to publish the events to a BPEL process receive activity.

For more information, see Adapter Integration with Oracle Fusion Middleware.

Deployment

Legacy adapters are deployed as J2CA resource adapters within the Oracle WebLogic Server J2CA container during installation. The adapter must be in the same Oracle WebLogic Server container as that of the Oracle Fusion Middleware for integration.

Packaged-Application Adapters

Packaged-application adapters integrate the Oracle Application Server with various packaged applications, such as SAP and Siebel.

In SOA Suite 11g, Oracle provided the following enterprise application adapters:

- OracleAS Adapter for PeopleSoft
- OracleAS Adapter for SAP R/3
- OracleAS Adapter for Siebel
- Oracle AS Adapter for J.D.Edwards EnterpriseOne and OneWorld

These adapters are downloaded and installed separately from the standard SOA Suite install. Also, all of the design-time tasks for these adapters are performed by the Oracle Application Explorer to generate the required WSDL and schema artifacts and these are then imported into JDeveloper.

Note the distinction, the older Oracle iWay adapter is for the JDE EnterpriseOne and JDE OneWorld products, whereas the new JDE Adapter is only for JDE World product.

In SOA Suite 12.1.3, there are two "out of the box" Adapters that are available with SOA Suite.

- Oracle Adapter for SAP, and
- Oracle Adapter for J.D.Edwards World

The design-time for these adapters, similar to other technology adapters, is Oracle JDeveloper.

The adapter components are available "out of the box" in the Components from where user can model integrations with the respective applications. Refer to the Licensing Document for licensing information on these adapters.

This section includes the following topics:

Architecture

Packaged-application adapters can be deployed as J2EE Connector Architecture (J2CA) 1.5 resource adapters or as Web service servlets within the Oracle WebLogic Server container. Packaged-application adapters support the Web Service Definition Language (WSDL) and Simple Object Access Protocol (SOAP) interface, in addition to a J2CA interface. J2CA and Web service deployments of packaged-application

adapters should have a repository project. In J2CA deployment, the resource adapter points to a repository project that can contain multiple back-end connection objects. The deployment descriptor, `weblogic-ra.xml`, points to the J2CA repository project and the connection name to access within the J2CA repository project. In the WSDL deployment, the WSDL repository project consists of a set of WSDL files that describe the adapter metadata.

Note:

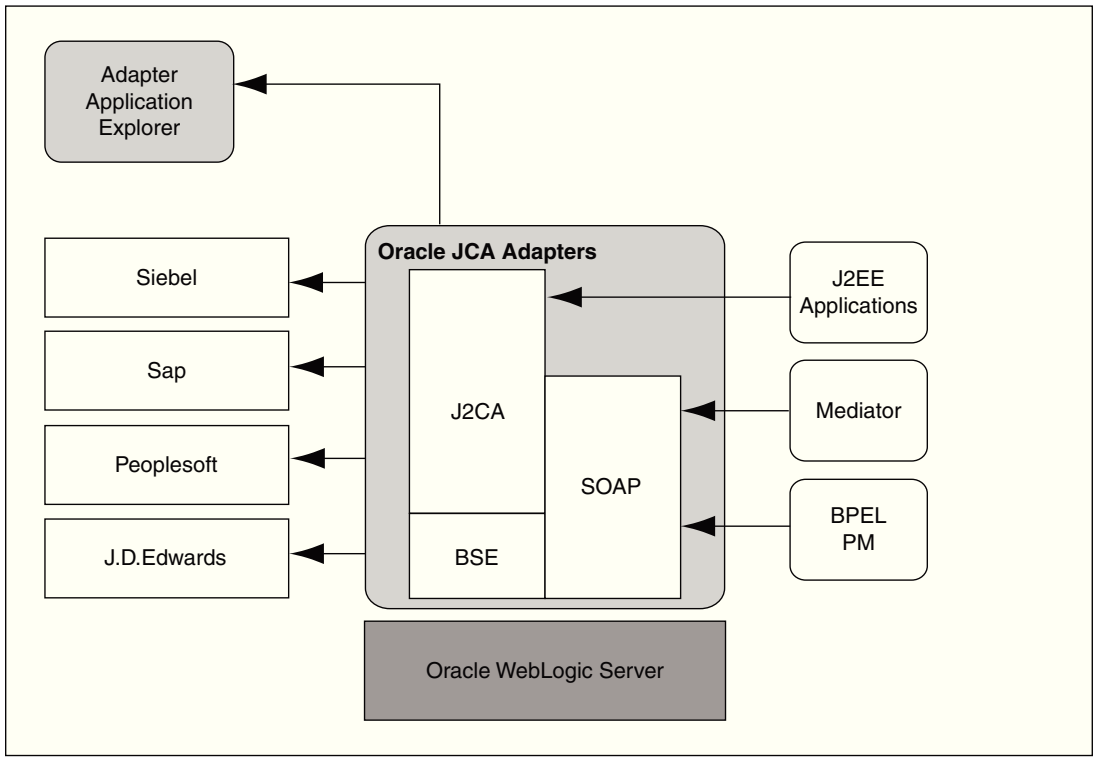
Only the following packaged-application adapters support WSDL and SOAP extensions in this release:

- OracleAS Adapter for SAP
- OracleAS Adapter for Siebel
- OracleAS Adapter for Peoplesoft
- OracleAS Adapter for J.D. Edwards

The architecture of packaged-application adapters consists of OracleAS Adapter Application Explorer (Application Explorer), J2CA 1.5 resource adapter, and Business Services Engine (BSE).

Figure 1-11 illustrates the architecture of packaged-application adapters:

Figure 1-11 Packaged-Application Adapters Architecture



This section describes the components of the packaged-application adapter architecture.

This section includes the following topics:

- Application Explorer
- BSE
- J2CA 1.5 Resource Adapter

Application Explorer

Application Explorer is a Java swing-based design-time tool for configuring packaged-application adapters. Using Application Explorer, you can configure the back-end application connection, browse the back-end application schemas, and expose these schemas as adapter services. Application Explorer is shipped with packaged application-specific plug-ins for browsing the back-end application-specific metadata.

You can use Application Explorer to create repository projects for the Oracle Adapter. Each repository project can consist of multiple back-end application connections. The schemas are represented as either XML Schema Definition (XSD) for the Oracle Adapter J2CA interface or BSE or as a WSDL with SOAP binding.

BSE

Application Explorer works with BSE, which is deployed in the Oracle WebLogic Server container of the Oracle Application Server. BSE uses SOAP as a protocol for accepting requests from clients, interacting with the back-end application, and sending responses from the back-end application back to clients.

J2CA 1.5 Resource Adapter

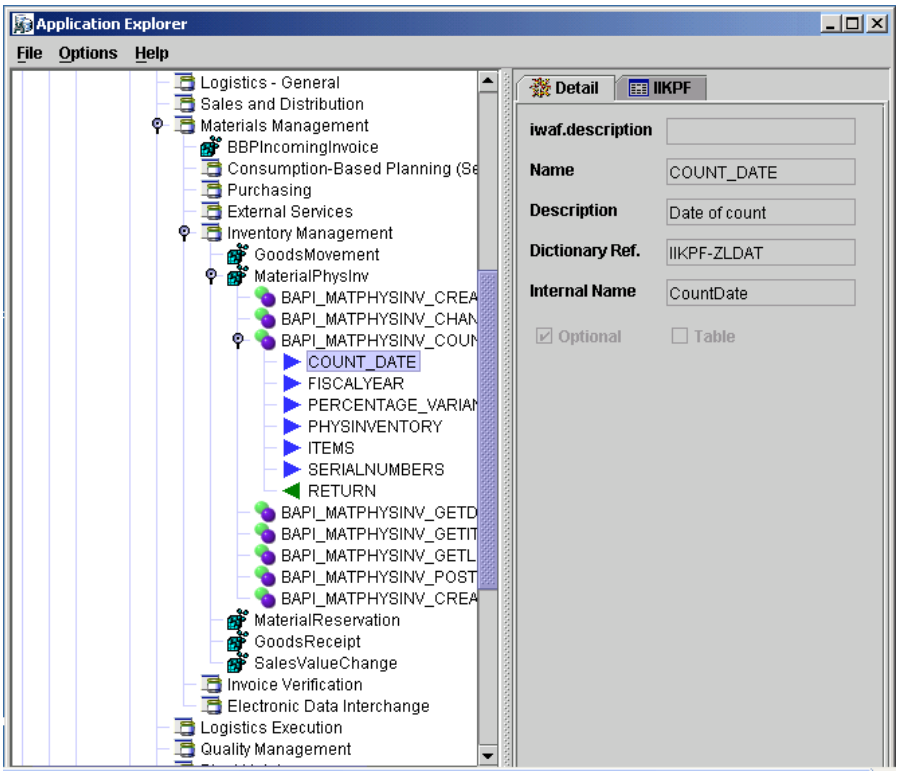
The J2CA 1.5 resource adapter consists of a Channel component for receiving back-end events.

Design-Time Components

Application Explorer is used to configure packaged-application adapters during design time. This tool is used to create a repository project for the J2CA 1.5 resource adapter, which contains a list of back-end connections. Application Explorer exposes back-end metadata as XSD and WSDL with J2CA extensions. The XSD metadata is used by the Oracle WebLogic Server application clients for integration through the J2CA Common Client Interface (CCI) Application Programming Interface (API). The WSDL with J2CA extension is used for integration with Business Process Execution Language for Web Services (BPEL) Process Manager. The BSE metadata can be defined as WSDL or SOAP.

Figure 1-12 shows the Application Explorer.

Figure 1-12 Application Explorer



Example of generating XML Request Schema for OracleAS Adapter for SAP

You can use Application Explorer to establish a connection for OracleAS Adapter for SAP. To establish such a connection, you must first define a target to OracleAS Adapter for SAP, as shown in Figure 1-13 and Figure 1-14.

Figure 1-13 Selecting OracleAS Adapter for SAP

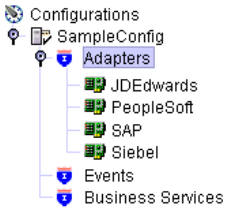
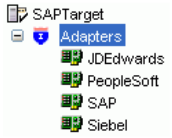
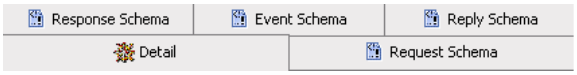


Figure 1-14 Defining a Target to OracleAS Adapter for SAP



After you have explored the SAP business function library and have selected an object, you can use Application Explorer to create the XML request schema and the XML response schema for that function. To view the XML for each schema type, select the required tab, as shown in Figure 1-15:

Figure 1-15 Viewing the XML Schema



Runtime Components

The runtime components of packaged-application adapters include J2CA 1.5 resource adapter, BSE, and servlet. The WebLogic Server application clients use the CCI API to directly interface with the J2CA 1.5 resource adapter. The J2CA 1.5 resource adapter integrates with Oracle Fusion Middleware through the JCA Binding Component. During runtime, the JCA Binding Component translates the Oracle Fusion Middleware service requests to J2CA calls and back based on the adapter metadata (WSDL and binding configuration) configured during design time.

During runtime, the WSDL files generated during design time are consumed by the integrating components. For example, Oracle Fusion Middleware uses the JCA Binding Component to integrate the request-response service (J2CA outbound interaction) with a BPEL process invoke activity and to publish adapter events to a BPEL process receive activity.

For more information about integrating with Oracle Fusion Middleware, see Adapter Integration with Oracle Fusion Middleware.

Deployment

Packaged-application adapters are deployed as J2CA 1.5 resource adapters within the WebLogic Server J2CA container during installation. The adapter must be in the same WebLogic Server container as Oracle BPEL PM for integration.

You can integrate any Web service client with the BSE servlet.

BSE is deployed as a servlet within the WebLogic Server container during installation. BSE can be remotely located and need not be in the same container as the Oracle BPEL PM.

Oracle E-Business Suite Adapter

Oracle Applications are built on a unified information architecture that consolidates data from Oracle and non-Oracle applications and enables a consistent definition of customers, suppliers, partners, and employees across the entire enterprise. This results in a suite of applications that can give you information, such as current performance metrics, financial ratios, profit and loss summaries. To connect Oracle Applications to non-Oracle applications, you use Oracle E-Business Suite Adapter.)

Oracle Adapter for Oracle Applications (E-Business Suite) provides comprehensive, bidirectional, multimodal, synchronous, and asynchronous connectivity to Oracle Applications. The adapter supports all modules of Oracle Applications in Release 12 and Release 11i including selecting custom integration interface types based on the version of Oracle E-Business Suite.

The architecture of the Oracle Adapter for Oracle Applications (E-Business Suite) is similar to Oracle technology adapters.

Types of Oracle JCA Adapters Services

This topic provides information about types of Oracle JCA Adapters services.

Adapters provide the following types of services to facilitate communication between applications:

- Request-Response (Outbound Interaction) Service
- Event Notification (Inbound Interaction) Service
- Metadata Service

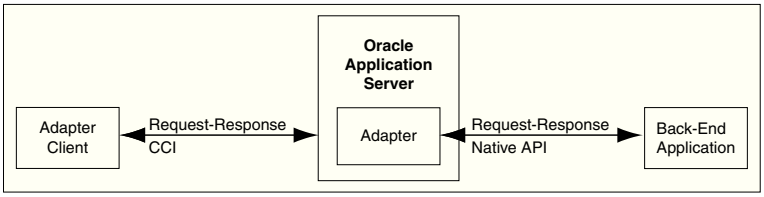
Request-Response (Outbound Interaction) Service

Adapters support the synchronous request-response service. The adapters receive requests from adapter clients, translate these requests into the native back-end data format, and call the appropriate method in the back-end application. In addition, the request-response service retrieves the back-end response to the JCA Binding Component after performing reverse translation. In J2CA terminology, this type of service is also known as outbound interaction.

You can use the request-response service to create, delete, update, and query back-end data, and to call back-end workflows and transactions. For example, a WebLogic Server application client can use OracleAS Adapter for SAP to create a customer within the SAP application.

Figure 1-16 illustrates the request-response service.

Figure 1-16 Request-Response Service



Event Notification (Inbound Interaction) Service

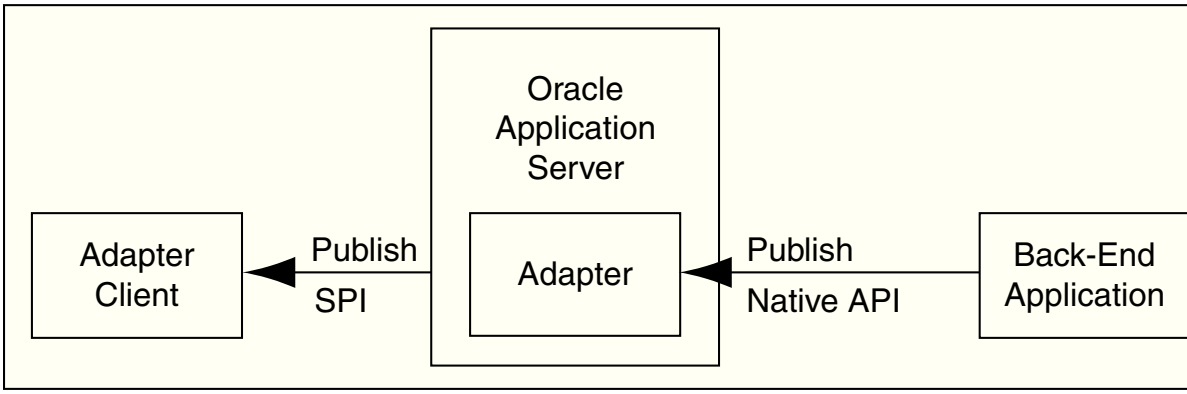
Adapters support the event-notification service, which is an asynchronous communication paradigm. In J2CA terminology, this type of service is also known as inbound interaction.

Adapters either listen or poll for back-end event changes. When listening for events, an adapter registers as a listener for the back-end application that is configured to push events to the adapter. The adapter can also poll the back-end application, which is usually a database or file, for the events required by the client application.

You can use the event-notification service to keep a track of back-end events associated with successful back-end transactions for creating, deleting, and updating back-end data.

Figure 1-17 illustrates the event-notification service.

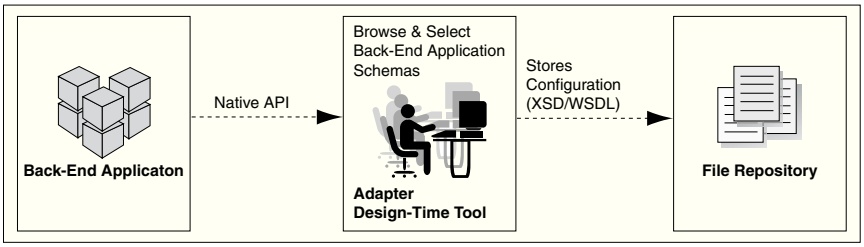
Figure 1-17 Event-Notification Service



Metadata Service

The adapter metadata definition stores information about the back-end connection and schemas for business objects and services. Adapters consist of a design-time component for browsing and storing metadata and a runtime component for running services. The adapter metadata definitions are generated as XML Schema Definition (XSD), WSDL, and binding configuration files. Figure 1-18 illustrates the metadata interaction.

Figure 1-18 Metadata Service



2

Adapter Framework

This chapter describes the installation, starting and stopping, error handling, configuration and deployment of Oracle JCA Adapters that integrate with Oracle Fusion Middleware through the JCA Binding Component.

Oracle JCA Adapters are based on J2EE Connector Architecture (J2CA) 1.5 standards and deployed in the Oracle Containers for Java EE. The life cycle of Oracle JCA Adapters depend on Oracle Fusion Middleware. These adapters integrate with Oracle Fusion Middleware through the JCA Binding Component.

This chapter includes the following sections:

- Recommended Setting for Data Sources Used by Oracle JCA Adapters
- Error Handling
- Testing Applications
- Setting the Trace Level of Oracle JCA Adapters
- Viewing Adapter Logs
- Creating a Custom Adapter

Refer to guides in Related Documents for additional information and configurations.

Installing Oracle JCA Adapters

This topic discusses the installation procedure for Oracle JCA Adapters.

Oracle Technology Adapters and Oracle Adapter for Oracle Applications are available as part of the Oracle Fusion Middleware install. These adapters support both Oracle Containers for Java EE and middle tier deployments.

With SOA Foundation profiles, Adapters are logically grouped into two groups. Tier1 and Tier2.

Tier 1 adapters include:

- File
- FTP
- Database
- JMS
- AQ
- MQ Series
- UMS

Tier 2 adapters include:

- Socket
- LDAP
- Coherence
- MSMQ
- JDE
- SAP
- AS400

Tier 1 adapters are enabled by default when you install this release. The rest of the adapters are in an installed state, however, they are not targeted to any managed servers. You must manually target them in order to use them.

Legacy adapters and packaged-application adapters are available as part of the Oracle Fusion Middleware Adapters and Connectors CD. These adapters support middle tier deployment only.

Note:

Before installing any adapter, consult the documentation related to "System Requirements and Specifications" and "Planning an Installation of Oracle Fusion Middleware" in Related Documents.

Starting and Stopping Oracle JCA Adapters

Oracle JCA Adapters are deployed as JCA 1.5 resource adapters. Therefore, to start or stop an adapter, every resource adapter must implement the `start (BootstrapContext)` and `stop` methods as part of the SPI interface. Oracle JCA Adapters are started when an SOA composite using them starts a JCA outbound interaction. Adapters can also be started when an SOA composite is itself loaded for inbound interactions or when adapters publish events to the JCA binding component.

Once you have started an adapter, you can stop the adapter by shutting down the Oracle Containers for Java EE or by stopping the J2EE application within Oracle Fusion Middleware. In this release, the JCA Binding Component acts as a part of the JCA 1.5 container.

Note that you might see messages such as the following in server startup log files on a server before any composite is deployed. They are harmless warnings, and represent correct Adapter functioning.

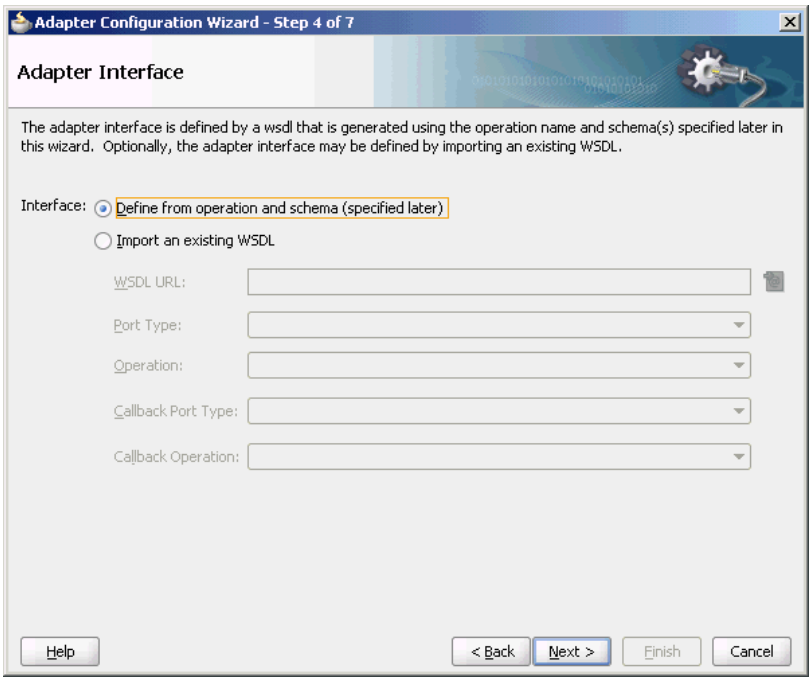
```
 <Warning> <J2EE> <slc01aec> <soa_server1> <[ACTIVE] ExecuteThread: '0' for   queue:
'weblogic.kernel.Default (self-tuning)'>
           <<WLS Kernel>> <>  <57ab8b08-803c-4edb-9da9-82791e12c429-00000004>
              <1372403795938> <BEA-160140>  <Unresolved optional package references
                (in META-INF/MANIFEST.MF):   [Extension-Name: oracle.adapter.ext,
referenced from:   /scratch/vaspatel/fmwhome12/soa/soa/connectors
               /FileAdapter.rar]. Ensure that   the referenced optional package has
been
                     deployed as a library.>  <Warning> <J2EE> <slc01aec>
<soa_server1>
                     <[ACTIVE] ExecuteThread: '0' for  queue:
'weblogic.kernel.Default (self-tuning)'>
                    <<WLS Kernel>> <>
<57ab8b08-803c-4edb-9da9-82791e12c429-00000004>
                    <1372403796778> <BEA-160140>  <Unresolved optional package
references
                      (in META-INF/MANIFEST.MF):  [Extension-Name:
oracle.adapter.ext, referenced from:  /scratch/vaspatel/fmwhome12/soa/soa/
                    connectors/FtpAdapter.rar]. Ensure that  the referenced
optional package has been
                  deployed as a library.>
```

Defining Adapter Interface by Importing an Existing WSDL

You can define an adapter interface in the Adapter Configuration Wizard Adapter Interface page, as shown in Figure 2-1, by using either of the following methods:

- Using a WSDL that is generated using the operation name and schema that you specify in the Adapter Configuration Wizard in the pages that appear after the Adapter Configuration Wizard Adapter Interface page.
- Importing an existing WSDL.

Figure 2-1 The Adapter Configuration Wizard Adapter Interface Page



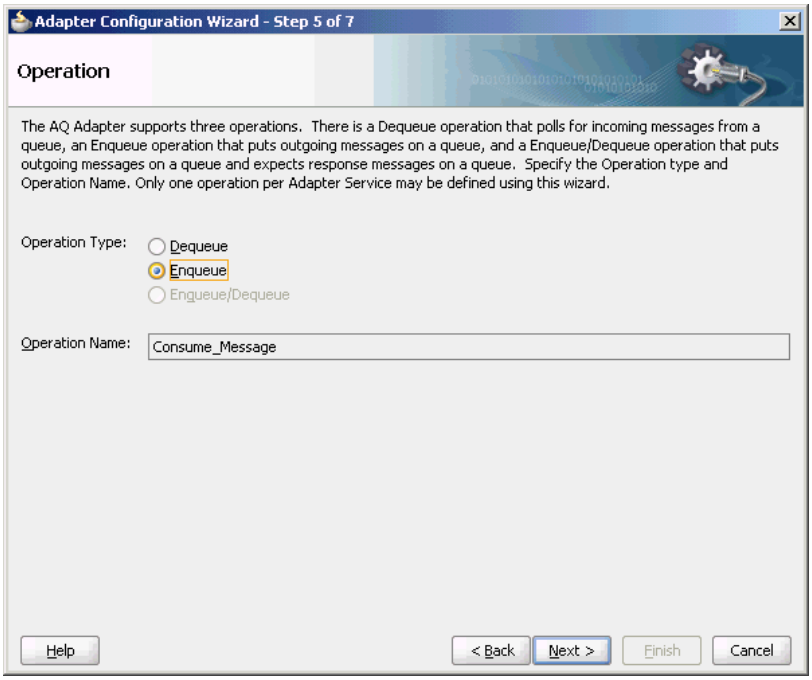
This section describes how to define an adapter interface by importing an existing WSDL. You can use this feature to create an adapter service or reference by using existing WSDLs. The option to choose an existing WSDL is supported for the following adapters only:

- Oracle File Adapter
- Oracle FTP Adapter
- Oracle Socket Adapter
- Oracle AQ Adapter
- Oracle JMS Adapter
- Oracle MQ Series Adapter
- Oracle JCA Adapter for MSMQ
- Oracle JCA Adapter for UMS
- Oracle JCA Adapter for Oracle JD Edwards World
- Oracle JCA Adapter for LDAP
- Oracle JCA Adapter for Coherence

If you select the option of defining the adapter interface by importing an existing WSDL, then some functionalities on subsequent wizard pages are disabled. For

example, because the WSDL defines the operation name and the message schema, the subsequent operation name and schema element fields are automatically filled in and you cannot modify it, as shown in Figure 2-2. However, if you do not choose to use an existing WSDL, then the adapter wizards behaves exactly as before.

Figure 2-2 Operation Page for Oracle AQ Adapter with Fields Automatically Populated



Adapter Configuration Wizard for Oracle MQ Series Adapter, Oracle JMS Adapter and the Oracle AQ Adapter

The Adapter Configuration Wizard for Oracle MQ Series Adapter, Oracle JMS Adapter, and the Oracle AQ Adapter appears different from the other adapters. These adapters have the additional option to select a callback including the port type and operation.

Subsequent options in the Adapter Configuration Wizard are enabled or disabled depending on the port types and operations you select.

Example of Use of Callbacks

For example, while using the Adapter Configuration Wizard for defining the Oracle MQ Series Adapter, if a callback is selected, only the **Send Message to MQ** and **Get Reply/Reports** and the **Get Message** from MQ and Send Reply/Reports Asynchronous options are enabled.

Note that if a callback is not selected, only the Put Message into MQ and Get Message from MQ options are enabled.

If a WSDL operation that has a synchronous reply is selected, only the Get Message from MQ and Send Reply/Reports Synchronous option are enabled. When you use an existing WSDL, the options to use CICS or IMS schemas are disabled.

Note:

The most common approach to importing an existing WSDL is to first create an Oracle BPEL process or a Mediator, and then define their WSDL files from schemas (or NXSD). After this is done, adapter services are created, and the WSDL file generated for the BPEL process or the Mediator component is imported as the *existing* WSDL file.

However, you must keep in mind that this feature works only for those messages which use schema element. Simple and complex types are not supported.

Configuring Message Header Properties for Oracle JCA Adapters

Oracle JCA Adapters expose the underlying back-end operation-specific properties as message header elements and enable the manipulation of these elements within a business process.

As the properties are exposed, you can add, delete, or revert Oracle JCA Adapters properties from the Fusion Middleware Control Console. However, depending on the type of property, you must redeploy your composite application to apply the property change.

Note:

You can modify jca properties using the SOA composite editor's property inspector. However, there is no validation of the properties you have changed if perform changes in this way.

Table 2-1 lists the types of message header properties you can configure and whether redeployment is required.

Table 2-1 Oracle JCA Adapters Property Types

Property Type	Description	Restrictions
Activation specification and interaction specification	Activation specification properties operate as services and interaction specification properties operate as references in a SOA composite application.	Do *not* add or remove these properties. You can only change their values. These properties require the adapter endpoint to be recycled. These types of properties are also dependent upon other properties. If you attempt to add properties, you have no way of knowing which dependent properties must also be added.

Table 2-1 (Cont.) Oracle JCA Adapters Property Types

Property Type	Description	Restrictions
Endpoint	These are tuning-related properties that are not exposed through the activation or interaction specification properties, such as specifying time outs, thresholds, maximum intervals, and so on.	There are no restrictions on adding, removing, or changing endpoint properties. The adapter is notified when these properties are added, removed, or changed, but it does not require redeployment. You cannot add or remove `jca.retry.*` endpoint properties without redeploying the composite. However, you can change these properties by using the Fusion Middleware Control Console without redeploying the composite.

For more information, see Oracle File and FTP Adapters Properties

Physically Deploying Oracle JCA Adapters

Oracle JCA Adapters are deployed as JCA 1.5 resource adapters in an Oracle Containers for Java EE container. Adapters are packaged as Resource Adapter Archive (RAR) files using the Java Archive (JAR) format.

The physical deployment of adapters involves using the RAR file to register the adapters as connectors with the underlying WebLogic Server or the middle tier platform.

Note:

It is important to restart servers and to update deployment after making changes to adapters; otherwise, changes you make do not go into effect. After saving configuration change into a configuration plan file, you must update the deployment to reflect the changes.

The update deployment action synchronizes the new plan file to the staging directory of each server instance. At that point, the new deployment plan is activated in the resource.

The RAR Deployment Descriptor File and the weblogic-ra.xml Template File

The RAR file contains the `ra.xml` file, which is the deployment descriptor XML file containing deployment-specific information about the resource adapter. In addition, the RAR file contains declarative information about the contract between Oracle Containers for Java EE and the resource adapter.

In addition to the `ra.xml` file in the `.rar` file, adapters package the `weblogic-ra.xml` template file. The `weblogic-ra.xml` file is used to define resource adapter `ConnectorFactory` objects (logical deployment). The `weblogic-ra.xml` file is the Oracle Containers for EE-specific deployment descriptor for a resource adapter. It contains deployment configurations for deploying resource adapters to the WebLogic Server, which includes the back-end application connection information as specified in the deployment descriptor of the resource adapter, Java Naming and Directory

Interface (JNDI) name to be used, connection pooling parameters, resource principal mapping mechanism, and configurations.

File	Contents
RAR file	Contains deployment-specific information about resource adapter Contains declarative information about the contract between the Oracle Containers for Java EE and the resource adapters
`weblogic-ra.xml` template file	Defines resource adapter `ConnectorFactory` objects (logical deployment) Contains deployment configurations for deploying resource adapters to the WebLogic Server Provides back-end application connection information as specified in the deployment descriptor of the resource adapter Provides the Java Naming and Directory Interface (JNDI) name to be used Provides the connection pooling parameters Provides a resource principal mapping mechanism Provides configuration information

Creating an Application Server Connection for Oracle JCA Adapters

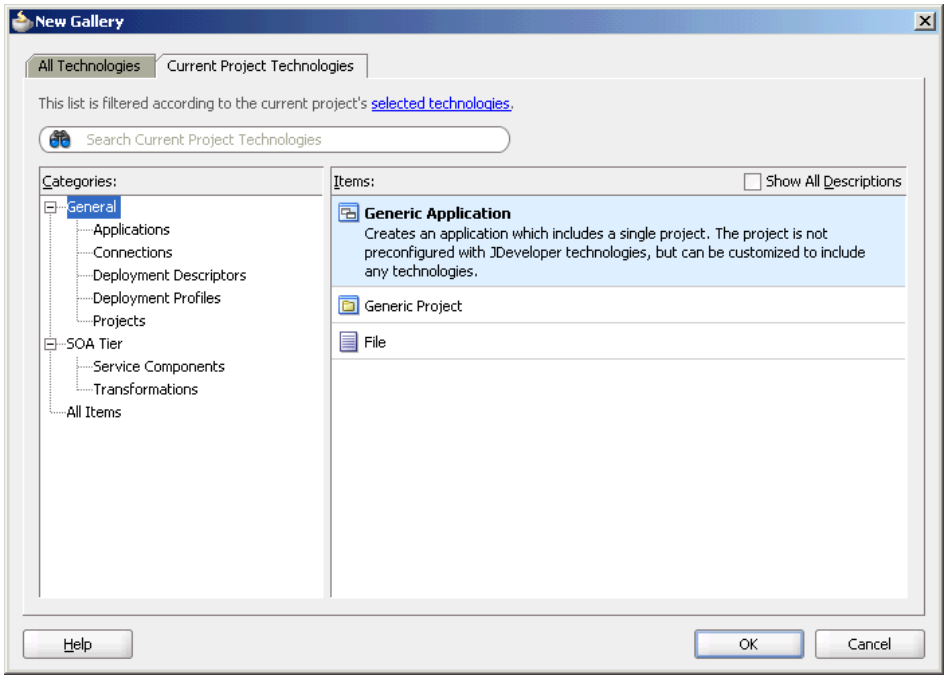
You must establish connectivity between the design-time environment and the server to which you want to deploy. To establish such connectivity, you must create an application server connection.

The following are the steps to create an application server connection:

1. In the **File** menu, click **New**.

   The New Gallery page is displayed, as shown in Figure 2-3.

Figure 2-3 The New Gallery Page



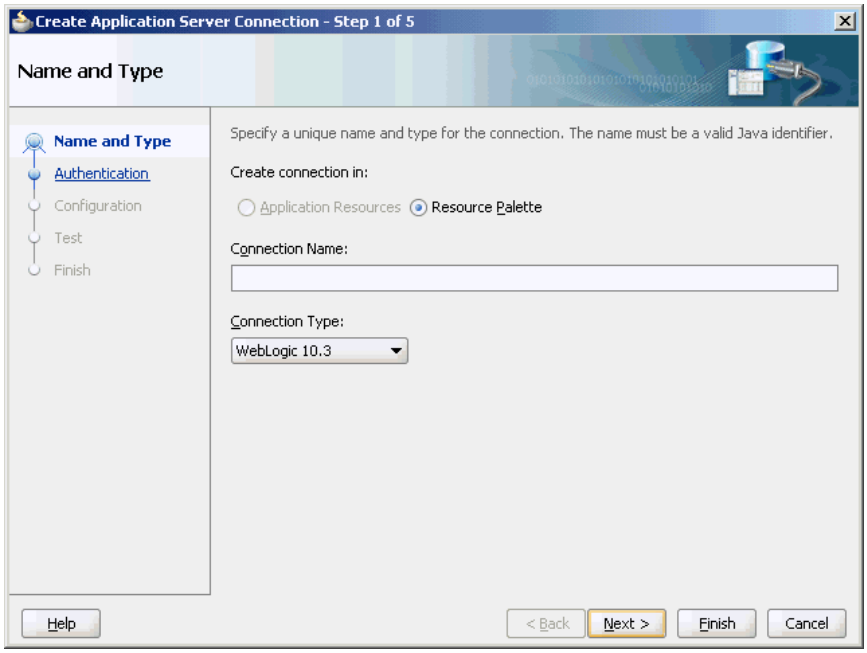
2. In the **All Technologies** tab, under **General** categories, select **Connections**.

   A list of the different connections that you can make is displayed in the Items pane on the right side of the New Gallery page.

3. Select **Application Server Connection**, and then click **OK**.

   The Create Application Server Connection page is displayed, as shown in Figure 2-4.

Figure 2-4 The Create Application Server Connection Name and Type Page

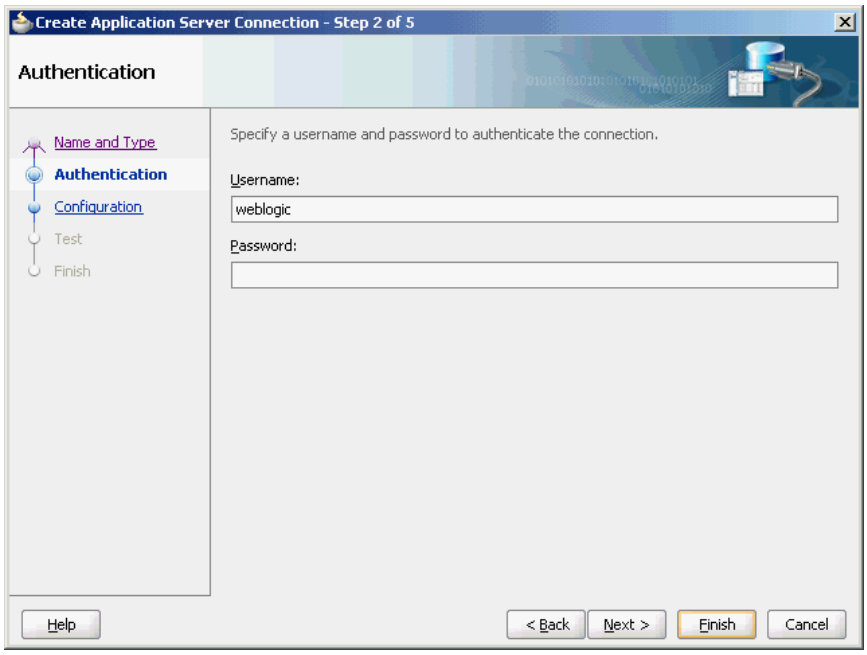


4. Enter a connection name in the **Connection Name** field. For example, `AppsServerConnection1`.

5. Select **WebLogic 10.3.** for Connection Type and click **Next**.

   The Authentication page is displayed, as shown in Figure 2-5.

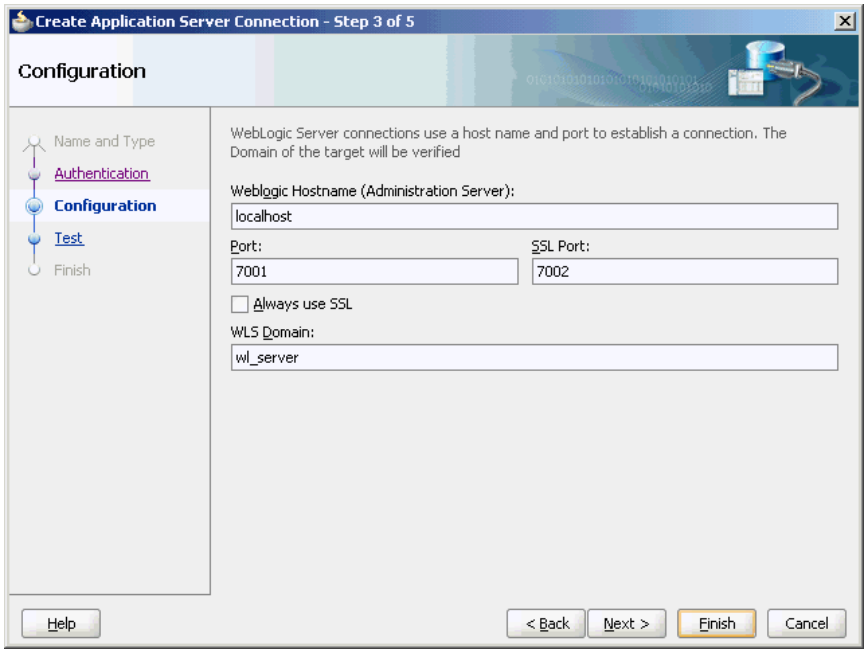
Figure 2-5 The Create Application Server Connection Authentication Page



6. Enter the user name and password, and then click **Next**.

   The Create Application Server Connection Configuration page is displayed, as shown in Figure 2-6.

Figure 2-6 The Create Application Server Connection Configuration Page

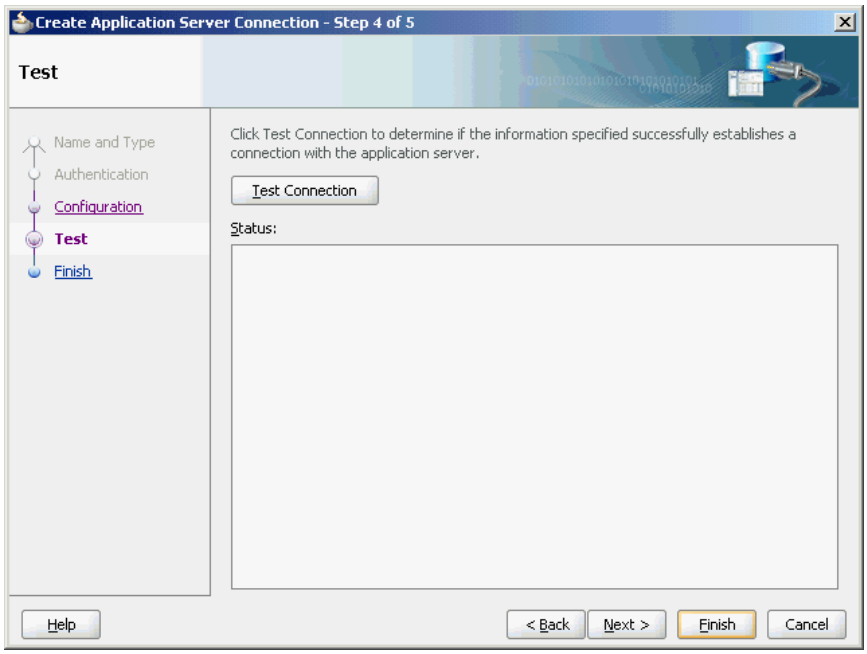


7. Enter the host name, the port details, and the domain server name in the Configuration page.

8. Click **Next**.

   The Create Application Server Connection Test page is displayed, as shown in Figure 2-7.

Figure 2-7 The Create Application Server Connection Test Page



9. Click **Test Connection**. A success message is displayed in the Status pane.

10. Click **Finish**.

    You have created a server connection.

Deploying Oracle JCA Adapter Applications from JDeveloper

You deploy an SOA composite application from JDeveloper.

JDeveloper requires the use of profiles for the SOA projects and applications to be deployed. This section describes how to create and deploy such profiles with JDeveloper.

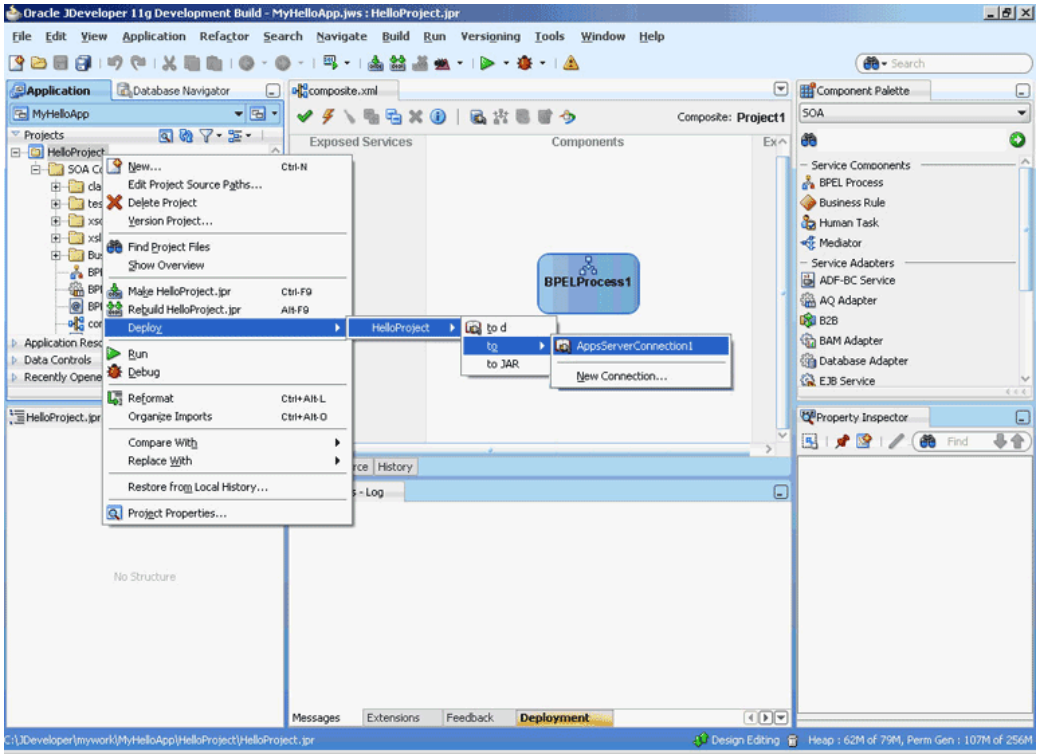
Deploying an Application Profile for the SOA Project and the Application

This section specifically describes how you deploy an application profile for the SOA project and the application. To deploy the application, you must perform the following steps:

1. Right-click the project to deploy, and select **Deploy** > ***project_name***, to ***Application_Server_Connection_Name***, as shown in Figure 2-8.

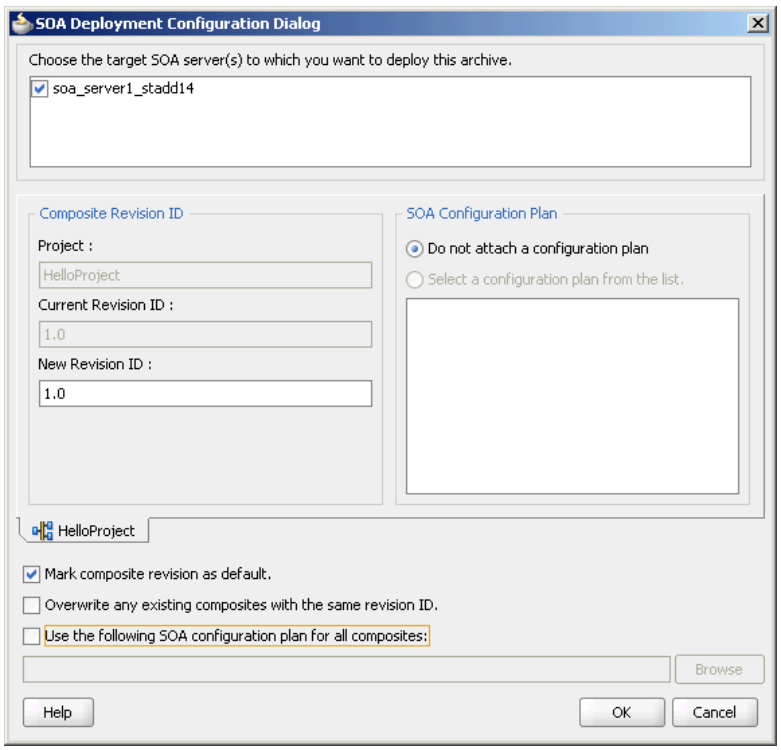
   The SOA Deployment Configuration dialog is displayed.

Figure 2-8 Application Profile Deployment



2. Use the default settings, as shown in Figure 2-9.

Figure 2-9 The SOA Deployment Configuration Dialog



3. Click **OK**.

The Authorization request dialog is displayed.

4. Enter the user name and password, and then click **OK**.

   The project is compiled and deployed to the Managed Server. You can view the deployment log clicking the **Deployment** tab in the design area.

   To redeploy the same version of a SOA composite application, you cannot change the composite name. You can deploy with the same revision number if you selected the **Overwrite any existing composites with the same revision ID** check box on the SOA Deployment Configuration dialog. However, if you do not do so, then the following error message is deployed in the deployment log:

```
pr 29, 2009 1:55:57 AM
oracle.integration.platform.blocks.deploy.
          servlet.CompositeDeployerMessages
severeSendError
SEVERE: Sending back error message:
Error during composite deployment:
oracle.fabric.common.FabricDeploymentException:
Composite with same revision ID already exists:
default/<application name>!<revision id>.
Please set the overwrite flag or use different
                revision ID.
Abort deployment...
```

Adapter Deployment Validation

The JCA Adapter metadata captured during the JCA Adapter design time Configuration wizard screen is validated when it is deployed to the server.

For example. you have chosen the PhysicalDirectory value of "/in/valid/directory"; as the SOA server is trying to deploy the adapter with this metadata, the (File) Adapter looks at all the metadata and throw an exception if one or more pieces of metadata is faulty.

This causes the rollback of the deployment action in JDeveloper. The Adapter runtime validates that all referenced and JNDI-named JCA connection factories are available (that is, they can be looked up without error). Once the data source and connection factory are available, the deployment is validated and proceed.

If you have migrated a project from Releases in the 11xx-series of the Oracle SOA Platform, the deployment validation logic for that project is not enforced. This causes old composite applications not to break.

Manually Deploying an Adapter RAR File that Does Not Have a Jar File Associated With It

This section describes how to manually deploy any adapter RAR file that does not have a jar file associated with it.

If you deploy any adapter RAR file that only contains `META-INF/ra.xml` and `META-INF/weblogic-ra.xml` and also does not contain the jar file adapter required for creating JNDIs, then while deploying, you must change the deployment order to a higher value (say 500) so the Oracle WebLogic Server can deploy this RAR file after the jar file of this adapter is loaded.

Example of Manual Deployment

For example, to deploy the `DBAdapter_NewJndis.rar` file that contains only `META-INF/ra.xml` and `META-INF /weblogic-ra.xml` and does *not* contain the jar file adapter (`DbAdapter.jar`) required while instantiating the new JNDIs, you can follow a specific procedure.

Note:

In this case, after deploying the `DBAdapter_NewJndis.rar` file, you must change the deployment order to a *higher value*. This ensures that the Oracle WebLogic Server deploys the `DBAdapter_NewJndi.rar` file correctly even if you restart the Oracle WebLogic Server.

Use the following steps to manually deploy an adapter RAR file that *does not* have a jar file associated with it:

1. Navigate to the Oracle WebLogic Server Administration Console: `http://servername:portnumber/console`.
2. Use the required credentials to open the Home page of the Oracle WebLogic Server Administration Console.

   The Home page of the Oracle WebLogic Server Administration Console is displayed.
3. Select **Deployments** in the Domain Structure pane.

   The Oracle WebLogic Server Administration Console Summary of Deployments page is displayed.
4. Click **Install**.

   The Install Application Assistant page is displayed.
5. Enter the path of the application directory or file in the **Path** field, and then click **Next**.
6. Select the servers to which you want to deploy this application, and then click **Next**.

   The Optional Settings page is displayed.
7. Modify these settings or accept the defaults, and then click **Next**.

   The Review your choices and click Finish page is displayed.
8. Click **Finish** to complete the deployment.
9. After you deploy the RAR file, under **Summary of Deployments**, click the name of the RAR file that you deployed.

   The Settings page is displayed.
10. Change the value of **Deployment Order** field to a value that is *higher* than the default value. For example, `500`.

This ensures that the newly deployed RAR file is always loaded after the supporting classes are loaded by the Oracle WebLogic Server.

Handling the Deployment Plan When Working on a Remote Oracle SOA Server

If the Adminserver is running on computer A and the Oracle SOA server is running on computer B, you must copy the deployment plan file to computer B before you activate changes made on the Oracle SOA server.

If you try to activate changes *without copying* the deployment plan to the Oracle SOA Server computer, a `NullPointerException` is thrown.

Migrating Repositories from Different Environments

All the JCA files generated by the Adapter Configuration Wizard have a reference to the JNDI name. The reference is defined in the `weblogic-ra.xml` file, which is the adapter's deployment descriptor.

The JNDI name is the *key* when you want to migrate from a development environment to a test environment to a production environment.

You must update the `weblogic-ra.xml` file to have the same JNDI name in all three environments: development, testing, and production.

You should specify values for deployment time properties, such as retry interval and retry count, and then redeploy to testing environment or production environment.

The `weblogic-ra.xml` identifies the end point as a development EIS or testing EIS or production EIS. For example, consider that when running through the Database Adapter Service Wizard, you specify `eis/DB/custStore` as the JNDI name for the `createCustomer` service.

After modeling the composite by using this adapter service, you should deploy it to the development, test, or production environments without making any changes. But before you deploy, ensure that you have a corresponding JNDI entry for `eis/DB/custStore` in each of your various environments pointing to the right EIS instance.

To summarize:

- All JCA files reference the JNDI name as defined in the `weblogic-ra.xml` file
- You must update the `weblogic-ra.xml` file to have the *same* JNDI name in all your environments in which it is deployed.
- Use the `weblogic-ra.xml` deployment descriptor to specify values for deployment time properties, such as retry interval and retry count. This file also identifies the end point's environment.
- Before deployment, ensure you have a corresponding JNDI entry for the correct environment.

Message Ordering

This topic describes a process for making a one-way asynchronous BPEL process synchronous by adding an output message and a reply activity.

Message ordering can be achieved by making a BPEL process synchronous. When this is achieved, the resource adapter thread publishing a message to the BPEL engine is used for executing the entire BPEL instance.

For asynchronous BPEL processes, a pool of BPEL engine worker threads process received messages. Thus, it prevents a message ordering guarantee, as the duration of one BPEL instance is almost always be different from that of others.

Generally, a JCA resource adapter wizard only creates a one way (asynchronous) WSDLs, that is, WSDLs where the operation has only an input message. It is necessary to manually modify the wizard generated WSDLs, that is, changing it so that it becomes a request-response (synchronous) type WSDL with input and output messages.

The following is an example to create a XML schema type for a response (output) message that is generally not generated:

```
<types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://acme.com/adapterService/">
    <import namespace="http://TargetNamespace.com/adapterService/type"
        schemaLocation="adapterTypes.xsd" />
.
.
.
        <element name="empty">
        <complexType/>
        </element>
```

Define the WSDL message (in the adapter WSDL file) as shown in the following example:

```
<message name="ignoreMessage">
    <part name="empty" element="tns:empty"/>
</message>
```

Add an output message to the inbound WSDL operation as shown in the following example:

```
<portType name="Receive_ptt">
    <operation name="Receive">
        <input message="tns:payloadMessage"/>
        <output message="tns:ignoreMessage"/>
</operation>
</porttype>
```

Add an output element in the binding section as shown in the following example:

```
<binding name="Receive_binding" type="tns:Receive_ptt">
    <pc:inbound_binding />
    <operation name="Receive">
        <jca:operation .../>
        <input/>
        <output/>
</operation>
</binding>
```

Add a reply activity at the end of the business process logic in the BPEL Process as shown in the following example:

```
<variables>
<variable name="ignore" messageType="ns1:ignoreMessage"/>
.
.
.
<sequence name="main">
```

```
<receive partnerLink="ReceivePL" portType="ns1:Receive_ptt"
     operation="Receive"
    variable="Receive_1_Read_InputVariable"
    createInstance="yes"/>
    .
    .
    .
    <!-- processing -->
<invoke partnerLink="DB_Insert" portType="ns1:DB_Insert_ptt"
    operation="InsertCust" inputVariable="NewCust"/>
<reply partnerLink="ReceivePL" portType="ns1:Receive_ptt"
    operation="Receive" variable="ignore"/>
    .
    .
    .
    <!-- optionally more processing -->
</sequence>
```

If the resource adapter in the preceding example supports multi-threading inbound, configure / tune to only use one thread, because multiple threads break the message ordering guarantee due to their stochastic durations.

How Oracle JCA Adapters Ensure No Message Loss

This section describes how adapters ensure that messages are not lost.

Transactional adapters allow the Enterprise Information System (EIS) to participate in one-phase or two-phase commits (local transactions or global/distributed transactions).

Non-transactional adapters implement their own schemes to ensure delivery, without the use of transactional semantics.

XA Transaction Support

The goal of XA is to allow multiple resources (such as databases, application servers, message queues, transactional caches) to be accessed within the same transaction. XA uses a two-phase commit to ensure that all resources either commit or rollback any particular transaction consistently.

The XA specification describes what a resource manager must do to support transactional access. Resource managers that follow this specification are said to be XA-compliant.

XA transactions are part of the scenario you use when you want to work with multiple resources: for example, or two or more databases, or a database and a JMS connection, or all of these plus the adapter, all in a single transaction.

Transactional adapters enable XA transaction support, which, along with the inherent data processing, ensures that each modification has a clearly defined outcome, resulting in either success or failure, thus preventing potential corruption of data, It ensures execution independently from other changes, and, when completed, leaves underlying data in the same state until another transaction takes place.

XA is a two-phase commit protocol, more robust than a one-phase commit or emulated protocol. With a one-phase, or emulated, protocol, you can see message loss or other rollback/commit inconsistency.

Local Transactions and Global (XA) Transactions

An XA transaction is a transaction started by an application server's transaction manager. All XA resources must participate in any active global transaction, and only commit or rollback when provided a signal by the transaction manager. If a failure to commit occurs after the signal is received, a recovery mechanism must also exist to ensure the commit eventually happens.

A non-participating local resource can start and end a local transaction irrespective of an active global transaction. The commit can be done immediately and is not in response to a signal from the transaction manager. If the commit fails, the transaction is rolled back instead, with an exception thrown. No special recovery is required for that transaction because there is no other resource with which to synchronize its commit.

Adapter Support of Local Transactions

Adapters define the type of transaction support by specifying the transaction-support element in the `ra.xml` deployment descriptor file.

Adapter Support of Global Transactions

Adapters support global transactions in the JCA 1.5 XA contracts that leverage the *underlying application server transaction manager.*

The types of adapters that leverage the underlying application transaction manager includes Oracle Adapter for Oracle Applications, Database, Advanced Queuing, JMS and MQSeries Adapters.

Non-transactional adapters, which *do not* leverage the underlying transaction manager, include Oracle File Adapter and Oracle FTP Adapter.

Global Transactions, Retries and Rollbacks and Fault Policies

A global transaction can be marked *rolled back* by any parties that participate in the global transaction. Once a party marks the global transaction for rollback, other parties cannot revoke the rollback,

The fault type indicates if the errors are retryable. If retryable, the retries are governed by the JCA retry properties. Refer to the error handling section. If the error is deemed unretryable, the handling of such an error is governed by the fault policy, in which case the fault policy gets executed. This is the same for both inbound and outbound adapters.

Actions performed by a fault policy are in its local transaction and not in the global transaction.

Specifically, the fault policy, running in its own transaction, commits any existing JTA transaction before it starts executing a particular Reference (for example, in Oracle BPEL PM it is an Invoke activity). The pre-existing JTA transaction is not suspended and then committed.

Exercise care when using non-transactional adapters, including Oracle File Adapter and Oracle FTP Adapter, with transactional adapters, as retries can affect non-transactional data, including creating duplicate messages. The type of care you must exercise can include, for example, modelling business processes so message duplicates do not occur.

Inbound synchronous request response scenarios that have a dequeue and enqueue response to reply to a queue are not candidates for manual recovery. Consequently, they are marked as failed once retries are exhausted.

Basic Concepts of Transactions and Adapters

For additional information on topics related to retryability, see Handling Rejected Messages , and following sections.

- **Polling**: All Oracle JCA Adapters and legacy adapters, support a pull, or polling, model for connecting to the back-end application for receiving events, that is, periodically querying the EIS endpoint for available messages and data.The exception to this is the Oracle Socket Adapter, which uses a different set of logistics, where the socket adapter can either connect to the EIS endpoint as the other adapters do using a client socket (polling), or, alternatively, create a server socket and then wait for incoming requests (push.) With polling, connection-related issues are recoverable and the inbound adapters keep retrying until the adapters are able to establish connection with the EIS. The adapter endpoints attempts to recover a lost connection for the duration of the active life of the composite. During this time they also update the log with diagnostics pinpointing the issue with connection

- **Local retry**: These are typically transient connectivity errors. where retries can be tried again and data is not compromised by a retry. However, non-successive local retries can change transaction state. Examples of retryable errors include temporary permission errors or resource constraint errors, or both. If a transaction can be retried, this does not necessarily mean a rollback.

- **Global retry**: A transaction that is rolled back to the beginning of the composite, for example, to a BPEL Receive where BPEL is part of the composite, which is at the beginning of the BPEL flow within a composite application. The transaction can be retried as indefinitely, or as many times as `jca.count.retry` indicates. Prior to the retry, a rollback can occur. An example could be where there is a BPEL fault in a synchronous process, or where there is a partial update to a database with master and child records and a temporary database fault occurs, and the toplink mapping logic des a retry is acceptable. In other words, a global retry can occur if data is not tainted and it can be considered an explicit retry, where a rollback is required.

- **Not-retriable**: A transaction that is not retried. With not-retriable conditions, there is no change to existing state. No-retry conditions derive from binding faults. Not-retriable situations typically occur where database integrity is an issue. Hence, not-retriable transactions are rolled back, when rejected; they are typically related to database constraint issues. Errors such as `Data already exists` (for example, Primary Key Errors) are not retryable as are message correlation ID errors. A list of errors that are not retryable is provided later in this chapter.

- **Inbound transaction**: A transaction initiated by an inbound adapter. For example, a transaction entering the SOA system from a JMS system.

- **Outbound transaction**: A transaction outbound from the SOA system (and hence from an adapter). For example, a transaction that is made against a database outside the SOA system.

- **JTA transaction**: Every step of a process is executed within the context of a JTA transaction. A JTA transaction ensures that one or more operations execute as an atomic unit of work. See the previous section on XA.

- **Asynchronous transaction**: A composite transaction composed of sub-transactions. However, these sub-transactions are consecutive and serialized, that is, some sub-transactions may have been committed while others may be still executing or have not yet executed. Asynchronous transactions are guaranteed to be propagated once and only once. When an update at the source is committed, the transaction commits and expects that the update is propagated to the target appropriately.
- **Synchronous transaction**: These are transactions that execute in one thread from one endpoint to another, without intermediate processes, and which are not serialized.

Asynchronous Transaction Flow

In the following sections, asynchronous and synchronous transactions are illustrated through a canonical combination set of adapters, JMS and DB, with BPEL technology intermediary. The example could employ other adapters, and other intermediaries, for example, the Mediator.

For an asynchronous service entry point, a transactional adapter initiates a global JTA transaction before sending an inbound message to the composite.

Example using JMS, BPEL, DB Adapter and a Database

The example described next uses a test composite bound to the JMS adapter, which is bound to a composite bound in this example to BPEL which in turn is wired to a DB Adapter. BPEL dispatches messages to the DB adapter.

In this example, messages are read from JMS by the polling JMS Adapter and written to the BPEL process, where there the transaction commits. This is JTA1, the first XA transaction.

For any BPEL activity errors that, however, could *not* be retried or *which exhausted their retry count*, BPEL writes to its recovery table to store information. This information includes BPEL errors.

The second transaction, JTA2, begins with the DB Adapter reading from the BPEL dispatch table, obtaining the database insert argument. and writing an update message to the DB Adapter. This transaction, JTA 2, proceeds Outbound from the reference endpoint DB Adapter (that is, Outbound from SOA) to the Database itself. Retry situations from a duplicate data situation in the Database are retried either back from the DB Adapter to BPEL's table, or from the database back to the DB Adapter.

Global retries for any error handling are returned to the BPEL Receive activity instance, for example, or, more generally, to the point at which the transaction started. Such a retry could occur if there was an error such as a temporary database fault. The default retry count is by default *indefinite*, or specified in the `jca.retry.count` property.

If any errors are caught as part of the second XA transaction, JTA2, a rollback occurs.

Synchronous Transaction Flow

For a synchronous process, the *global* transaction initiated by the adapter spans both:

- Message delivery
- Composite execution

As with asynchronous transaction flow, the default retry count is indefinite, but can be specified through `jca.count.retry`.

Synchronous transaction flow is similar to the asynchronous flow, with these differences:

- Flow consists of request-response messages between the JMS Adapter and intermediary processing, for example, BPEL processing, and between, using the same example, BPEL and the Database Adapter, where messages requesting, for example, an insert are written. With a *synchronous* transaction, a retryable error is *not* caught by BPEL (the example intermediary) within the composite; the transaction returns all the way back to the JMS adapter for possible global retry.
- The synchronous transaction is just *one* JTA transaction, rather than two.
- The Adapter rejection table keeps a record of adapter rejections. Within the context of a synchronous transaction, local BPEL error handling is bypassed, and with a synchronous transaction, the *private BPEL table* does not contain relevant Adapter rejection data. The data is instead kept in the Adapter rejection table.
- Local retries that exhaust the retry count are stored in the BPEL recovery table.

Using a similar example as that used in the synchronous example, and keeping in mind that an example synchronous message flow, parallel to the one used in the asynchronous example, consists of only *one* JTA transaction, JTA 1, throughout the transaction, processing is straightforward. The transaction starts with a polled message Inbound to the service endpoint, a JMS read message that then writes to the BPEL process.

Unlike the situation with the asynchronous transaction, with a synchronous transaction, the JTA transaction does not commit at this point.

Instead, the same JTA transaction proceeds Outbound from the reference endpoint DB Adapter to the Database itself. The message is then read from BPEL, and the DB Adapter is invoked with the insert argument from BPEL.At this point the JTA transaction commits.

Similar to asynchronous transactions, retries can be global and subject to a count indicated in the `jca.retry.count` property. In this example, faults which are locally retryable are tried either from the database back to the BPEL process or from the Database back to the DB Adapter.

Inbound Transactions

Inbound the adapter runs in an autonomous work thread; the adapter is in charge of connection recovery, and uses its own retry properties (for example, `adapter.jms.retry.interval`).

A transactional adapter initiates a global JTA transaction before sending an inbound message to a composite.

For transactional adapters, retries can either be local retries (for example, a BPEL remote fault), global, or no retry (similar to a binding fault). Global retries are returned to the location where the transaction started. The default retry count is again, by default, indefinite, but are retriable only as the `jca.retry.count` specifies.

When control returns to the adapter, the adapter commits the JTA transaction, and executes the following set of actions as an atomic unit of work.The adapter:

- Commits the removal of the message from the inbound adapter endpoint (for example, table and queue).
- Commits the execution of the composite instance.

If anything fails during this set of commit actions, that is, in removing the message and executing the composite instance, both actions are rolled back.

Outbound Transactions

All outbound transaction composite activities, including Oracle JCA adapter invocations, are part of a *global* transaction, and if an error occurs the default behavior is that all activities are either committed or rolled back.

For example, a BPEL process can insert data into several tables (on different databases) through different Invoke activities (invoking the Database adapter).

When the BPEL instance is about to finish, the JTA transaction is committed.

Only at that point are the database insert operations be committed.

However, *if errors occur during the BPEL instance execution*, all activities (and thus database operations) *are rolled back* to the last BPEL dehydration point (the last time the BPEL instance was stored to a database.)

Whether an outbound transaction is retryable depends on the nature and scope of a specific interaction. Specifically:

- Interactions that involve integrity, for example, database integrity, on the target side of the Outbound transaction, are not retried.
- There can be local retries where a locally retriable condition exists, for example, a minor database issue with a single record.
- If the retry situation is a more complicated database integrity scenario that could possibly be corrected, for example, an issue with updating both a Master Detail and a child record, the transaction might be rolled back to its beginning, back to a BPEL Receive (if BPEL were part of the scenario), and the transaction started again. The retry is again subject to `jca.retry` but also could be subject to any BPEL fault handling retry parameters.
- Connectivity issues outbound from an adapter are typically always retryable. For an outbound transaction, the adapter throws a retryable exception when it cannot get a connection, and then lets the appropriate JCA binding conduct retries (through `jca.retry.count).`

An example for a connectivity retryable error related to an outbound interaction is where a database listener might not have started and, accordingly, that state might be issuing connection errors.

Composite Availability and Inbound Adapters

Oracle WebLogic Server migration is used on WebLogic platform so that if a managed server fails, the server automatically restarts on the same or another physical system and inbound adapters specific to a composite on the failed server resume functioning.

Meanwhile, inbound adapters in other cluster members continue working servicing messages.

For more information, see:

- Singleton (Active/Passive) Inbound Endpoint Lifecycle Support Within Adapters
- File and FTP adapters: High Availability
- Database adapter: High Availability

- MQ adapter: High Availability
- How Oracle JCA Adapters Ensure No Message Loss

Singleton (Active/Passive) Inbound Endpoint Lifecycle Support Within Adapters

The JCA Binding Component supports active fail over of inbound Adapter Services.

To enable this fail over feature for a given inbound adapter endpoint, you must add the `singleton` JCA service binding property in the `composite.xml` within the `<binding.jca>` element and set it to a value of `true` as the code sample below shows.

To disable this feature, set the `singleton` property to a value of `false` (or remove the property from the `<binding.jca>` element).

Example - singleton Property in composite.xml

```
<service name="JmsTopicSubscr" ui:wsdlLocation="JmsTopicSubscr.wsdl">
    <interface.wsdl interface=
         "http://xmlns.oracle.com/
           ...#wsdl.interface(Subscr_ptt)"/>
    <binding.jca config="JmsTopicSubscr_file.jca">
        <property name="singleton">true</property>
    </binding.jca>
</service>
```

Multiple Activations of the Same Adapter Endpoint

In an Oracle WebLogic cluster, *multiple* activations of the same (for example, JMS) adapter (inbound) endpoint (for a specific composite service) are detected implicitly and automatically by *all* instances of the adapter framework active in that cluster.

However, only *one* activation is allowed to *start* the reading or publishing of messages.

The JCA Binding Component instances choose one among the activations, randomly the activation that assumes the Primary Activation responsibility.

Hot-Standby State

The other activations (also called instances) in the Oracle WebLogic cluster initiate to a hot stand-by state, without invoking `EndpointActivation` on the JCA resource adapter. These activations can be reassigned primary activation responsibility.

If a primary activation at some point becomes unresponsive, is deactivated manually, or crashes or exits, any of the remaining JCA Binding Component members of the Oracle WebLogic cluster immediately detect the deactivation, and reassign the primary activation responsibility to an activation agent that is in stand-by state.

For more information, see Composite Availability and Inbound Adapters.

Correlation Support Within Adapters

You can use Native Correlation to correlate an inbound asynchronous message with a previous outbound message, by defining a callback interface (for a Reference) or by a mid-process BPEL Receive:

For example, the following composite defines such a correlation:

```
<reference name='Outbound'>
<interface.wedl
interface="http://xmlns.oracle.com/pcbpel/demo#wsdl.interface
(JMSOutbound_PortType)"
callbackinterface=
"http://xmlns.oracle.com/pcbpel/demo#wsdl.interface
(JMSCallback_PortType)"/>
<binding.jca.operation="Consume"
        config="SampleOutbound_adapter.jca"/>
```

The `jca` file *must* contain both JCA interaction and JCA activation.

The correlation between the request and the response is done transparently by the JCA binding runtime.

For a JMS use case, the third-party application must copy the JMS message ID from the request message to the JMS CorrelationID of the response message.

For the Oracle AQ Adapter and Oracle JMS Adapter use cases, if an external application copies the `MessageId` from the request (Invoke) message to the `CorrelationId` of the response (Receive) message, the adapter framework ensures that the BPEL correlation occurs.

CorrelationID of Receive Message Not Matching Invoke: Log Error Message

However, when the `CorrelationId` of the Receive message does *not* match any earlier Invoke message, the message is mapped to a BPEL conversation that does not actually exist.

In this case, although the message is persisted in the database, the SEVERE log message can occur, as the code sample below shows:

Example - Log Error When CorrelationId of the Receive Does not Match any Earlier Invoke

```
SEVERE: JCABinding=>  aqadapter
            aqadapterAdapter Service aqadapter was
unable to perform delivery of inbound message
to the composite ... due to: Cannot
simply post callback message to the composite
 as there is no
 service element associated with the callback.
        Recommendation:
add/set the JCA reference/binding property
          'rejectUncorrelatedMessages' to true ...
SEVERE: JCABinding=>  aqadapter Unable to
create/save Composite Instance Fault due to: null
```

Rejecting Nonmatching Native Correlation IDs

You can explicitly alter the adapter framework behavior so that it *rejects* nonmatching native correlation IDs by adding the `rejectUncorrelatedMessages` service binding property to the `composite.xml` file as shown in code sample below.

Example - Setting the rejectUncorrelatedMessages Property

```
<reference name="ReqReply" ui:wsdlLocation=
        "ReqReply.wsdl">
    <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel
/adapter/mq/MQAsyncSol_ReplyQ_
NonRelatedMsg/SOA_AsyncSol_ReplyQ_NonRelatedMsg/
```

```
      ReqReply/#wsdl.
interface(Enqueue_ptt)"
callbackInterface="http://xmlns.oracle.com/
                 pcbpel/adapter/mq/
MQAsyncSol_ReplyQ_NonRelatedMsg/
SOA_AsyncSol_ReplyQ_NonRelatedMsg/ReqReply/
          #wsdl.interface(Dequeue_ptt)"/>
    <binding.jca config="ReqReply_mq.jca">
        <property name="rejectUncorrelatedMessages">
          true</property>
    </binding.jca>
</reference>
```

When `rejectUncorrelatedMessages` is set to `true`, uncorrelatable Receive messages are rejected by the adapter framework; that is, the messages are pushed back to the publishing JCA resource adapter.

By default, this property is set to `false`.

For more information, see:

- Error Handling
- Oracle JMS Adapter Features
- Normalized Message Support
- Oracle JCA Adapter Properties.

Setting Payload Size Threshold

System resources are finite and have a threshold limit for processing. The Oracle SOA Suite, dependent on system resources, also has certain size limitations, largely due to the underlying resources beyond which the system cannot process incoming requests.

For example, Oracle JCA Adapters can process large payloads but the Oracle BPEL PM consumes huge memory when processing large payloads, which can cause `OutOfMemory` conditions and affect the whole system.

You must set the payload threshold for Oracle JCA Adapters to avoid errors such as `OutOfMemory`. Setting the payload threshold helps ensure that Oracle JCA Adapters process payloads that are less than the threshold limit and reject others that are not less than the threshold limit. This section provides information relative to your consideration of the relative size of payloads.

Payload Native Size

If the native size of the payload is available, then the pertinent adapters use the native size of the payload to limit the payload size below the threshold limit.

For example, with Oracle File Adapter, the native size of file polled) is available to the adapter, and if it is greater than the payload size threshold then the file is rejected.

If the native size of payload is not available, for example, as is the case for the Oracle Socket Adapter, the adapter must calculate the native size of the payload internally.

Native size can be determined internally if you use the native translation library to translate non-XML or parse serialized XMLs.

The Oracle Database Adapter does not rely on the translation framework but has a special built-in handling mechanism to calculate the size of native messages.

Caution:

In case of debatching with error recovery, payload size threshold must be used carefully. Payload size violations might lead to unwarranted rejections while skipping the stream in case of erroneous records.

Setting the Payload Threshold

You can set the payload threshold by using the knob exposed by Oracle JCA Adapters. The knob can be set in the `composite.xml` file as a binding property for the adapter service, as shown in the following sample, where the 1000 value is in bytes.

```
<binding.jca config="getMsg_mq.jca">
  <property name="payloadSizeThreshold" type="xs:string"
           many="false"override="may">1000</property>
</binding.jca>
```

Limitations on Payload Size Enforcement

Where the native size of the payload is not available and if the specific adapter does not use the native translation library, you cannot enforce the payload size threshold limit. For example, in case of xml-debatching, where the Oracle File and FTP Adapters pass a chunk of file content and the actual native size is not known, payload size threshold limit cannot be used. Also, where there are serialized XML payloads and where XDK parser that lacks the feature to calculate native size is used for parsing instead of the native translation library, you cannot use payload size threshold limit.

XSD and Opaque translator scenarios can only be handled in adapters where the payload size is deterministic. For more information on the scenarios that are supported for specific Oracle JCA Adapters, refer to Table 2-2.

Table 2-2 Translation Scenarios Supported for Oracle JCA Adapters

Scenario	Oracle File and FTP Adapters	Oracle JMS Adapter	Oracle MQ Series Adapter	Oracle AQ Adapter	Oracle Database Adapter
NXSD	Yes	Yes	Yes	Yes	Not applicable
XSD	Yes	Yes	Yes	No	Yes
Opaque	Yes	Yes	Yes	No	Not applicable
DTD	No	No	No	No	Not applicable

Changing Global Payload Size to a Finite Value

Also, you can set the global property for capping payload size to change the default value of `payloadSizeThreshold` (from indefinite) to a finite number. In this case, where you set the default value of `payloadSizeThreshold` to a finite number, even if you do not explicitly configure a value for the `payloadSizeThreshold` property for a particular inbound adapter endpoint, the global default takes effect. If you specify the global default along with the value in `composite.xml`, then the value specified in `composite.xml` overrides the global value.

You can modify this global property using the MBeans browser (Adapter Mbean) of the Fusion Middleware Control. This change takes immediate effect for all current and future endpoints

Streaming Large Payload

Oracle JCA Adapters support large payload processing for both inbound and outbound processing. However, only the following adapters support the streaming feature explicitly:

- Oracle File Adapter

  For more information, see Scalable DOM.

- Oracle AQ Adapter

  For more information, see Stream Payload Support.

- Oracle JMS Adapter

  For more information, see Streaming Large Payload.

- Oracle Database Adapter

  For more information, see Streaming Large Payload.

The other adapters do not have explicit support for both.

Batching and Debatching Support

The batching and debatching functionality is supported for these adapters:

- Oracle JCA Adapter for Files
- Oracle JCA Adapter for FTP
- Oracle JCA Adapter for Databases

Oracle JCA Adapter for File and Oracle JCA Adapter for FTP consist of a `Reader` to debatch a single large file into several batches. You must specify the batch size during the design-time configuration. In addition, the adapter includes a Writer to batch a set of messages into a single file. For more information, see File Debatching.

Oracle JCA Adapter for Databases consists of a Publish component to poll a set of tables to detect events. This component can raise events to the BPEL process one record at a time or multiple records at a time. For more information, see Polling Strategies.

Adding an Adapter Connection Factory

The logical deployment of adapters implies the creation of `ConnectionFactory` objects in the `weblogic-ra.xml` deployment descriptor file. The `weblogic-ra.xml` file contains runtime connection parameters for an adapter.

To add the connection information and assign to a JNDI name, you must edit the corresponding `weblogic-ra.xml` file of the resource adapter by either using Oracle WebLogic Server Administration Console or WLST scripts.

The following steps describe how to set up a Database connection factory in the Oracle WebLogic Server Administration Console.

This section includes the following topics:

- Creating a Data Source
- Creating a Connection Pool

Creating a Data Source

To create a data source:

1. Navigate to `http://servername:portnumber/console`.
2. Use the required credentials to open the Home page of the Oracle WebLogic Server Administration Console.

   The Home page of the Oracle WebLogic Server Administration Console is displayed.
3. Under **Domain Structure**, select **Services**, **JBDC**, and then click **DataSources**.

   The **Summary of JDBC Data Sources** page is displayed.
4. Click **New**. The **Create a New JDBC Data Source** page is displayed.
5. Enter the following values for the properties to be used to identify your new JDBC data source:
   - **Name**: soademoDatabase
   - **JNDI Name**: jdbc/soademoDatabase
   - **Database Type**: Oracle

     Retain the default value for Database driver.
6. Click **Next**. The **Create a New JDBC Data Source Transaction Options** page is displayed.
7. Click **Next**. The **Create a New JDBC Data Source Connection Properties** page is displayed.
8. Enter the connection properties in the **Connection Properties** page, and then click **Next**.

   The **Create a New JDBC Data Source Test Database Connection** page is displayed.
9. Click **Test Configuration** to test the database availability and the connection properties you provided. A message stating that the connection test succeeded is displayed at the top of the **Create a New JDBC Data Source Test Database Connection** page.
10. Click **Next**. The **Create a New JDBC Data Source Select Targets** page is displayed.
11. Select a target, and then click **Finish**. You have created a data source.

    The **Summary of JDBC Data Sources** page is displayed. This page summarizes the JDBC data source objects that have been created in this domain. The Data Source that you created is displayed in this list.

Creating a Connection Pool

To create a connection pool:

1. Navigate to `http://servername:portnumber/console`.
2. Use the required credentials to open the Home page of the Oracle WebLogic Server Administration Console.

   The Home page of the Oracle WebLogic Server Administration Console is displayed.
3. Under **Domain Structure**, click **Deployments**.

   The **Summary of Deployments** page is displayed.
4. Click the Database adapter name from the Deployments list.

   The **Settings for DbAdapter** page is displayed.
5. Click **Configuration** tab, and then click **Outbound Connection Pools** tab.

   The **Outbound Connection Pool Configuration Table** is displayed.
6. Click **New**.
7. Select **javax.resource.cci.ConnectionFactory**, and click **Next**.

   The **Create a New Outbound Connection** page is displayed.
8. In the **JNDI Name:** field, enter `eis/DB/soademoDatabase`.

   Note:

   The JNDI value that you enter in this step is different from the same value that you entered in Step 5 in Creating a Data Source. The JNDI name specified in this step must match the value you enter in your database connection you create when building your application later.

   ---
9. Click **Finish**.

   The **Settings for DbAdapter** page appears, showing a table of Outbound Connection Pool groups and instances for this resource adapter is displayed.

   The configuration changes that you made must be stored in a new deployment plan. You do this in the next step.
10. In the **Path** field, select or enter the path of a deployment plan file. The path must end with ".xml".

    Note:

    If the Adminserver is running on computer A and the Oracle SOA server is running on computer B, then you *must copy* the deployment plan file to computer B before you activate changes made on the Oracle SOA server.

    If you try to activate changes *without* copying the deployment plan to the Oracle SOA Server computer, a `NullPointerException` is thrown.

    ---

11. In the Properties field, enter the value for `xADataSourceName` as `jdbc/soademoDatabase`

12. Click **Save**.

    Note:

    The properties do not get saved when you click **Save** as mentioned in this step. Instead, you must press **Enter** in the keyboard to save the changes you made.

13. Under Domain Structure, click **Deployments**.

    The Summary of Deployments is displayed.

14. Perform the following steps:

    a. Select the **DbAdapter** check box, and then select **Update**.

       The **Update Application Assistant** page is displayed.

    b. Select the **Update this application in place with new deployment plan changes** option.

    c. Click **Next**, and then click **Finish**.

       The **Summary of Deployments** page stating that the deployment you selected is updated is displayed.

15. Under Domain Structure, click **Deployments**, **DbAdapter**, **Configuration**, and then **Outbound Connection Pools**.

    Notice that the value of the `xADataSource` property that you entered in Step 11 is displayed in the Connection Factory Interface tab.

    Note:

    If you are adding a new value for the outbound connection pool, then you do not have to restart the Managed server or the Admin server. However, if you edit any property of an existing connection pool, you must restart the server.

Adding or Updating an Adapter Connection Factory

You can add a new adapter connection factory or update an existing adapter connection factory.

If you add or update an adapter connection factory, you must perform any of the following procedures to ensure that the composite uses the new adapter connection factory properties.

Follow the steps:

Modify the JCA File

Modify the JCA file of the deployed composite to point to the new JNDI. The composite takes the properties from the newly-created JNDI.

Use a Config Plan

- 1. Create a new JNDI for a JCA adapter connection factory.

  2. Create a Config plan for the composite.

     To create a Config Plan, right-click composite.xml in the JDeveloper design area. From the menu that appears, click Generate Config Plan. The Config Plan is generated.

  3. Specify a logical name for the JNDI in the JCA file.

     For example, in the following sample, `jndi-name` is the logical JNDI name:

     ```
     <connection-factory location="jndi-name" adapterRef=""/>
     ```

  4. Replace the logical name with the absolute value of the new JNDI in the Config plan.

     For example, in the following sample, the logical JNDI name, `jndi-name` is replaced by the absolute value, `eis/MQ/MQSeriesAdapter7`:

     ```
     <wsdlAndSchema
            name="DQ1.wsdl|DQ1_mq.jca
                        |EQ1.wsdl|EQ1_mq.jca|monitor.config">
           <searchReplace>
                 <search>jndi-name</search>
                 <replace>eis/MQ/MQSeriesAdapter7</replace>
           </searchReplace>
          </wsdlAndSchema>
     ```

When a composite uses new adapter connection factory properties, you must perform the following steps to avoid an Oracle Containers for Java EE restart:

1. Log into the Home page of the Oracle WebLogic Server Administration Console.
2. Select **Deployments** in the Domain Structure pane.

   The **Oracle WebLogic Server Administration Console Summary of Deployments** page is displayed.
3. Select the adapter for which you added a new connection factory.
4. Click **Update**.

   The **Update Application Assistant** page is displayed.
5. Select the **Update this application in place with new deployment plan changes** option.
6. Click **Next**, and then click **Finish**.

   The **Summary of Deployments** page stating that the deployment you selected is updated is displayed. You can use this procedure to change adapter endpoints, for example, without having to perform a restart.

Use the Web Logic Server Console to Create a New Connection

You can use the Web Logic Console to create connection factories for use with JMS. Refer to Creating a New Connection by Using the Oracle WebLogic Server Administration Console

Recommended Setting for Data Sources Used by Oracle JCA Adapters

This section describes the recommended setting for non-XA and XA data sources used by Oracle JCA Adapters.

The following are the recommended settings for multi-data sources:

- `test-frequency-seconds` should be 5
- `algorithm-type` should be Load-Balancing
- `data-source-list` should point to a list of comma-delimited child data sources. For example, `(JDBC Data Source-0,JDBC Data Source-1)`

If your endpoint property resides in an Oracle RAC database, use multi-data sources.

Table 2-3 lists the recommended setting for XA and non-XA data sources used by Oracle JCA Adapters.

Table 2-3 Recommended Setting For XA and Non-XA Data Sources

XA Data Sources	Non-XA Data Sources
The driver used is `oracle.jdbc.xa.client.OracleXADataSource`.	The driver used is `oracle.jdbc.OracleDriver`.
The JDBC URL should be in the following format: `jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=host-vip)(PORT=1521))(CONNECT_DATA=(SERVICE_NAME=service_name)(INSTANCE_NAME=inst1)))`	Same as that of XA data source.
You must set the following property `<property><name>oracle.net.CONNECT_TIMEOUT</name><value>10000</value></property>`	Same as that of XA data source.
The value of `initial-capacity` must be 0	Same as that of XA data source.
The value of `connection-creation-retry-frequency-seconds` must be 10	Same as that of XA data source.
The value of `test-frequency-seconds` must be 300.	Same as that of XA data source.
The value of `test-connections-on-reserve` must be `TRUE`.	Same as that of XA data source.
The value of `test-table-name` must be `SQL SELECT 1 FROM DUAL`	Same as that of XA data source.
The value of `seconds-to-trust-an-idle-pool-connection` must be 0	Same as that of XA data source.
The value of `global-transactions-protocol` must be `TwoPhaseCommit`	The value for `global-transactions-protocol` must be `None`.

Table 2-3 (Cont.) Recommended Setting For XA and Non-XA Data Sources

XA Data Sources	Non-XA Data Sources
The value of `keep-xa-conn-till-tx-complete` must be `TRUE`.	NA
The value of `xa-retry-duration-seconds` must be 300.	NA
The value of `xa-retry-interval-seconds` must be 60.	NA

Note:

The settings mentioned in Table 2-3 are applicable to both types of database, single instance and an Oracle RAC database. In case of an Oracle RAC database, these settings must be used for constituent data sources for multi data sources created for endpoints. See the Oracle RAC Documentation at http://www.oracle.com/technetwork/database/options/clustering/documentation/index.html

In addition to applying the settings mentioned in Table 2-3, you must perform the steps documented in "Using Oracle Thin/XA Driver" in the *Developing JTA Applications for Oracle WebLogic Server*

These steps are required for data sources using XA driver. After performing the steps mentioned in the preceding link, you must run the following SQL statements to enable WLS JTA recovery to work:

```
grant select on sys.dba_pending_transactions to public
GRANT FORCE ANY TRANSACTION TO public
grant execute on sys.dbms_xa to public
```

Error Handling

The Oracle JCA Adapters provide error handling capabilities, as listed in the following sections. These rejection handlers are applicable in synchronous processes *only*. They do not apply to asynchronous or one-way processes.

This section includes the following topics:

- Inbound Interaction Error Handling
- Outbound Adapter Interaction Error Handling
- Handling Message Errors: A Sample Scenario

Handling Rejected Messages

The messages that error out before being posted to the service infrastructure are referred to as *rejected messages*. For example, the Oracle File Adapter selects a file having data in CSV format and tries to translate it to XML format (using NXSD). If there is any error in the translation, this message is rejected and are not be posted to the target composite.

Primarily, adapters and binding components are the generators of rejected messages.

How Errors or Faults that Arise Downstream are Handled

Errors or faults that arise downstream in a synchronized flow are handled in the following manner by the inbound adapter:

- *Immediately* rejected if the exception is non-retryable.
- Retried *indefinitely* if the exception is retryable.
- Retried *several times* equal to the value of `jca.retry.count` (if configured) and then *rejected* when the retries are exhausted.

Adapters reject messages that create errors at the binding level; that is, they error out before entering the Service Infrastructure layer.

All rejected messages are stored in the database with the payload. The rejected messages can later be queried against.

Faults can be also categorized as binding or remote. Remote faults are typically attributed to transient failures such as database connection down and are retriable where as binding faults are attributed to non-transient failures and are marked as non-retriable. But depending on the situation, these non-transient failures can also be rectified.For example, faults such as an MQ series Adapter target queue being full or unique constraint violation in a database can be rectified.However, certain non-transient failures, such as validation failure due to translation/transformation, cannot be rectified without redeployment of the composite.Messages rejected due to both binding faults and remote faults are marked as recoverable (for manual recovery).Recovery success depends on the scenario; you should inspect the faulted instance and initiate action if the manual recovery fails.

This section includes the following topics:

- Configuring Rejection Handlers
- Checking for Rejected Messages

For more information, see Correlation Support Within Adapters.

Configuring Rejection Handlers

In the 10.x release, rejection handlers were defined in the deployment descriptor (`bpel.xml`) of an Oracle BPEL process.

However, in the 11g release, you must define rejection handlers by using *fault policies*.

You can specify only one action handler for inbound rejection handlers.

Creating Fault Policies

You must create two files named `fault-policies.xml` and `fault-bindings.xml`, and copy them to the SOA project directory in JDeveloper, as described in the following steps:

1. Define a fault policy for the rejected messages in the `fault-policies.xml` file, stored with the `composite.xml` file in the JDeveloper project directory.

   The following is an example of a fault policy:

```
<?xml version="1.0" encoding="UTF-8"?>
  <faultPolicies>
```

```
<faultPolicy version="2.0.1" id="RejectedMessages">
  <Conditions> <!-- All the fault conditions
            are defined here -->
    <faultName xmlns:rjm="http://schemas.oracle.com
              /sca/rejectedmessages" name="rjm:
             <SERVICE_NAME>"> <!-- local part of fault
               name should be the service name-->
      <condition>
       <action ref="writeToFile"/> <!-- action
       to be taken, refer to Actions section
              for details of the action -->
      </condition>
    </faultName>
  </Conditions>
  <Actions> <!-- All the actions are defined here -->
      <Action id="writeToFile">
        <fileAction>
          <location>/tmp/rej_msgs</location>
          <fileName>emp_%ID%_
                %TIMESTAMP%.xml</fileName>
        </fileAction>
      </Action>
  </Actions>
 </faultPolicy>
</faultPolicies>
```

2. You must associate the fault policy with a service endpoint of the composite in `fault-bindings.xml`, as is done in the following example:

```
<faultPolicyBindings version="2.0.1"
xmlns="http://schemas.oracle.com/bpel
                  /faultpolicy"
xmlns:xsi="http://www.w3.org/2001/
                    XMLSchema-instance">
    ...
    <service faultPolicy="RejectedMessages">
        <name>Read</name>
    </service>
    ...
</faultPolicyBindings>
```

3. Copy the `fault-policies.xml` and the `fault-bindings.xml` files to your SOA composite project directory.

4. Deploy the SOA composite project.

Note:

If you do not configure rejection handlers as mentioned in Configuring Rejection Handlers, a default file-based rejection handler starts processing and the rejected messages is directed to `<domain_home>/rejmsgs/<wls_server_name>/<composite_name>`.

Also, you can configure rejected messages with a Mediator Component in the same fault policy as that of Oracle BPEL Process Manager (Oracle BPEL PM).

Checking for Rejected Messages

Rejected messages are stored in the `rejected_message` table.

You can check for rejected messages by using either of the following steps. You can obtain the messages and perform additional processing on them, according to your own implementation.

- Checking from the Database
- Checking from the Fusion Middleware Control Console

Checking from the Database

To check from the database, you must connect to the database as soainfra schema, and run the following SQL command:

```
select * from rejected_message
```

Checking from the Fusion Middleware Control Console

You can view the rejected messages in the **Recent Faults and Rejected Messages** section of the **Dashboard** tab or in the **Faults and Rejected Messages** tab.

For more information about using the Fusion Middleware Control Console for checking for rejected messages, see:

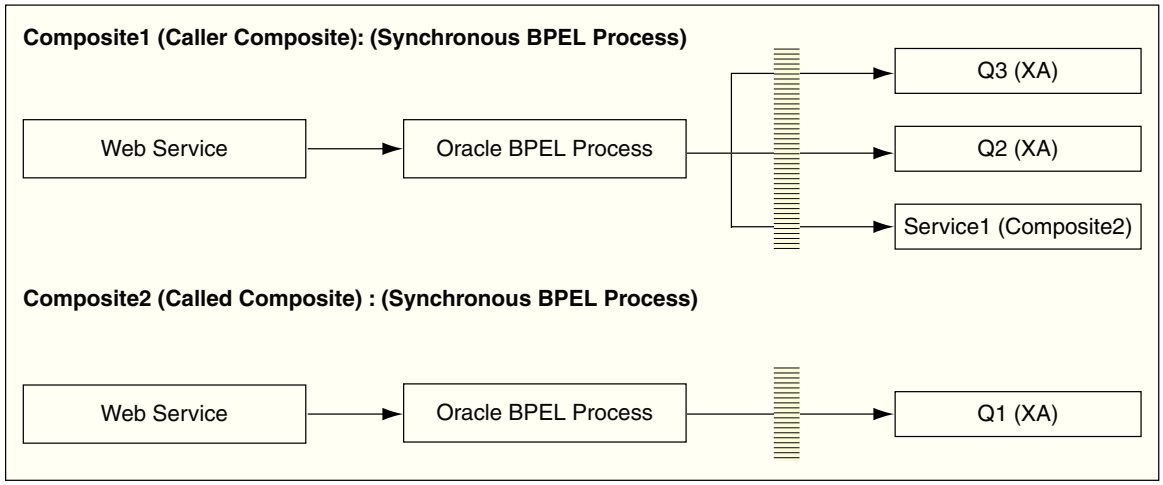
- "Monitoring Recent Faults and Rejected Messages for an Inbound Adapter" in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.
- "Monitoring Faults and Rejected Messages for an Inbound Adapter" in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*

Handling Message Errors: A Sample Scenario

This section describes how to handle message errors through a sample scenario.

There are two composites, Composite 1 and Composite 2 each having an Oracle BPEL process and there is a mix of local and XA resources, as shown in Figure 2-10.

Figure 2-10 Sample Scenario: Handling Message Errors



When the message is successfully delivered to all the queues (Q1, Q2 and Q3), the transaction commits successfully.

If the message cannot be delivered to Q1 (or to any queue) but the message is delivered to queues Q2 and Q3, the transaction must roll back *all the three messages* because *all* are XA resources and there is an exception in an XA unit.

The rollback exception is thrown *only* for the second composite where Q1 failed, and the transactions *commits Q2 and Q3* instead of rolling back the messages for all the three queues.

To have the transaction roll back *all the queues even if only one fails, and for the other two have messages successfully delivered to them*, you must make the change in the `composite.xml` file of the called composite (Composite2) as the code sample below shows:

Example - Changes in composite.xml of Composite2

```
<component name="BPELProcess1">
  <implementation.bpel src="BPELProcess1.bpel"/>
  <property name="bpel.config.transaction">
            required</property>
</component>
```

This sets the property `bpel.config.transaction` to the value of `required`, which causes the transaction to roll back *all* the queues even if only one fails.

If you set property `bpel.config.transaction` to a value of `required`, the Oracle BPEL engine effectively processes the synchronous request without creating a new transaction; rather, it uses the caller's transaction. Therefore, if at any point the transaction gets rolled back, nothing done in that transaction commits.

Inbound Interaction Error Handling

You can indicate the way inbound adapters should handle errors by specifying rejected message handlers.

Message Error Rejection Handlers

You can create rejection handlers to handle message errors. Message errors include those that occur during translation, correlation ID mismatch and XML parsing after message reception.

Available Rejection Handlers for Message Errors

Before considering error handling in terms of retryability, it is important to understand the error handlers that are available.

The following are the system-defined error handlers, which you can configure through fault policies:.

- Web Service Handler
- Custom Java Handler
- JMS Queue
- File

Web Service Handler

A rejected message can be handled by calling a Web Service. If you choose to use a Web Service to handle these errors, you should implement a predefined WSDL interface implemented by the target service, SOAP bindings for the Web service

invocation, and native payloads passed as WebService-attachments, as shown in the following example:

-
```
<Action id="ora-ws">
  <invokeWS uri="WebServiceURI"/>
<!-- format - <Absolute wsdl path>|service name|port name -->
</Action>
```

  The WSDL Interface for the Web Service handler must have one port type, only one input operation, and a schema for the input message. This is shown in the following example.

  Example - WSDL Interface for Web Service Handler

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://xmlns.oracle.com/pcbpel/errorHandling"
xmlns:tns="http://xmlns.oracle.com/pcbpel/errorHandling"
elementFormDefault="qualified">
  <element name="RejectedMessage" type="tns:RejectedMessageType">
  <complexType name="RejectedMessageType"/>
    <sequence>
      <element name="MessageHeader" type="string"/>
<!-- base64 encoded string -->
      <element name="MessagePayload" type="string"/>
<!-- base64 encoded string -->
      <element name="RejectionReason" type="string"/>
  </sequence>
    <attribute name="RejectionId" type="string"/>
  </complexType>
</schema>
```

Custom Java Handler

Another option to handle errors is to create a predefined Java framework, an interface, that forwards errors. You can implement a Java interface by the target class, as shown in the following example.

-
```
<Action id="ora-custom">
  <javaAction className="mypackage.myClass"
            defaultAction="ora-terminate">
    <returnValue value="SUCCESS" ref="ora-file"/>
    <returnValue value="FAILED" ref="ora-ws"/>
  </javaAction>
</Action>
```

  The interface itself specifies a fault recovery class. See the following snippet for an example of the interface.

```
package oracle.integration.platform.faultpolicy;
public interface IFaultRecoveryJavaClass
{
    public void handleRetrySuccess
            ( IFaultRecoveryContext ctx);
    public String handleFault( IFaultRecoveryContext ctx);}
```

JMS Queue

You cannot enqueue a rejected message to a JMS queue as a JMS message with the appropriate context and payload, however you can do this task with the AQ adapter as shown in the following two examples.

The first example uses a standalone database:

Example - Using a Standalone Database with the AQ Adapter to Enqueue a Rejected Message

```
<faultPolicies>
  <faultPolicy version="2.0.1" id="MQ_ServiceFaults_MED">
    <Conditions>
      <faultName xmlns:rjm="http://schemas.oracle.
               com/sca/rejectedmessages" name="rjm:DQ">
        <condition>
          <action ref="ora-enqueue"/>
        </condition>
      </faultName>
      </Conditions>
      <Actions>
      <Action id="ora-enqueue">
        <enqueue uri="jdbc:oracle:thin:
            @//myhost.com:1521/orcl#scott/tiger#REJ_MSG_Q" />
      </Action>
     </Actions>
  </faultPolicy>
</faultPolicies>
```

The second example is used with an Oracle RAC database:

Example - Using an Oracle RAC Database with the AQ Adapter to Enqueue a Rejected Message

```
<Action id="ora-queue">
  <enqueue uri="QueueURI"/> <!-- QueueURI format  -
jdbc:oracle:thin:@(DESCRIPTION=(LOAD_BALANCE=on)(ADDRESS_
LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=<host1>)(PORT=<port1>))(
ADDRESS=(PROTOCOL=TCP)(HOST=<host2>)(PORT=<port2>)))(CONNECT_DATA=(SERVICE_
NAME=<service_name>)))#<un>/<pw>#queue -->
</Action>
```

File

You create an error handler for messages by storing a rejected message in a file. You can store the payload with the proper context, as shown in the following example. The Payload file is created at the configured location.

```
<Action id="ora-file">
  <fileAction>
    <location>FOLDER_LOCATION</location>
    <fileName>FILE_NAME</fileName>
<!-- FILE_NAME will support %ID%(rejected message instance id) or %TIMESTAMP%
wildcards -->
  </fileAction>
</Action>
```

Error payload persistence in the Database is available by default. Only the File Adapter handler creates a metadata file that contains all the properties of the rejected message.

For example, for the Oracle File Adapter, this metadata file could include information such as the inbound direction and file name. The location of metadata file is same as the payload file and the naming pattern is `<FILE_NAME>_metadata`.

For resubmitting rejected messages, payload persistence is imperative. Payloads are stored in the Database and a facility to view the payloads is available through the Fusion Middleware Control Console. The message/payload is provided in full to each *configured* error handler, in addition to providing the payload to the *default* error handler.

Inbound Retryable Errors

Inbound retryable errors are typically *transient connectivity* errors. Only retryable errors for a synchronous process thrown by the outbound binding is subject to retry by the inbound adapter (an indefinite number of times by default, which is limited by setting the `jca.retry.count` property). Any JTA transaction is rolled back before a retry.

Examples of retryable errors thrown by outbound adapters include connection errors but include also temporary permission errors or resource constraint errors, or both.

Errors such as `Data already exists` (for example, Primary Key Errors) are not retryable. In addition, message correlation ID errors are not retryable.

When a set number of retries have been exhausted, the rejection mechanism handles the error.

Configuring Inbound Adapters to Handle Retryable Errors

You can configure inbound adapters to handle inbound retryable errors. The following properties, which you can specify in the `composite.xml` file, are supported for retryable exceptions for inbound interactions:

By default, there is unlimited retry for inbound errors; however, adapter retry is either at the level of the composite (local) application or at the global level.

Once you have configured properties in the composite, at the service level, the configuration of the properties has meaning. (For example, when you configure the number of retries before rejection, the value of the interval property takes its default value.)

Properties you can specify in the composite.xml file include:

- `jca.retry.count`

  Specifies the maximum number of retries before rejection. Again, specifying this value is a pre-requisite to specifying the other property values.

- `jca.retry.interval`

  Specifies the time interval between retries (measured in seconds.)

- `jca.retry.backoff`

  Specifies the retry interval growth factor (positive integer.)

- `jca.retry.maxInterval`

  Specifies the maximum value of retry interval, that is, a cap if backoff > 1

Specifying Inbound Retry Properties in the composite.xml File

You can modify the composite application's xml descriptor to specify properties that apply to retries. The preceding list of properties are specified in the composite.xml file in JDeveloper, as shown in the following example:

```
 <service name="Inbound">
   <interface.wsdl interface="http://
         xmlns...#wsdl.interface(Inbound_PortType)"/>
   <binding.jca config="Inbound_db.jca">
         <property name="jca.retry.count">5</property>
         <property name="jca.retry.interval">1</property>
       <property name="jca.retry.backoff">2</property>
       <property name="jca.retry.maxInterval">6</property>
     </binding.jca>
</service>
```

For retryable exceptions, you must set the value of `jca.retry.count` to the number of times the retry is to be carried out.

For example, if you set the value of `jca.retry.count` to 10, the retry occurs 10 times.

However, if you have *not* set any value for `jca.retry.count`, the retry is carried out *indefinitely*, which is the *default* for retryable errors.

Note:

Infinite retries by inbound adapters for errors results in the creation of multiple composite instances, because for every retry a separate composite instance is created.

Changing the Default Value of jca.retry. count for Inbound Adapter Endpoints

You can change the global property for capping retries to alter the default value of `jca.retry.count` from an indefinite to a finite number.

In this case, where you set the default value of `jca.retry.count` to a finite number, even if you do not explicitly configure a value for the `jca.retry.count` property for a particular inbound adapter endpoint, the global default takes effect.

If you specify the global default along with the value in the composite.xml, the value specified in the `composite.xml` overrides the global value.

You can modify the global property using the MBeans browser (Adapter Mbean) of the Fusion Middleware Control Console. Any change you do through the MBeans browser takes immediate effect for all current and future endpoints.

Global Property Modification using the MBeans Browser

To modify the *global property* using the MBeans browser (Adapter Mbean) of the Fusion Middleware Control, you must use the following procedure:

1. Navigate to `http://servername:portnumber/em`.

   The Fusion Middleware Control Console displays its home page.

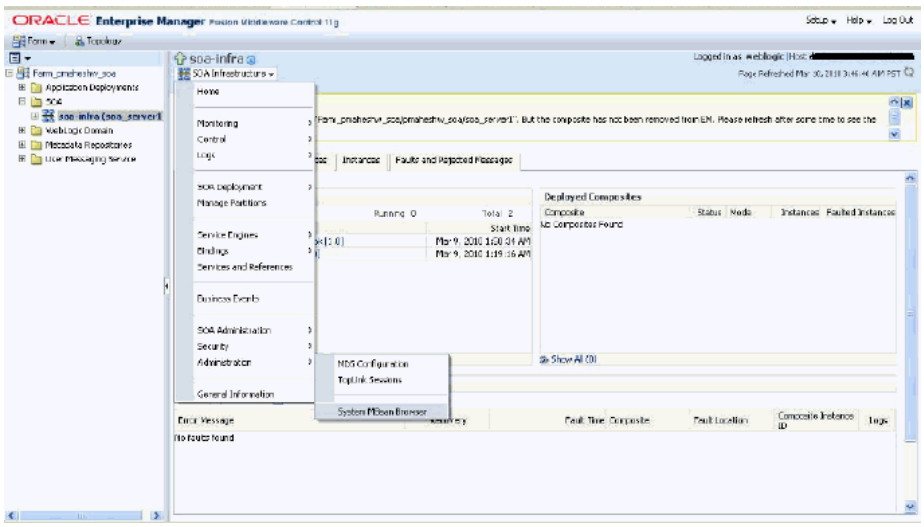
2. Right-click **soa-infra** from the SOA Folder in the navigator in the left pane.

   The soa-infra page is displayed.

3. From the SOA Infrastructure menu, select Administration, and then System Mbean Browser, as shown in Figure 2-11.

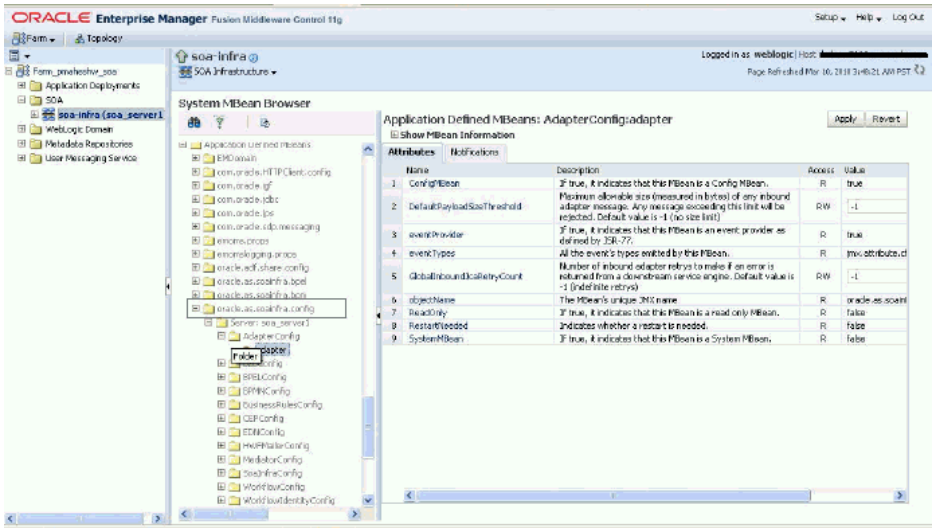
   The System Mbean Browser page is displayed.

Figure 2-11 The soa-infra Page



4. Select **oracle.as.soainfra.config**, **Server**, **AdapterConfig**, and then **adapter**, as shown in Figure 2-12.

Figure 2-12 The soa-infra Page: System MBean Browser



5. Modify the **GlobalInboundJcaRetryCount** attribute (as an example of a Global Property)

Inbound Non-Retryable Errors

Typically *non-retryable* errors are a result of either transformation or message parsing.

Inbound adapters handle non-retryable errors thrown from the Enterprise Information System by rejecting the inbound messages. If the error is a non-retryable error, you must use the rejection handler to handle the non-retryable error.

Examples of Non-Retryable Errors

Examples of non-retryable errors thrown from interaction with an Enterprise Information System include the following:

- Primary key violation
- Queue does not exist
- Master record does not exist
- Unable to serialize payload

Non-retryable errors are never retried because they are never expected to resolve themselves simply by being retried. For example, messages can be sent from a file to an inbound file adapter through a Mediator. The Mediator, in turn, has sequential routing to an outbound Database Adapter that inserts data to a database table. The DB adapter might encounter a unique *constraint error* as it is performing the insert operation. This unique constraint error is:

- Considered by the outbound Database Adapter as a *non-retryable error*
- Propagated back to the inbound Adapter
- Considered by the inbound adapter as a non-retryable error as well, using a rejection handler. The adapter uses a fault policy if one is defined.

A mediator could have errors on a transformation. This type of error is a *non-retryable* error. The error returns to the inbound adapter where it is handled, depending on the signature of the WSDL.

Outbound Adapter Interaction Error Handling

Outbound Interaction errors occur with messages that have interactions outbound from an adapter.

This section addresses the retryability and non-retryability of these Outbound Interaction errors and provides a basis for understanding the related properties you can set.

Retryable Errors for Outbound Adapter Error Handling

Outbound retryable errors can be retried based on the value of `jca.retry.count` in the `composite.xml` file.

Setting Retryable Properties for Outbound Error Handling in the composite.xml File

For retryable exceptions for outbound error handling, you must set the value of `jca.retry.count` to the *number of times* the retry is to be carried out.

For example, if you set the value of `jca.retry.count` to 10, the retry occurs 10 times.

However, if you have *not* set any value for `jca.retry.count`, the retry is carried out by the *fault policy*, if you have included the fault policy as part of the composite.

Example: How to Set Values for Retryable Exceptions for Outbound Interactions

The following code snippet is an example of how to set values in the `composite.xml` file for retryable exceptions for outbound interactions.

The retry is set to 5 minutes with an interval of 1 minute, and the other properties are appropriately configured. As stated before, the additional properties have meaning when the `jca.retry.count` property is specified.

```
<reference name="Outbound">
  <interface.wsdl interface="http://
              xmlns...#wsdl.interface(Outbound_PortType)"/>
  <binding.jca config="Outbound_jms.jca">
    <property name="jca.retry.count">5</property>
    <property name="jca.retry.interval">1</property>
    <property name="jca.retry.backoff">2</property>
    <property name="jca.retry.maxInterval">6</property>
    <property name="jca.retry.maxPeriod">30</property>
  </binding.jca>
</reference>
```

Non-Retryable Errors for Outbound Interaction Handling

You can handle non-retryable exceptions for outbound interactions by defining the maximum number of reconnection attempts that can be made in the `fault-policy.xml` file, which establishes the expected behavior for non-retryable errors.

In this fault policy file, you specify the parameters for reconnection attempts, as shown in the following example. This includes:

- The number of reconnection retries (`retryCount`)
- Intervals between reconnection retries (`retryInterval`)
- An exponential backoff value for the connection retries (`exponentialBackoff`)

All time measurements are specified in seconds.

Example - Specifying Parameters for Reconnection Attempts

```
<faultName xmlns:bplex="http://schemas.oracle.com/bpel/extension"
name='bplex;bindingFault">
            <condition>
            <action ref="ora-retry"/>
        </faultName>
            </condition>
          <Actions>
          <Action id="ora-retry">
            <retry>
                <retryCount>10</retryCount>
                <retryInterval>2</retryInterval>
                <exponentialBackoff>2</exponentialBackoff>
            </retry>
         </Action>
       </Actions>
```

You must associate a fault policy with a reference end point of the composite in `fault-bindings.xml` file, as shown in the following example, with the `faultPolicy` `ConnectionFaults` and the reference name `writeMessageToQueue`.

Example - Associating a Fault Policy with a Reference End Point of the Composite

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<faultPolicyBindings version="2.0.1"
xmlns="http://schemas.oracle.com/bpel/faultpolicy"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <reference faultPolicy="ConnectionFaults">
      <name>writeMessageToQueue</name>
    </reference>
</faultPolicyBindings>
```

After the configured number of retries is reached without a positive result, the Service Infrastructure Invocation exception is thrown.

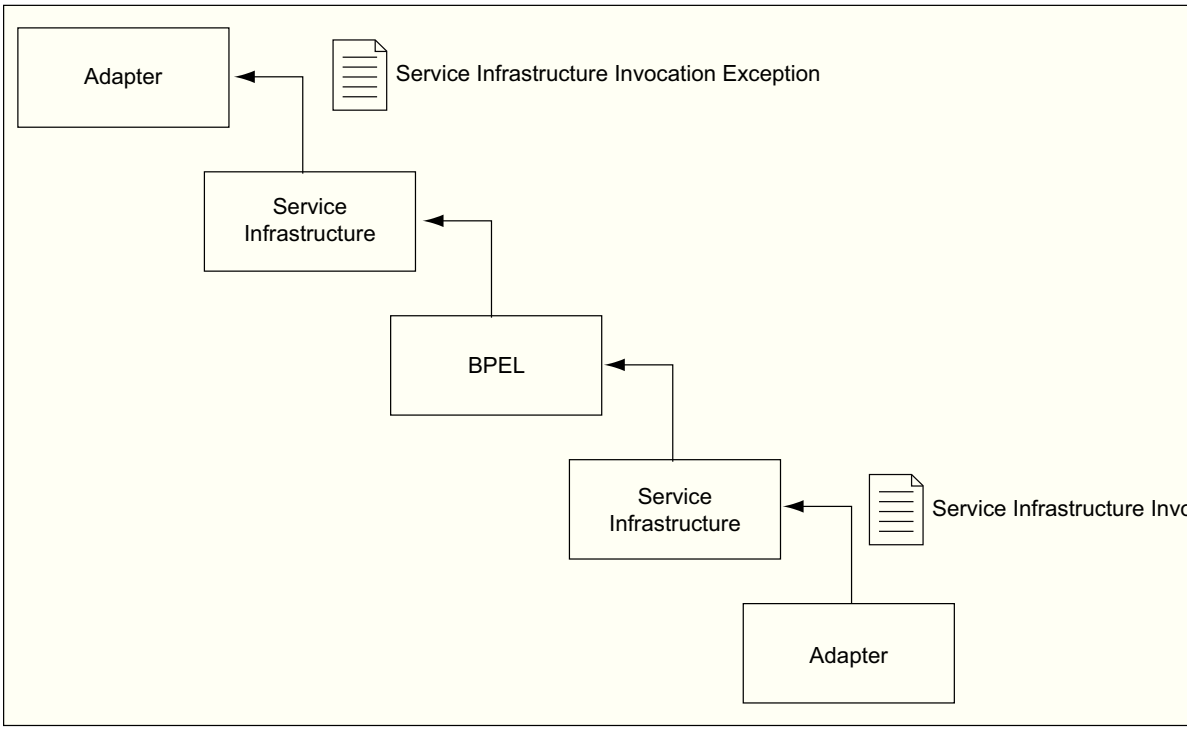
Fault Propagation

The propagation of the type of the Service Infrastructure Invocation exception is important to allow inbound adapters to respond to errors reported by outbound adapters.

Figure 2-13 shows the fault propagation when an adapter calls the service infrastructure synchronously, after which the Oracle BPEL Process Manager and Oracle Mediator calls a down-stream adapter.

In this figure, a Service Infrastructure Invocation exception propagates from the down-stream adapter, through Oracle BPEL Process Manager and Oracle Mediator, and to the caller adapter.

Figure 2-13 Fault Propagation



Two Cases When the Fault Policy Mechanism Does Not Work

There are two cases where the fault policy mechanism does not work:

- Outbound Adapters in XA Mode
- Outbound Adapters in Mediator Sequential Routing

Outbound Adapters in XA Mode

The fault policy mechanism does not work for outbound adapters in XA mode.

For example, in XA mode, if you want the fault policy to retry when the outbound adapter fails, it does not retry and any outbound adapter that has been successful before this failure occurred does not rollback messages.

Outbound Adapter i n Mediator Sequential

Fault policies also do not work for the outbound adapter that is invoked in Mediator sequential routing, because the mediator fault policies are applicable to parallel routing rules only.

Integrating JCA adapters with Oracle Web Services Manager to Protect Sensitive Data in Audit Trails

You can use Oracle Web Services Manager OWSM through JDeveloper and the Fusion Middleware Control to apply a PII (Personally Identifiable Information) policy to a payload in a scenario where an inbound adapter sends a message to a BPEL Process which feeds the message to an outbound Adapter.

Using OWSM in this manner, you can configure the Adapter payload to be encrypted and decrypted. An individual examining the BPEL Audit Trail does not see the data you specify to be encrypted.

You select any payload element from inbound to be encrypted; you must also select the same element to be decrypted for the corresponding endpoint.

In general, web services security includes several aspects: authentication, authorization (or Access Control), confidentiality and privacy, integrity, non repudiation.

The Adapter's use of this involves keeping information secret Personally Identifiable Information (PII).

Attaching the JCA Encryption on the Endpoint

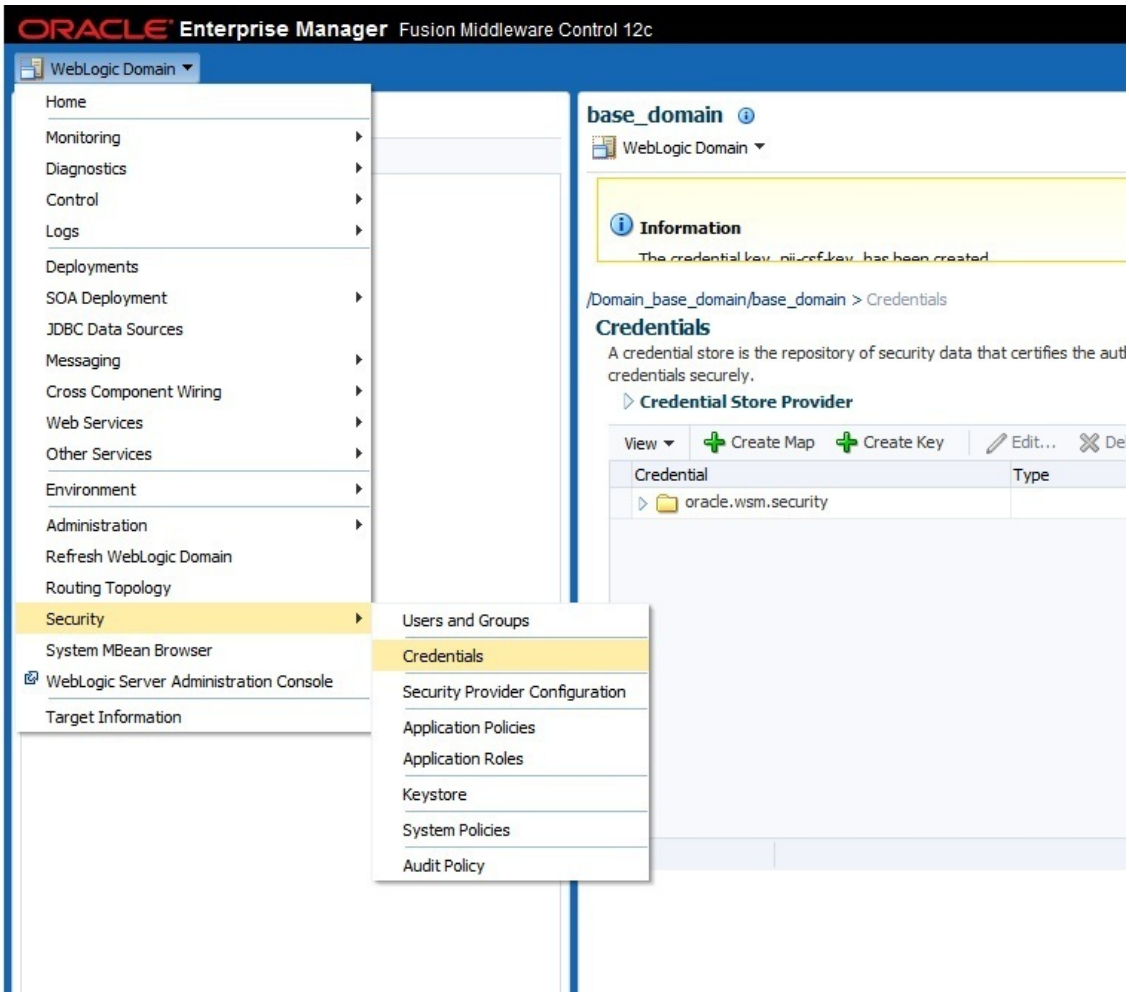
To apply a PII policy to a payload, you provide JCA encryption and decryption on the JCA endpoints. The goal of PII encryption is to keep data safe while it is inside the system.

To do this, you supply a security credential with its own map name and key name for the data at the Endpoint. The security credential you supply for the Endpoint is uniquely identified by this map name and key name you supply.

Use the following steps to attach JCA encryption and decryption on the Service and Reference Endpoints. The sample shown is a file service.

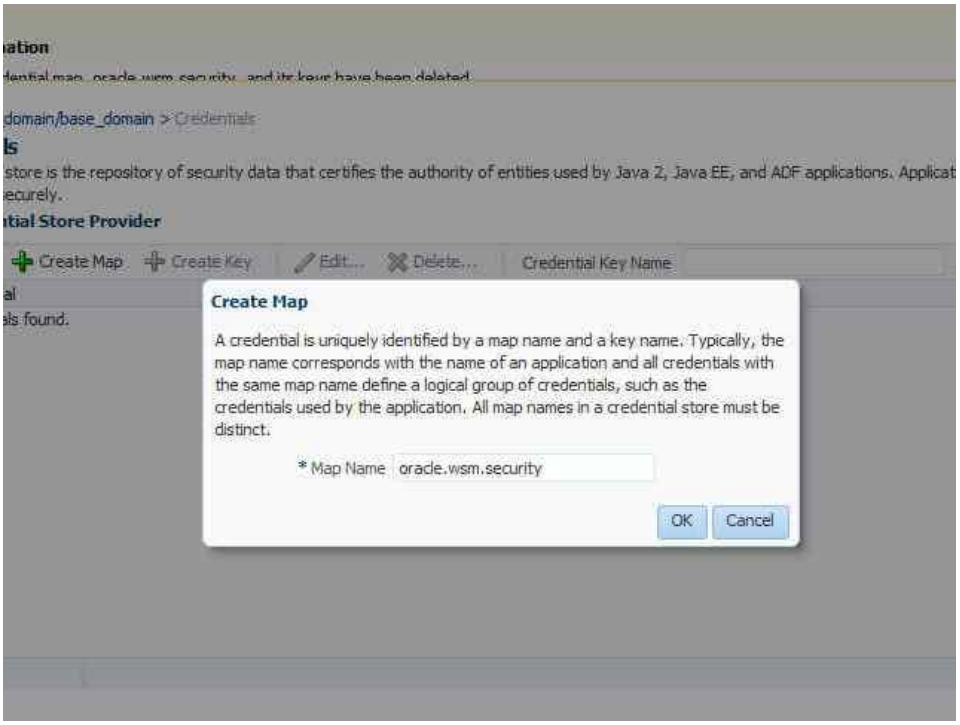
1. Log into Fusion Middleware Control.
2. Select **WebLogic Server -> Security->Credentials->Create Map**. Click **Create Map**. The **Create Map** dialog appears.

Figure 2-14 Navigating to the Credential Store and Beginning the Process of Creating a Map



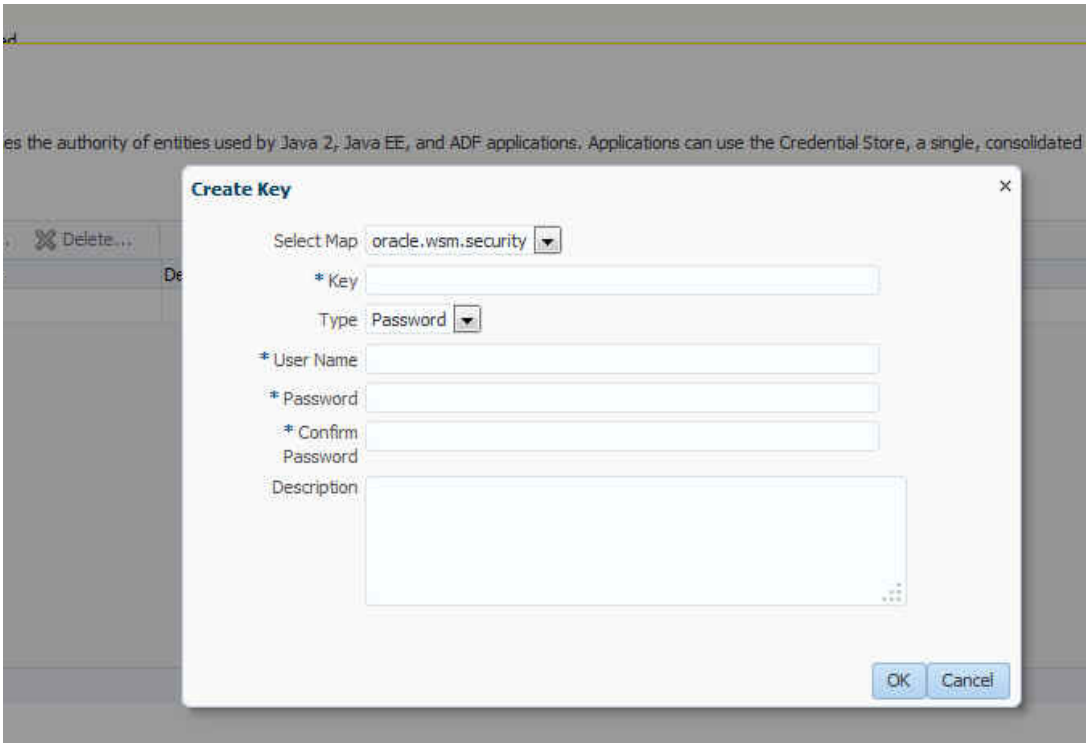
3. Enter `oracle.wsm.security` in the Map Dialog, and save it as shown in Figure 2-15.

Figure 2-15 Creating the Map, Showing the Create Map Dialog, Indicating the Map Name You Provided



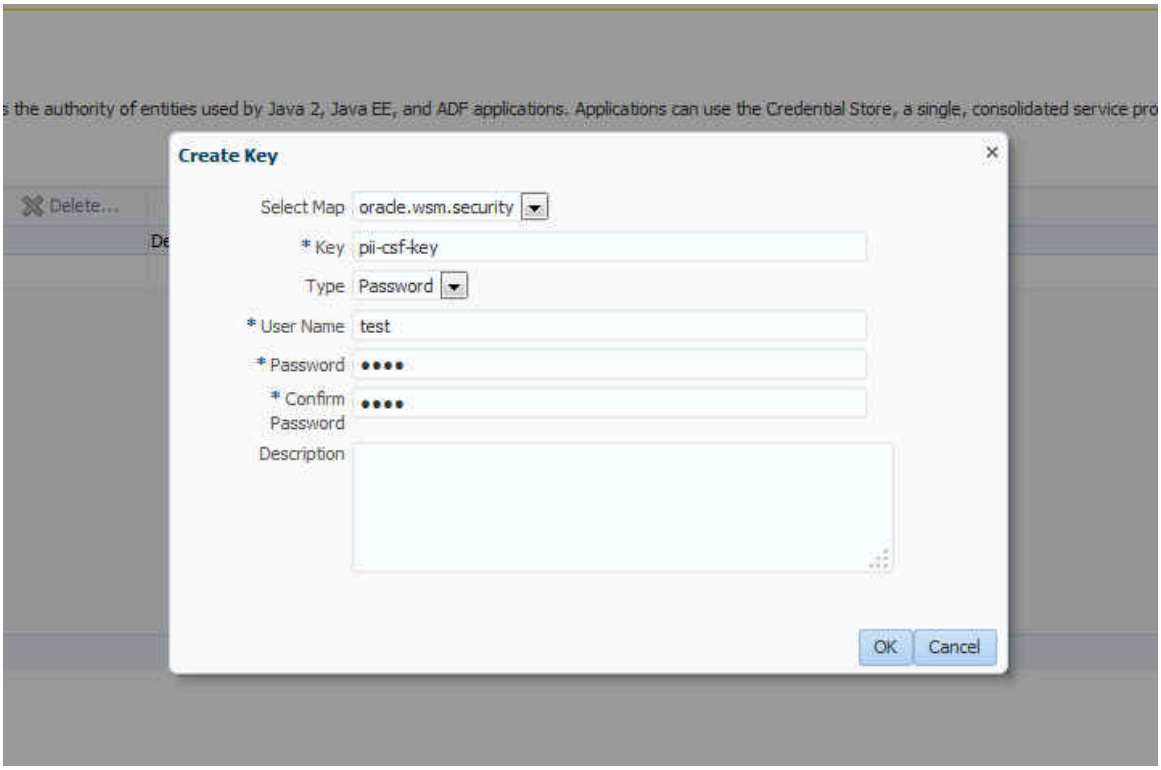
4. The create Key function is enabled. Provide the key name you want to associate with the map, as shown in Figure 2-16.

Figure 2-16 Creating the Security Key, with the Security Map Specified



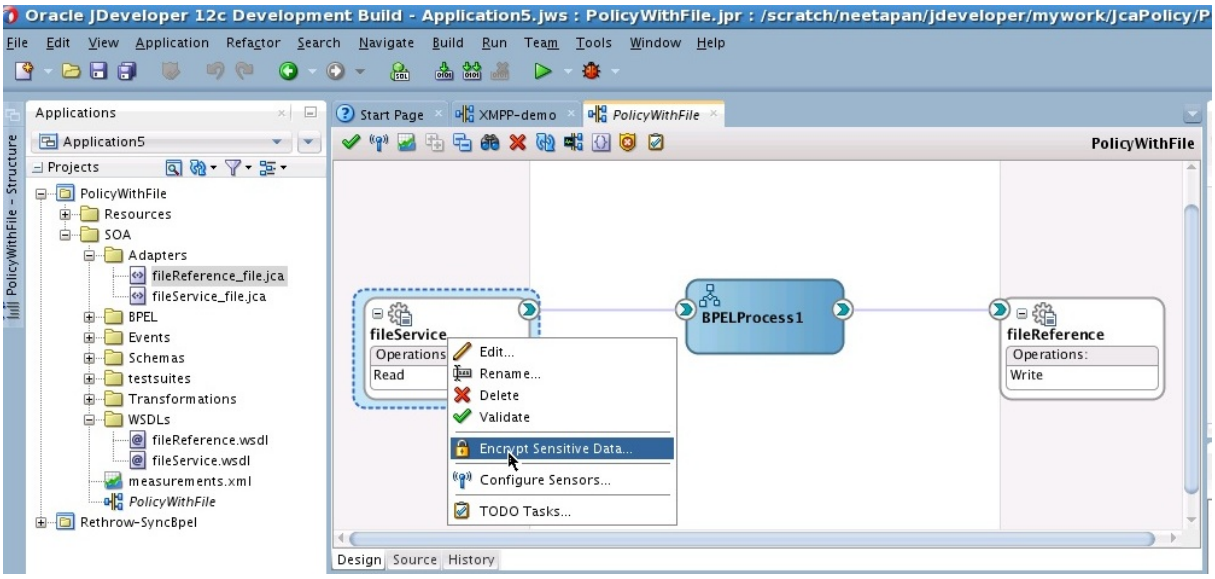
Once you have filled in the entries, the dialog should appear similar to Figure 2-17. Note that you must specify the Map when you create the key.

Figure 2-17 Creating the Security Key, with the Security Map Specified, and the Values Filled-In



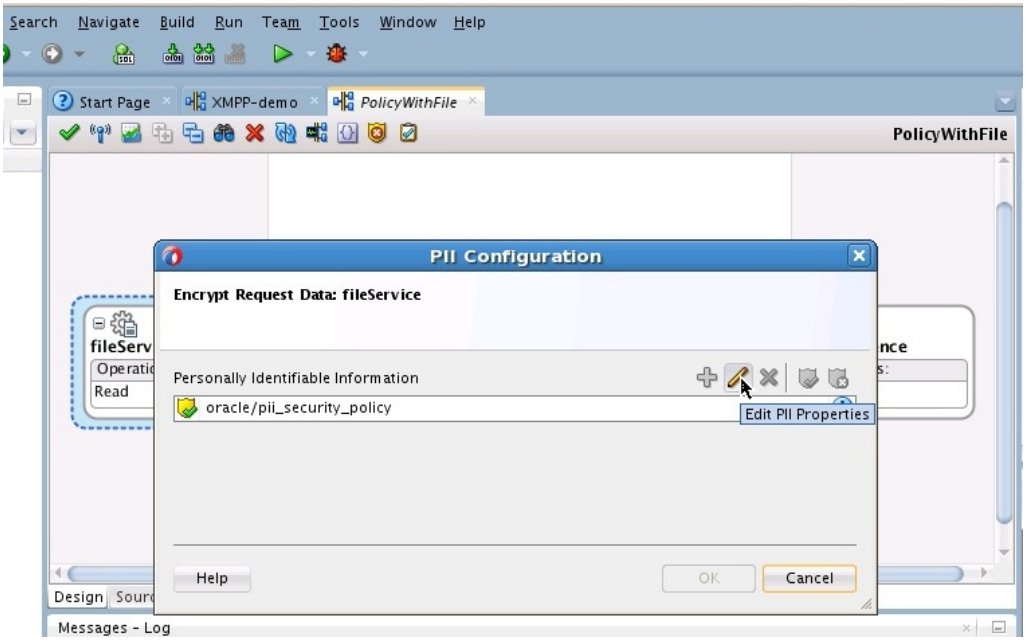
5. In JDeveloper, attach the encrypt key at the file Service Endpoint and select the Expression Builder to complete your composite design. Figure 2-18 shows the use of the **Encrypt Sensitive Data** option for the Service Endpoint.

Figure 2-18 Attaching the Encrypt Key at the Service Endpoint



The **Encrypt Request Data** dialog is shown in Figure 2-19. Select the **Edit** icon to edit the PII policies, and to specify the files to encrypt.

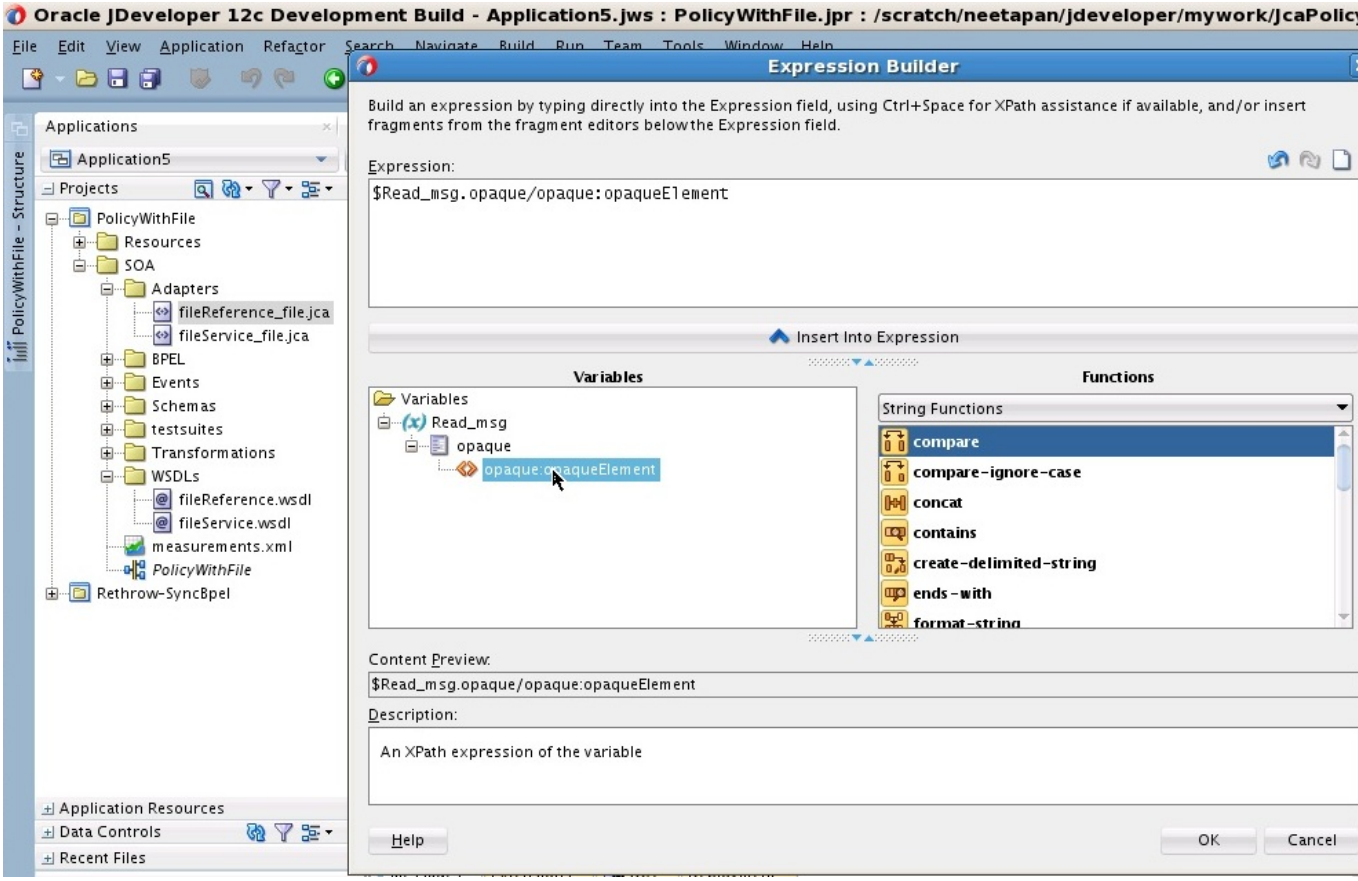
Figure 2-19 Encrypt Request Data Dialog



6. The **Select fields to encrypt** dialog appears. Click the **+** (Add) button to display the Expression Builder.

7. The **Expression Builder** provides you the variable list on which you can attach the encryption JCA policy.

Figure 2-20 Using the Expression Builder to Obtain the Variable List to which You Can Attach the Policy



8. Finally, you must attach decryption at the Reference Endpoint in a similar way as you have attached encryption at the Service Endpoint. Once you have supplied encryption and decryption on the respective endpoints, you have completed applying a PII policy to the data you have selected in the payload.

Testing Applications

You can run and test instances of deployed SOA composite applications from Oracle Enterprise Manager Grid Control Console. Running and testing your instances this way enables you to:

- Manage a composite application
- Initiate an instance of a composite
- Track an instance of a composite
- View detailed component instance audit trails

Fore more information about testing applications, see in Automating the Testing of SOA Composite Applications*Administering Oracle SOA Suite and Oracle Business Process Management Suite* guide.

Setting the Trace Level of Oracle JCA Adapters

Set the trace level for the following types of adapters as follows:

- Oracle JCA Adapters and Oracle Adapter for Oracle Applications: set the log level to, for example, `TRACE:32` in the logger `oracle.soa.adapter`.
- Packaged-application adapters: For outbound interactions, set the `Loglevel` property for packaged-application adapters in the `weblogic-ra.xml` file.
- Legacy adapters: you can use Oracle Studio to set the trace level for Oracle Connect, and the mainframe server.

How to Set the Trace Level of Oracle JCA Adapters

To set the trace level by using the Fusion Middleware Control Console:

1. Navigate to `http://servername:portnumber/em`.

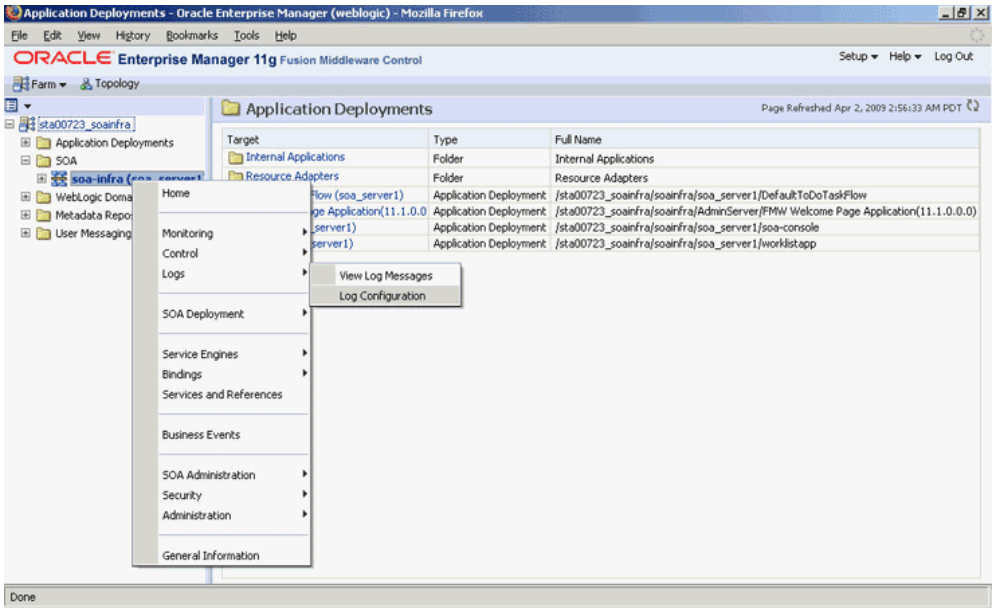
   The Fusion Middleware Control Console home page is displayed.

2. Right-click soa-infra from the SOA Folder in the Navigator in the left pane.

   The console displays a menu.

3. Select Logs, and the Log Configuration, as shown in Figure 2-21.

Figure 2-21 Navigating to the Log Configuration Page



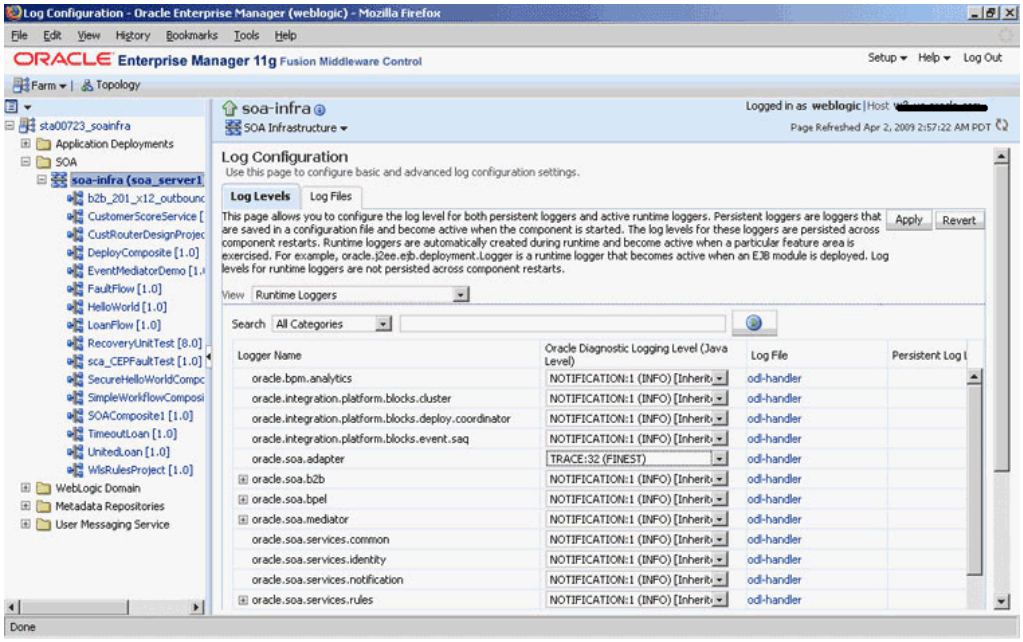
   The Log Configuration page is displayed.

4. Locate **oracle.soa.adapter** in the Logger Name list, and change the log level in the Oracle Diagnostic Logging Level (Java Level) field. In this example, select **Trace:32 (FINEST)** from the list, as shown in Figure 2-22.

Note:

To ensure that log levels persist across component restarts, select **Loggers With Persistent Log Level State** from the View list. By default, the log level is set for runtime loggers. Runtime loggers do not persist across component restarts.

Figure 2-22 The Log Configuration Page



For more information, see Viewing Adapter Logs.

Viewing Adapter Logs

You can view the logs for Oracle JCA Adapters as follows:

- Oracle JCA Adapters and Oracle Adapter for Oracle Applications: These adapters implement the `LogManager` interface of the JCA Binding Component, which redirects log files in the Oracle Diagnostic Logging (ODL) format. For both outbound and inbound interactions, the log files are redirected to the `soa-diagnostic.log` file.

  The log files for the Oracle SOA Suite that is deployed to the `server-soa` managed server are located in:

  `MW_HOME/user_projects/domains/domain_name/servers/server-soa/logs/soa-diagnostic.log`

- Packaged-application adapters: These adapters do not implement the `LogManager` interface because it is not part of the J2CA 1.5 standard. Therefore, for system components the log outputs are redirected to

  `ORACLE_INSTANCE``\diagnostics\logs\component_type\component_name`. For outbound interactions, the logs are directed to the same location. On the other hand, for inbound interactions, logs are redirected to `soa-diagnostic.log`.

- Legacy adapters: In addition to the J2CA resource adapter, legacy adapters consists of Oracle Connect, which consists of native adapters for communicating with the mainframe application and data stores. Oracle Connect logs can be viewed using Oracle Studio, which is the mainframe adapter design-time tool and Oracle Connect management tool. Oracle Connect generates various types of logs, such as the daemon log, workspace log, and server process log.

For more information, see Setting the Trace Level of Oracle JCA Adapters .

Adapter Diagnosability Dumps

For information on Diagnosability Dumps, see Supported SOA Adapter Diagnosability Dumps in the SOA Administration Guide.

Creating a Custom Adapter

You can configure a Custom JCA Adapter wizard as a generic adapter wizard within the JDeveloper IDE that reads and displays its interaction/activation specs, properties and default values from a configuration file. The wizard enables you to select the specs, override the default property values, and add new properties. The Custom Adapter wizard has several purposes:

- You can use the Custom Adapter Wizard on an “as-is" basis to support custom runtime adapters. Supply (or extend) the Custom Adapter configuration file, `customAdapter-config.xml` to use the Custom Adapter.
- You can modify or extend the Custom Adapter classes if you want to create a more specific adapter (for example, you can change the text to match your adapter
- You can use the Custom Adapter wizard to see a simple example of how to develop a new adapter wizard by using the JCA Adapter framework and by hooking into the SCAEndpoint interface.

  After the SOA JDeveloper extension is installed, the Custom Adapter java source files can be found in `<JAVA_HOME> /jdeveloper/integration/adapters/samples/custom`

Configuring a Custom Adapter

The Custom Adapter does not appear by default in JDeveloper Components. The following is the recommended way to update the `extension.xml` file so the Custom Adapter appears in the JDeveloper Components.

1. Open `<jdev_home>\soa\plugins\jdeveloper\extensions\oracle.sca.ui.adapters.jar` with Archive Manager.
2. Find the `META-INF/extension.xml` file, select it, and select **Extract**. You can save the file to the current file that the jar is in.
3. From the Linux command line, edit (using vi or emacs) the extracted `extension.xml` file.
4. Uncomment the Custom Adapter entry and save the file.
5. In the Archive Manager, press the **Add** button (add files to archive).Select the `extension.xml` file, and hit **Add**.
6. Restart JDeveloper.

The `<JDEV_HOME>\jdeveloper\integration\seed\soa\configuration\custom\Adapter-config.xml` file contains the detailed options for the Custom Adapter (connection-factory location, interaction-spec className, activation-spec className, and properties).

The properties within an activation-spec are properties that are specific to an inbound adapter. The properties within an interaction-spec are the properties specific to an

outbound adapter. The property values are the default values shown by the Custom Adapter.

You can modify the contents of the `customAdapter-config.xml` to match options required by your custom runtime adapter. For example, you can change all property names and their default values, add new properties, or add multiple activation or interaction specs.

The `displayResourceKey` and `resourceBundle` attributes are optional. If an activation-spec, interaction-spec, or property element has a `displayResourceKey`, the attribute value is used as a key to retrieve displayable text from a resource bundle. If a resource bundle is not available or the key is not found in the bundle, the key itself is used as the displayable text (it is not required to have a resource bundle). The resource bundle you want to use can be specified by putting the `resourceBundle` attribute on the connection-factory element.

Here is an example of a `customAdapter-config.xml` that has been modified.

Example - Modified customAdapter-config.xml

```
<adapter-config xmlns="http://platform.integration.oracle/blocks/
                      adapter/fw/metadata">

  <connection-factory location="eis/Custom/CustomAdapter"
resourceBundle="oracle.tip.tools.ide.pm.modules.
               bizintegration.adapter.custom.resource. CustomStringResourceBundle"/>

  <endpoint-interaction >
    <interaction-spec className="oracle.tip.adapter.
             custom.outbound.
CustomInteractionSpec" displayResourceKey=
      "CustomInteractionSpec" >
      <property name="PropX" value="x"
       displayResourceKey="SAMP_PROP_X" />
      <property name="PropY" value="y"
       displayResourceKey="Sample Property Y"/>
      <property name="Append" value="false"/>
      <property name="NumberMessages" value="1"/>
    </interaction-spec>
  </endpoint-interaction>

   <endpoint-activation>
    <activation-spec className=
     "oracle.tip.adapter.custom.inbound.CustomActivationSpec"
       displayResourceKey="CustomActivationSpec">
      <property name="UseHeaders" value="false"/>
      <property name="PhysicalDirectory" value="x"/>
      <property name="Recursive" value="true"/>
      <property name="DeleteFile" value="true"/>
      <property name="IncludeFiles" value="x"/>
      <property name="PollingFrequency" value="60"/>
      <property name="MinimumAge" value="0"/>
    </activation-spec>
  </endpoint-activation>

</adapter-config>
```

Custom Adapter Screen Flow

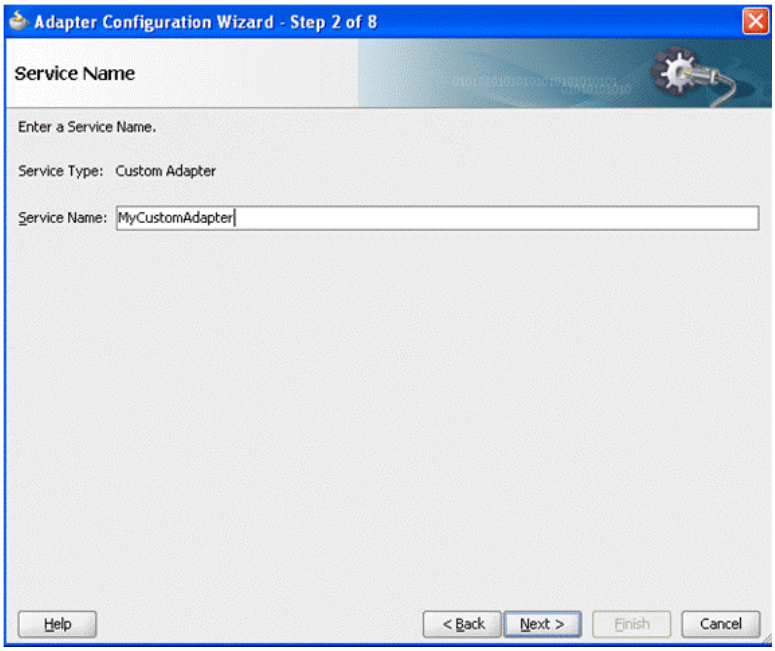
When you drag-and-drop the Custom JCA Adapter icon to the Exposed Service or External Reference swimlane within JDeveloper, the IDE displays the Adapter

Configuration Welcome Page. You can then select **Next** to begin the Custom Adapter Configuration Wizard workflow.

Figure 2-23 Adapter Configuration Wizard Welcome Screen

The next screen displays the service type and name, similar to the way it occurs with the Configuration Wizards of other adapters. This screen enables you to provide the name of a Service that makes sense in the Adapter you are configuring.

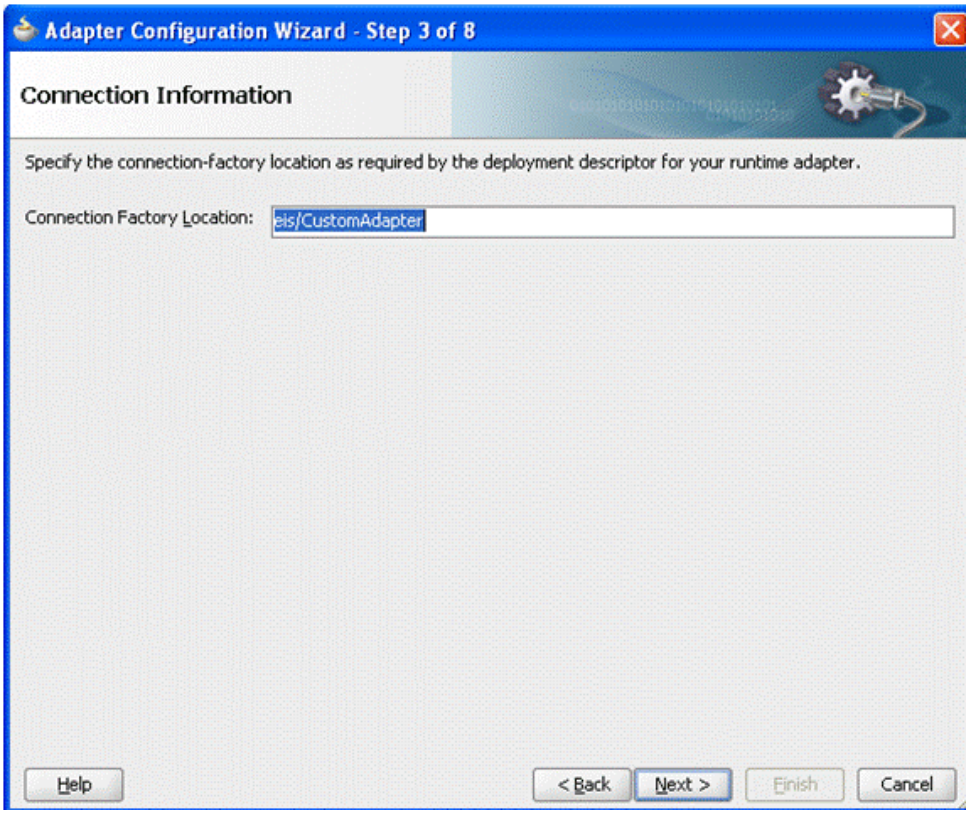
Figure 2-24 Adapter Configuration Wizard Service Name Screen



If the `config.xml` file contains a `<connection-factory>` entry (as required by the custom runtime adapter), the Wizard displays the Connection Information page displaying the default Connection Factory Location. If the `config.xml` does not

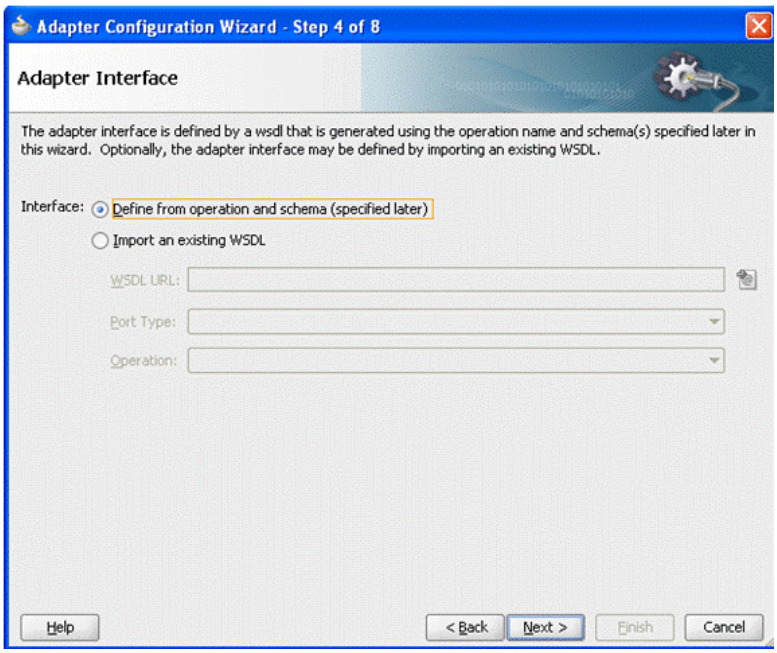
contain a `<connection-factory>` entry (not required by the custom runtime adapter), the Wizard does not display this page.

Figure 2-25 Adapter Configuration Wizard Connection Information Screen



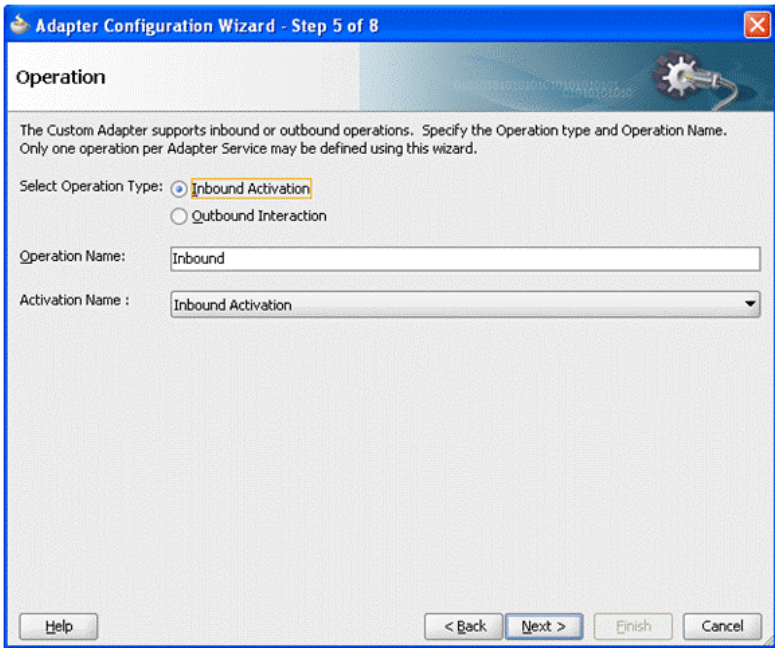
The next screen is the Adapter Interface Screen, which displays information in a similar manner to the configuration wizard for other Adapters. (Refer to the appropriate chapter on each Adapter for information on its configuration wizard.)This screen provides you a way to either define a new WSDL from an operation and schema you provide later, or import an existing WSDL, using the WSDL name, port type and operation.

Figure 2-26 Custom Adapter Wizard Adapter Interface Screen



The next screen enables the user to choose the type of interaction: Inbound or Outbound. If Outbound Interaction is selected, the Wizard provides a list of Interaction Class names (or translated display names as seen in this example) from which to choose. You earlier provided these names in the `customAdapter-config.xml` file.

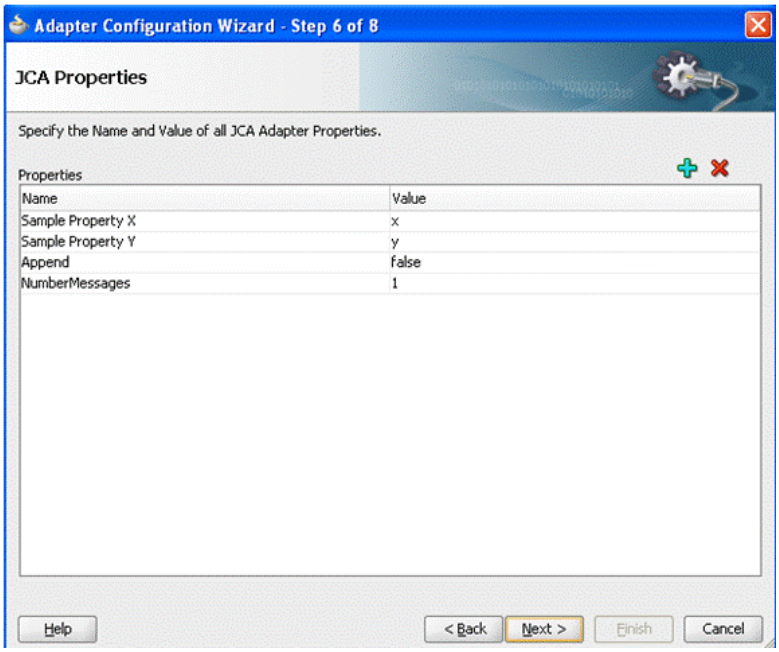
Figure 2-27 Custom Adapter Configuration Wizard Operation Screen (Inbound Choice)



The following screen enables you to specify the name and value of JCA properties. Depending on the Class Name chosen, the screen displays the properties associated

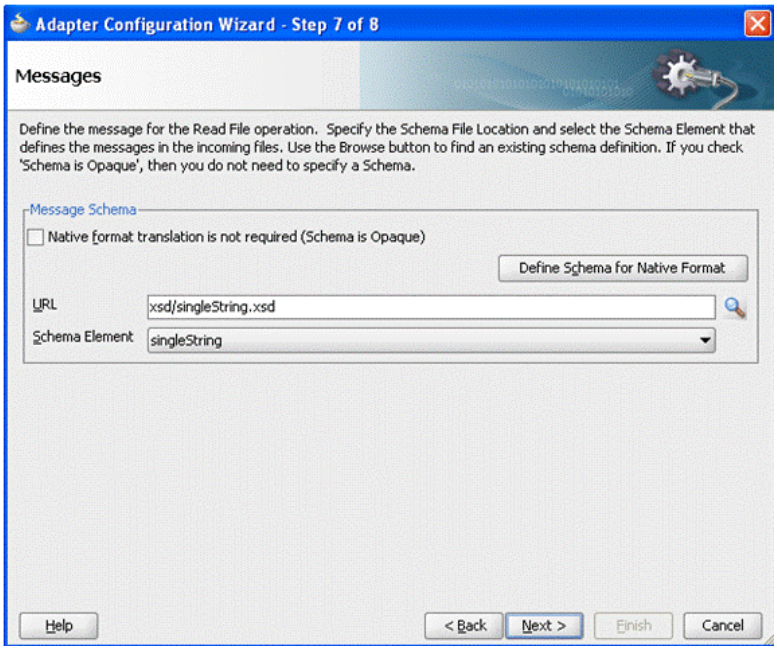
with that class in the `customAdapter-config.xml` file. You can use this screen to change any of the default values and to add or delete properties.

Figure 2-28 Custom Adapter Configuration Wizard JCA Properties Screen



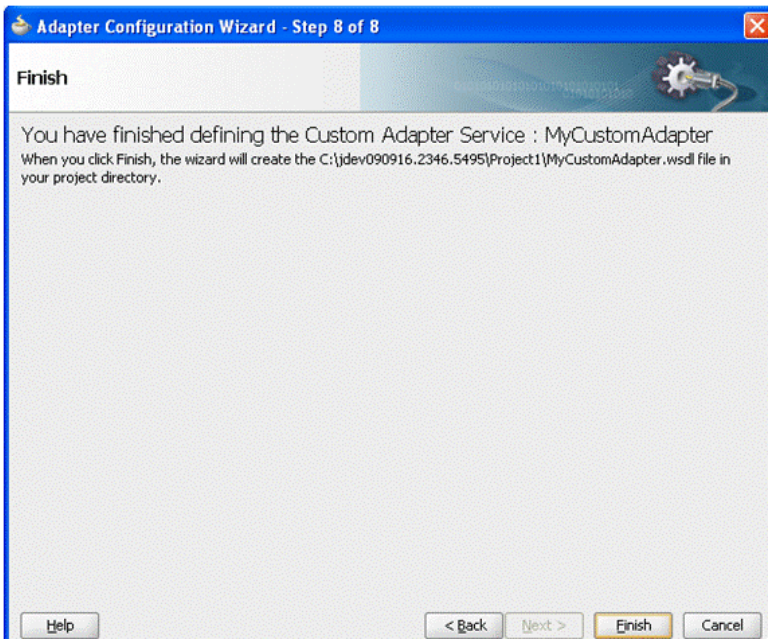
The next screen is the Custom Adapter Wizard Messages Screen, which behaves in a way similar to that of other Adapter Configuration Wizards, enabling you to define the message for the Read File operation, by either specifying a Schema or by declaring that the schema is opaque.

Figure 2-29 Customer Adapter Configuration Wizard Messages Screen



The next page is the Final screen for the Custom Adapter Configuration Wizard. The name of the WSDL files you created is displayed on the screen.

Figure 2-30 Custom Adapter Configuration Wizard Final Screen



Frequently Asked Questions about Adapters

Following are some frequently asked questions about adapters.

Why are My Applications Timing Out?

Why would composite applications are time out? Enough time has been provided for your composite applications to execute with adapters, but applications are still timing out.

A contributing factor is the WebLogic timeout value. The timeout value of the WebLogic Server JTA must be taken into account when you use adapters with your business processing.

For example, you have set the `Timeout Seconds` value at 30 seconds. You should increase the value of the Oracle WebLogic JTA attribute `Timeout Seconds` from its default of 30 to something greater, something that makes sense in the overall context of your business processing. You can use the WebLogic Server Console to change the JTA transaction timeout value by navigating in this fashion: `WLS Console -> SOADomain -> Configuration -> JTA`

How do Transactional and Non-Transactional Adapters Differ?

Transactional Adapters, such as the Oracle JMS Adapter execute within a JTA transaction. A transaction ensures that one or more operations execute as an atomic unit of work.

If an operation within a transaction fails, all operations are rolled-back so that the application is returned to its prior state. Depending on whether the business process logic is defined as stateful or stateless, there may be one or more transactions in a given business process.

Non-transactional adapters implement their own schemes to ensure delivery, without the use of transactional semantics.

The Service Engine obtains a file from an inbound directory, processes the file, and sends the processed file to an output directory. The inbound adapter is limited to translation (if there is one configured) and publishing the translated content which is processed as a part of the business scenario. The business scenario can use the adapter to write to an output directory. However, during this process, if a failover occurs in as a response to a disaster, the file may be lost because of the nontransactional nature of the Oracle File Adapter. As a result, some files read by the inbound adapter might not be sent to the output directory. Of course, when you have a a single node cluster (or no cluster), failover is not an option.

The file adapter is not configured for high availability to avoid message loss, but rather to ensure consistent access to the file system and load balancing across cluster nodes. If you have a single node setup, you do not need a high availability setup for the File adapter, and it does not loose messages.

Consequently, because it is non-transactional, you must configure the Oracle File Adapter for high availability, to ensure that files are not duplicated during a failover. The file adapter never loses messages, but might duplicate some (during disaster recovery).

Additionally, if you have processing scenarios that include Transactional and Non-Transactional Adapters, you must ensure file integrity within the part of your processing that is Non-Transactional.

The JMS adapter can also function with just local transactions; that is, a transaction that begins and commits independently from and within the boundary of a (global) JTA transaction, that is. the local transaction only spans the actual invocation of the adapter.

What Happened to My Application's Rejected Messages? Can One Do Anything With Them?

Rejected messages are stored in the database (specifically, in the `rejected_message` table) by default. A default rejected message handler, which stores them on the file system, participates if you have not defined any policy to handle the rejected messages explicitly. This handler stores the payload and properties of the message on the file system at a predefined location in `WLS_HOME`. Currently, the Oracle SOA suite does not provide the capability to resubmit rejected messages; consequently it is your responsibility to take care of the resubmission.

Advanced Topic: Using the Execution Context ID Across Technologies

SOA Suite 11g deployments can include the use of the technology adapters for various activities including integration with FTP, database, and files, and other activities. Although the integrations with these adapters are easy and feature rich, there can be some challenges from the operations perspective.

One of these challenges is how to correlate a logical business transaction across SOA component instances. This correlation is typically accomplished using the execution context ID (), but you can lose the ECID correlation when the business transaction spans technologies such as FTP, database, and files.

A new feature in the Oracle adapter JCA framework enables the propagation of the. This feature is available in the SOA Suite 11.1.1.7 with back ports available for 11.1.1.4 and 11.1.1.5.

The basic concept of propagating the ECID is to identify a location in the payload of the message where the ECID can be stored.

Next, two Binding Properties, relating to the location of the ECID in the message, are added to either the Exposed Service (left-hand side of composite) or External Reference (right-hand side of composite).

This provides the JCA framework enough information to either extract the ECID from, or add the ECID to, the message.

When you extract the ECID from the message, the ECID is used for the new component instance.

Placing the ECID

When determining where to store the ECID in the message, you have two options:

- Add a new optional element to your message schema.
- Leverage an existing element that is not used in your schema

The best scenario is that you are able to add the optional element to your message, as trying to find an unused element proves difficult in most situations. The schema is holding the ECID value which looks something like the following:

```
11d1def534ea1be0:7ae4cac3:13b4455735c:-8000-00000000000002dc
```

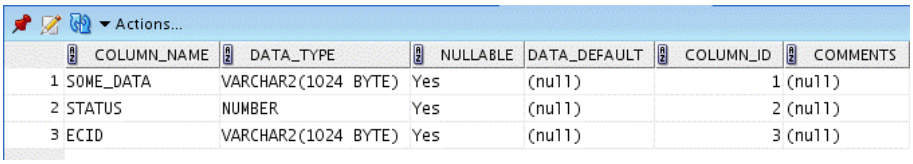
Configuring Composite Services/References

Once you have identified where you want the ECID to be stored in the message, the JCA framework must have this information as well. The two pieces of information that the framework must relate to the message schema:

- The namespace for the element in the message.
- The XPath to the element in the message.

To better understand this, first look at an example for the following database table:

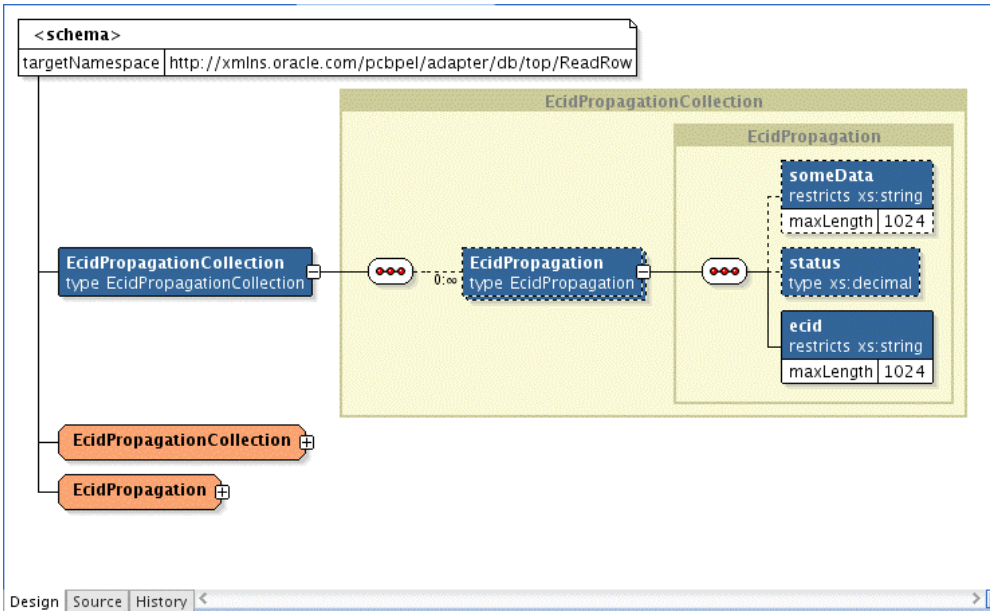
Figure 2-31 Example Database Table



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	SOME_DATA	VARCHAR2(1024 BYTE)	Yes	(null)	1	(null)
2	STATUS	NUMBER	Yes	(null)	2	(null)
3	ECID	VARCHAR2(1024 BYTE)	Yes	(null)	3	(null)

When the Database Adapter Configuration Wizard creates an Exposed Service in the composite, the following schema is created:

Figure 2-32 Schema Created when Exposed Service is Created



For this example, the two Binding Properties added to the ReadRow service in the composite are:

```
<!-- Properties for the binding to propagate the  from the database table -->
<property name="jca..nslist" type="xs:string" many="false">
  xmlns:ns1="http://xmlns.oracle.com/pcbpel/adapter/db/top/ReadRow"
</property>

<property name="jca..xpath" type="xs:string" many="false">
  /ns1:PropagationCollection/ns1:Propagation/ns1:
</property>
```

The property called `jca..nslist` contains the targetNamespace defined in the schema.

The property called `jca..xpath` contains the XPath statement to the element.

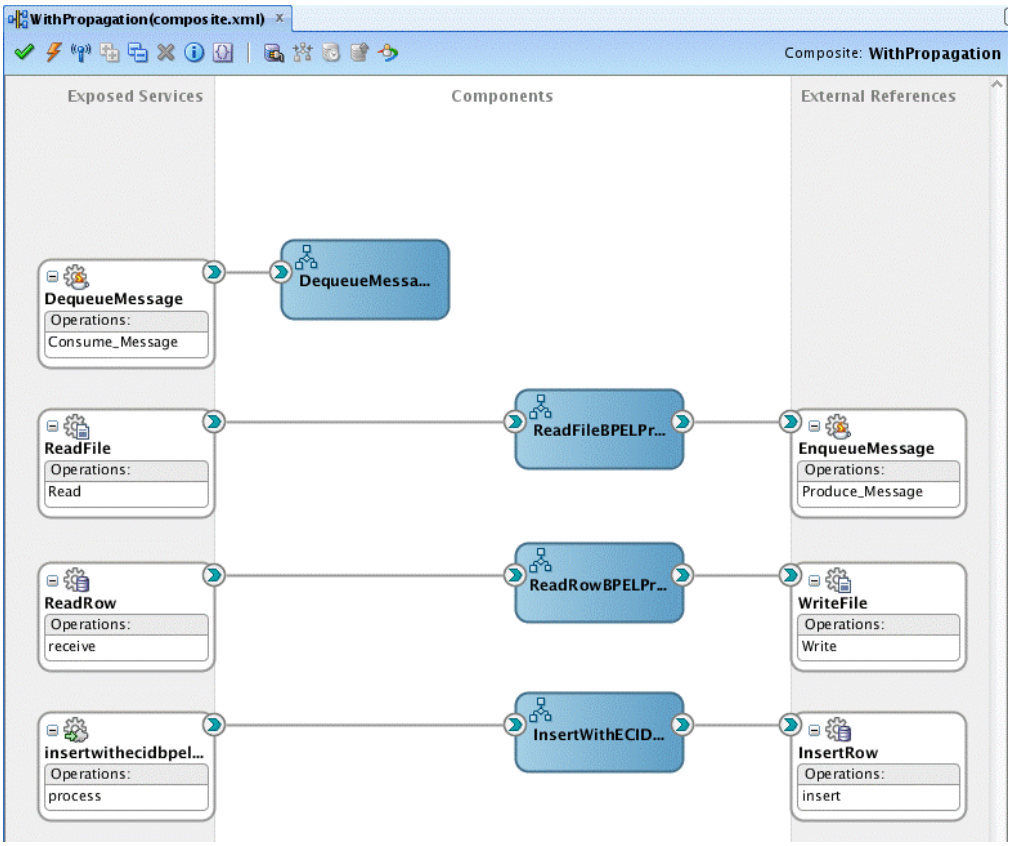
The XPath statement also contains the appropriate namespace prefix (ns1) which is defined in the `jca..nslist` property.

When the Database Adapter service reads a row from the database, it retrieves the ECID value from the payload and remove the ECID element from the payload. When the component instance is created, it is associated with the retrieved ECID and the payload contains everything except the ECID element/value. The only time the ECID is visible is when it is stored safely in the resource technology such as the database, a file, or a queue.

Simple Database/File/JMS Example

This section contains a simplified example of how the ECID can propagate through a database table, a file, and JMS queue. The composite for the example looks like the following:

Figure 2-33 The Composite for the Example



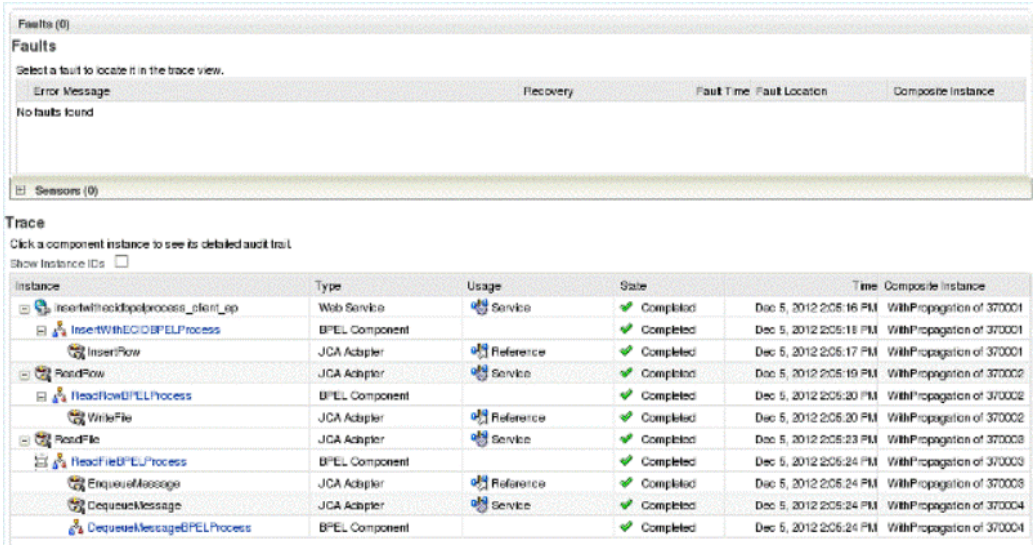
The flow of this example occurs in the following order:

- Invoke database insert using the `insertwithbpelprocess_client_ep` Service.
- The `InsertWithBPELProcess` adds a row to the database using the Database Adapter. The JCA Framework adds the ECID to the message prior to inserting.
- The `ReadRow` Service retrieves the record and the JCA Framework extracts the from the message. The ECID element is removed from the message.
- An instance of `ReadRowBPLProcess` is created and it is associated with the retried ECID.
- The `ReadRowBPELProcess` now writes the record to the file system using the File Adapter. The JCA Framework adds the ECID to the message prior to writing the message to file.
- The `ReadFile` Service retrieves the record from the file system and the JCA Framework extracts the ECID from the message. The ECID element is removed from the message.
- An instance of `ReadFileBPELProcess` is created and it is associated with the retried ECID.
- The ReadFileBPELProcess now enqueues the message using the JMS Adapter. The JCA Framework adds the ECID to the message prior to enqueuing the message ECID

- The `DequeueMessage` Service retrieves the record and the JCA Framework extracts the ECID from the message. The ECID element is removed from the message.
- An instance of `DequeueMessageBPELProcess` is created and it is associated with the retried ECID.
- The logical flow ends.

When viewing the Flow Trace in Fusion Middleware Control, you see all the instances correlated by the ECID:

Figure 2-34 All Instances Correlated by the ECID, as Seen in the Flow Trace



3

Adapter Integration with Oracle Application Server Components

This chapter discusses how Adapters integrate with the Oracle SOA Suite and the Oracle Weblogic Server.

This chapter includes the following topics:

- Adapter Integration with Oracle WebLogic Server
- Adapter Integration with Oracle Fusion Middleware
- Monitoring

Adapter Integration with Oracle WebLogic Server

Oracle JCA Adapters are based on the J2CA 1.5 specification and are deployed to the Oracle WebLogic Server. The resource adapter runs in the same Java Virtual Machine (JVM) as Fusion Middleware does. This section provides an overview of the Oracle WebLogic Server and design-time and runtime integration with an adapter.

This section includes the following topics:

- Oracle WebLogic Server Overview
- Oracle WebLogic Server Integration with Adapters

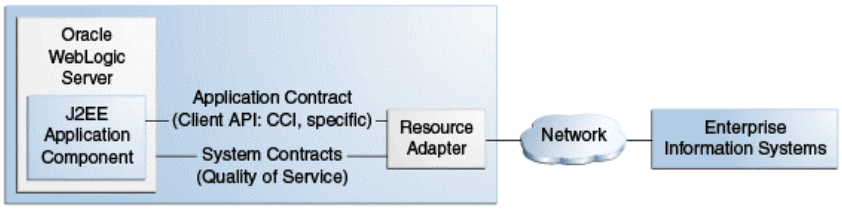
Oracle WebLogic Server Overview

Oracle WebLogic Server is a scalable, enterprise-ready Java Platform, Enterprise Edition (Java EE) application server. The WebLogic Server infrastructure supports the deployment of many types of distributed applications. It is an ideal foundation for building applications based on Service Oriented Architecture (SOA).

All client applications run within the Oracle WebLogic Server environment. To integrate an Oracle WebLogic Server client application with a resource adapter, use the common client interface (CCI). The Oracle WebLogic Server adapter clients include a servlet, EJB, or Java application client that implements the CCI Application Programming Interface (API). The CCI defines a standard client API for application components to access the back-end application.

On the other hand, the contract between the Oracle WebLogic Server container and the resource adapter is defined by the service provider interface (SPI). Contracts define a standard between Oracle WebLogic Server and adapters. The system handles these contracts automatically and hides them from the application developer. Figure 3-1 illustrates the CCI and SPI contracts:

Figure 3-1 Contracts Between Oracle WebLogic Server and Resource Adapter



The Oracle WebLogic Server architecture includes the following set of system-level contracts:

- Connection management: Enables application components to connect to a back-end application and leverage any connection pooling support of the Oracle WebLogic Server container. This leads to a scalable and efficient environment that can support a large number of components requiring access to a back-end application. For more information, see Adding an Adapter Connection Factory.
- Transaction management: Enables an application server to use a transaction manager to manage transactions across multiple resource managers. Most of the adapters support only local transactions (single-phase commit) and not XA transactions (two phase commit). For more information, see How Oracle JCA Adapters Ensure No Message Loss.

  All Oracle JCA Adapters are preconfigured with the correct value for transaction, and you must not change this configuration in the Oracle WebLogic Server Administration Console.
- Security management: The WebLogic Server security architecture provides a comprehensive, flexible security infrastructure designed to address the security challenges of making applications available on the web. WebLogic security can be used standalone to secure WebLogic Server applications or as part of an enterprise-wide security management system that represents a best-in-breed security management solution.

Oracle WebLogic Server Integration with Adapters

Oracle JCA Adapters are based on the J2CA 1.5 specification and are deployed as the J2CA resource adapter within the Oracle WebLogic Server container in this release. The J2CA resource adapter is packaged into a Resource Adapter Archive (RAR) file using the Java Archive (JAR) format. A RAR file contains a correctly formatted deployment descriptor (`/META-INF/ra.xml`). In addition, it contains declarative information about the contract between the Oracle WebLogic Server and resource adapter.

Oracle WebLogic Server generates the corresponding `weblogic-ra.xml` file during the deployment of the J2CA adapter. The `weblogic-ra.xml` file is the deployment descriptor for a resource adapter. It contains deployment configurations for deploying resource adapters to Oracle WebLogic Server, which includes the back-end application connection information as specified in the deployment descriptor of the resource adapter, Java Naming and Directory Interface (JNDI) name to be used, connection pooling parameters, and resource principal mapping mechanism and configurations.

Design Time

Use the adapter design-time tool to generate WSLD, XSD and JCA artifacts for the adapter request-response service. Information in these artifacts would be used a

runtime while creating the JCA interaction.. The Oracle WebLogic Server clients use these XSD files during runtime for calling the J2CA outbound interaction.

Packaged-application adapters use OracleAS Adapter Application Explorer (Application Explorer), Legacy adapters use OracleAS Studio, and technology adapters use Oracle JDeveloper (JDeveloper).

For more information, see Design Time.

Runtime

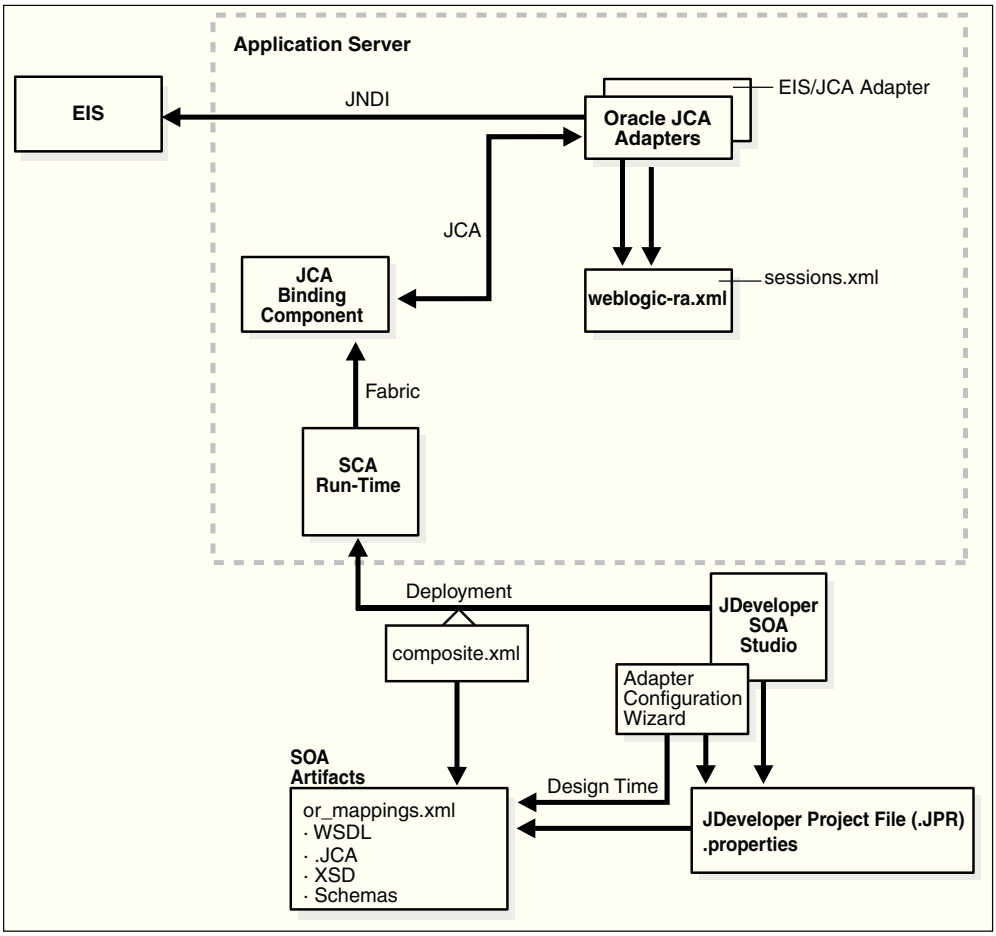
Oracle JCA Adapters are deployed as the J2CA 1.5 resource adapter within the Oracle WebLogic Server container. The J2CA 1.5 specification addresses the life-cycle management, message-inflow (for Adapter Event publish), and work management contracts.

Adapter Integration with Oracle Fusion Middleware

Adapters integrate with the JCA Binding Component of the Oracle Fusion Middleware platform, thereby seamlessly integrating with service engines, such as Oracle BPEL Process Manager (Oracle BPEL PM) and Oracle Mediator.

Figure 3-2 shows the architecture of Oracle JCA Adapters.

Figure 3-2 Oracle Adapter Architecture in Oracle Fusion Middleware



. The Adapter Configuration Wizard, amongst others, generates a WSDL and a JCA properties file, which contain the binding information for that service.

Oracle technology adapters gather and publish statistics for every inbound and outbound message they process. For more information, see Monitoring .

For information on using adapters with the Oracle Service Bus, see "Using the JCA Transport and JCA Adapters" in the Developing Services with Oracle Service Bus guide. For the Open Service Bus (OSB), the only difference is that rather than having the .jca file referenced by the composite.xml itself, for OSB the file is referenced by a proxy/business service.

This section includes the follows topics:

- Oracle BPEL Process Manager Overview
- Oracle Mediator Overview
- Oracle Fusion Middleware Integration with Adapters
- Oracle SOA Composite Integration with Adapters

Oracle BPEL Process Manager Overview

Oracle BPEL PM is a comprehensive solution for creating, deploying, and managing Oracle BPEL PM business processes. Oracle BPEL PM is based on the Service Oriented Architecture (SOA) to provide flexibility, interoperability, reusability, extensibility, and rapid implementation. Oracle BPEL PM reduces the overall cost of management, modification, extension, and redeployment of existing business processes. Each business activity is a self-contained, self-describing, modular application with an interface that is defined by a WSDL file and the business process that is modeled as a web service.

Oracle Mediator Overview

Oracle Mediator provides a lightweight framework to mediate between various producers and consumers of services and events. In most business environments, customer data resides in disparate sources including business partners, legacy applications, enterprise applications, databases, and custom applications. The challenge of integrating this data can be met by using Oracle Mediator to deliver appropriate real-time data access to all applications that update or have a common interest in the same data. For example, a Mediator can accept data contained in a text file from an application or service, transform it to a format appropriate for updating a database that serves as a customer repository, and then route and deliver the data to that database.

Oracle Fusion Middleware Integration with Adapters

The JCA Binding Component is used for the bidirectional integration of the J2CA 1.5 resource adapters with Oracle BPEL PM and Oracle Mediator. Oracle JCA Adapters generate a WSDL file and a JCA binding, and expose the underlying interactions as web Services.

The interface (input/output XML elements) to an adapter service is described through a WSDL file. However, in the 11*g* release, the binding element has been removed, making the WSDL file abstract. Instead the binding information, that the JCA Binding Component (referred to as adapter framework in the previous releases) and adapters must invoke for a particular call on a particular EIS, is stored in a separate `binding.jca` file.

This section describes:

- Design Time
- Runtime
- Use Case: Integration with Oracle BPEL Process Manager
- Oracle SOA Composite Overview
- Adapters Integration With Oracle SOA Composite

Design Time

While integrating adapters with Oracle BPEL PM and Oracle Mediator, the underlying adapter services are exposed as WSDL files with the J2CA extension. The following table lists the design-time tools used for generating WSDL and JCA files for various types of adapters.

Adapter	Tool
Oracle Technology Adapters	Oracle JDeveloper
Mainframe and TP-Monitor Adapters	Oracle Studio
Packaged-Application Adapters	Application Explorer
Oracle Adapter for Oracle Applications	Oracle JDeveloper

WSDL files are created for both request-response and event-notification services of an adapter. The J2CA extension contains J2CA elements that are required by the JCA Binding Component during runtime to convert web service messages to J2CA Interactions and back. The J2CA WSDL extension elements contain the metadata for the JCA Binding Component to call any request-response service and activate any inbound J2CA 1.5 endpoint to receive inbound events. The J2CA extension elements for the request-response service contains the JNDI location and `InteractionSpec` details for calling an outbound interaction. The J2CA extension elements for the event-notification service contains the resource adapter class name and `ActivationSpec` parameters for publishing an adapter event through the J2CA inbound interaction.

Figure 3-3 illustrates the design-time tool, JDeveloper, used by Oracle JCA Adapters.

Figure 3-3 Design Time Configuration of Technology Adapters

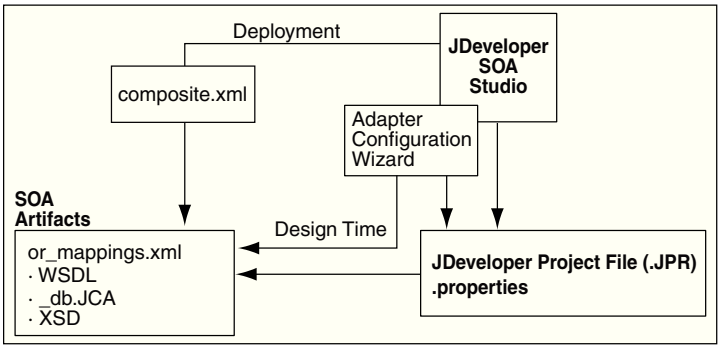
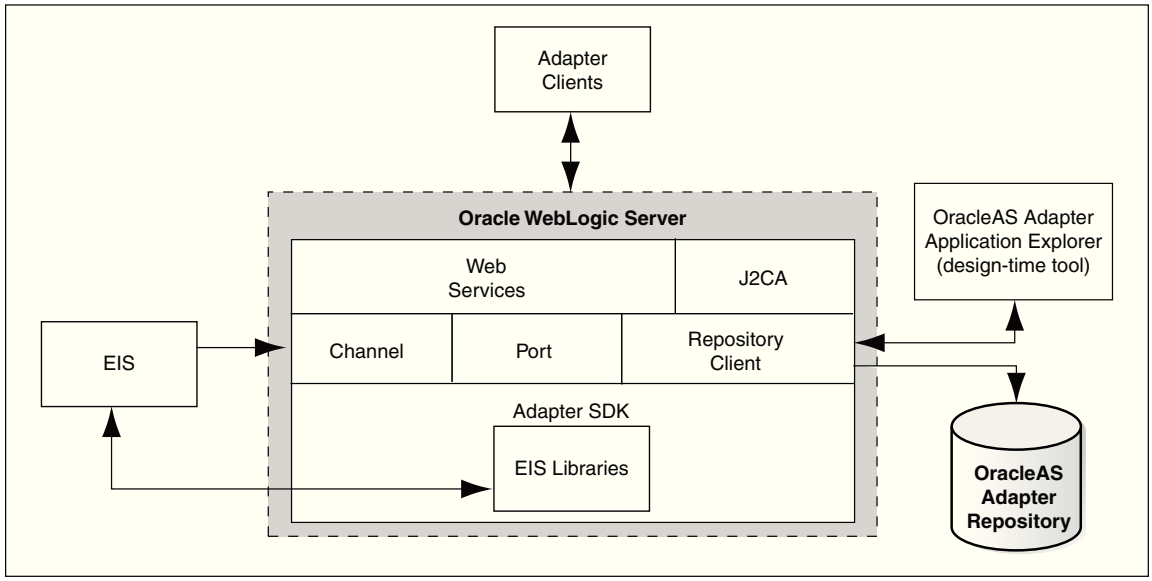


Figure 3-4 illustrates the design-time tool for configuring packaged-application adapters. In this figure, the design-time tools are used to expose adapter metadata as WSDL files. The WSDL files are consumed by BPEL Process Manager during runtime.

Figure 3-4 Configuring Packaged-Application Adapters



Runtime

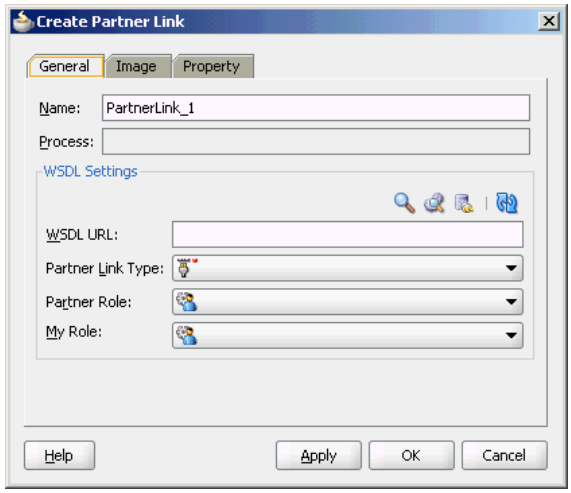
Oracle Application Server adapters are based on the J2CA 1.5 specification, and BPEL is deployed on the runtime on the Oracle WebLogic Server. The JCA Binding Component acts as a glue layer that integrates the standard J2CA 1.5 resource adapter with the Oracle BPEL Process Manager and Oracle Mediator during runtime.

The web service invocation launched by the BPEL Invoke activity is converted to a J2CA CCI outbound interaction, and the J2CA response is converted back to a web service response.

Use Case: Integration with Oracle BPEL Process Manager

From the Partner Link dialog in Oracle BPEL PM, shown in Figure 3-5, you can access the adapters that are provided with Oracle BPEL PM.

Figure 3-5 Partner Link dialog box

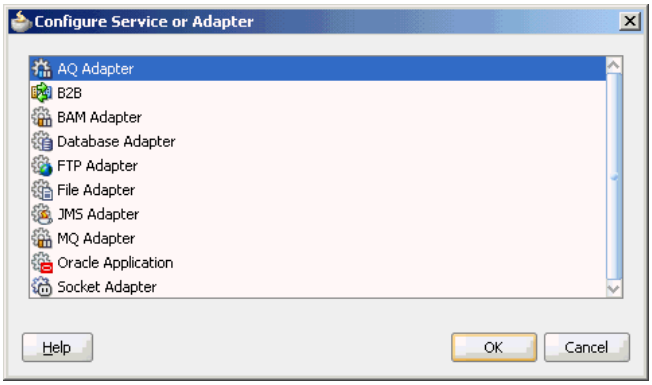


Click the **Define Service** icon, shown in Figure 3-6, to access the Configure Service or Adapter dialog.

Figure 3-6 Defining an Adapter

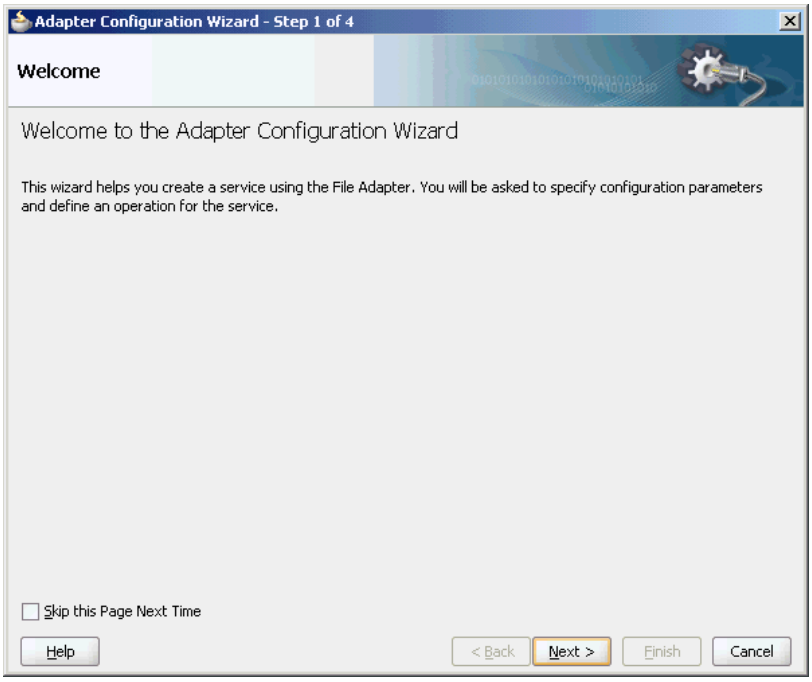
This dialog enables you to configure the types of adapters shown in Figure 3-7 for use with Oracle BPEL processes.

Figure 3-7 Adapter Types



When you select an adapter type (Oracle AQ Adapter in this example), and then click **OK**, the Adapter Configuration Wizard - Welcome page appears, as shown in Figure 3-8.

Figure 3-8 The Adapter Configuration Wizard- Welcome Page

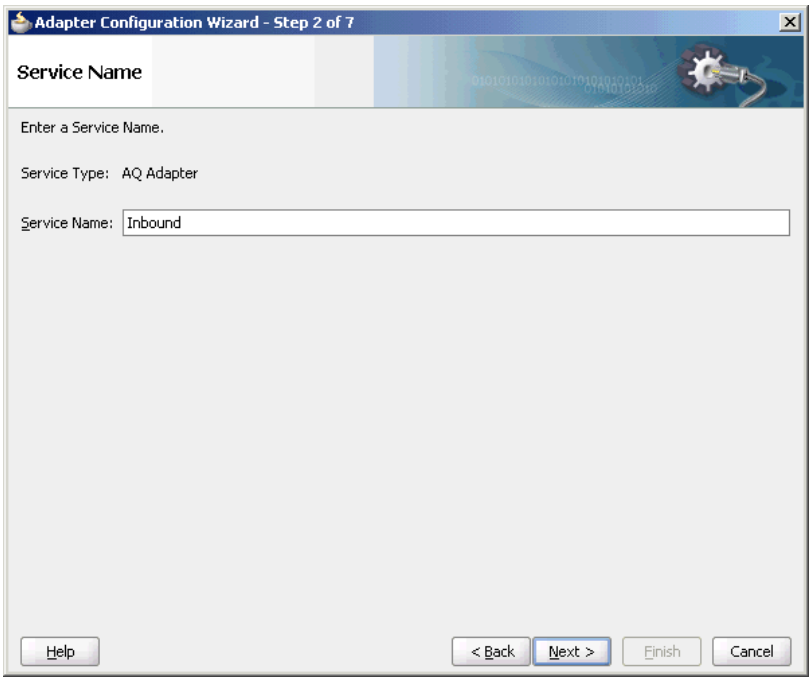


Click **Next**, and the Service Name page appears, as shown in Figure 3-9. You are prompted to enter a name for the service.

For this example, AQ Adapter is selected, as shown in Figure 3-7. When the wizard completes, a WSDL file by this service name appears in the Application Navigator for the BPEL process (for this example, named `DequeueDemo.wsdl`). This file includes

the adapter configuration settings you specify with this wizard. Other configuration files (such as header properties and files specific to the adapter) are also created and displayed in the Application Navigator.

Figure 3-9 The Adapter Configuration Wizard- Service Name Page



The Adapter Configuration Wizard windows that appear after the Service Name window are based on the adapter type you selected. These configuration windows and the information you must provide are described in later chapters of this guide.

Oracle SOA Composite Integration with Adapters

Oracle JCA Adapters can be integrated with Oracle SOA Suite.

This section includes the following:

- Oracle SOA Composite Overview
- Adapters Integration With Oracle SOA Composite

Oracle SOA Composite Overview

An SOA composite application is an assembly of services, service components, references, and wires designed and deployed to meet a business need.

SOA provides an enterprise architecture that supports building connected enterprise applications. SOA facilitates the development of enterprise applications as modular business web services that can be easily integrated and reused, creating a truly flexible, adaptable IT infrastructure.

Adapters Integration With Oracle SOA Composite

A composite is an assembly of services, service components, wires, and references designed and deployed in a single application. The composite processes the information described in the messages.

For example, a composite includes an inbound service binding component (an inbound adapter), a BPEL process service component, and an outbound reference binding component (an outbound adapter). The details of this composite are stored in the `composite.xml` file.

An Oracle SOA composite typically comprises the following parts:

- **Binding Components**

  The binding component establishes the connectivity between a SOA composite and the external world. There are two types of binding components:

  – *Service* Binding Components

    Provide the outside world with an entry point to the SOA composite application. The WSDL file of the service informs external applications of its capabilities. These capabilities are used for contacting the SOA composite application components. The binding connectivity of the service describes the protocols that can communicate with the service, for example, Oracle JCA adapter.

  – *Reference* Binding Components

    Enable messages to be sent from the SOA composite application to external services in the outside world.

  The Oracle SOA Suite provides web Services, such as Oracle JCA adapters for integrating services and references with technologies (for example, databases, file systems, FTP servers, messaging: JMS, IBM MQ, and so on) and applications (Oracle E-Business Suite, PeopleSoft, and so on). This includes Oracle AQ Adapter, Oracle Database Adapter, Oracle File Adapter, Oracle FTP Adapter, Oracle JMS Adapter, Oracle MQ Series Adapter, and Oracle Socket Adapter.

- **Service Infrastructure**

  Provides internal message transport. For example, receives the message from an inbound adapter and posts the message for processing to the BPEL process service engine.

- **Service Engines** (containers hosting service components)

  Host the business logic or processing rules of the service components. Each service component has its own service engine. For example, an Oracle BPEL process engine or an Oracle Mediator Component.

  For more information about adapter integration with service engines, see Adapter Integration with Oracle Fusion Middleware.

- **UDDI and MDS**

  The MDS (Metadata Service) repository stores descriptions of available services. The UDDI advertises these services and enables discovery and dynamic binding at runtime.

- **SOA Archive: Composite**

  The deployment unit that describes the composite application.

A composite is an assembly of services (for example, inbound adapters), service components, wires, and references (for example, outbound adapters) designed and deployed in a single application. The composite processes the information described in the messages. A `composite.xml` file is automatically created when you create a SOA

project. This file describes the entire composite assembly of services, service components, references, and wires. The `composite.xml` file describes the entire SOA composite.

See the example below for a sample `composite.xml` file.

Example - composite.xml File

```
Composite.xml (JCA Bindings)<?xml version="1.0"
encoding="UTF-8" ?>
<!-- Generated by Oracle SOA Modeler version
<! -- 1.0 at [2/23/09 3:02 PM]. -->
<composite name="MediatorFlatStructure"
   revision="1.0"
   label="2009-02-23_15-02-00_374"
   mode="active"
   state="on"
   xmlns="http://xmlns.oracle.com/sca/1.0"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
   xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy"
   xmlns:ui="http://xmlns.oracle.com/soa/designer/">
    <import namespace="http://xmlns.oracle.com/pcbpel/
         adapter/file/SOA-FlatStructure/
         MediatorFlatStructure/MedFlatIn%2F"
        location="MedFlatIn.wsdl" importType="wsdl"/>
    <import namespace="http://xmlns.oracle.com/pcbpel/
          adapter/file/SOA-FlatStructure/
          MediatorFlatStructure/MedFlatOut%2F"
          location="MedFlatOut.wsdl" importType="wsdl"/>
    <service name="MedFlatIn"
           ui:wsdlLocation="MedFlatIn.wsdl">
        <interface.wsdl interface=
                "http://xmlns.oracle.com/pcbpel/
          adapter/file/SOA-FlatStructure/
          MediatorFlatStructure/MedFlatIn%2F#wsdl.
          interface(Read_ptt)"/>
        <binding.jca config="MedFlatIn_file.jca"/>
    </service>
    <component name="MediatorFlat">
        <implementation.mediator
                src="MediatorFlat.mplan"/>
    </component>
    <reference name="MedFlatOut"
                ui:wsdlLocation="MedFlatOut.wsdl">
        <interface.wsdl interface=
          "http://xmlns.oracle.com/pcbpel/
      adapter/file/SOA-FlatStructure/
      MediatorFlatStructure/
           MedFlatOut%2F#wsdl.interface(Write_ptt)"/>
        <binding.jca config="MedFlatOut_file.jca"/>
    </reference>
    <wire>
        <source.uri>MedFlatIn</source.uri>
        <target.uri>
             MediatorFlat/MediatorFlat</target.uri>
    </wire>
    <wire>
        <source.uri>MediatorFlat/MedFlatOut</source.uri>
<target.uri>MedFlatOut</target.uri>
    </wire>
</composite>
```

For more information about Oracle SOA composite and its integration with various service engines, see "Getting Started with Developing SOA Composite Applications" and other sections in the *Developing SOA Applications with Oracle SOA Suite* guide.

Monitoring Oracle JCA Adapters

In Oracle BPEL Process Manager and Oracle Mediator, Oracle JCA adapters such as File, JMS, and Database, gather and publish statistics for every message they process, either inbound or outbound. The statistics are broken down into categories and individual tasks. The following is an example of how statistics are broken down in an outbound (reference) process:

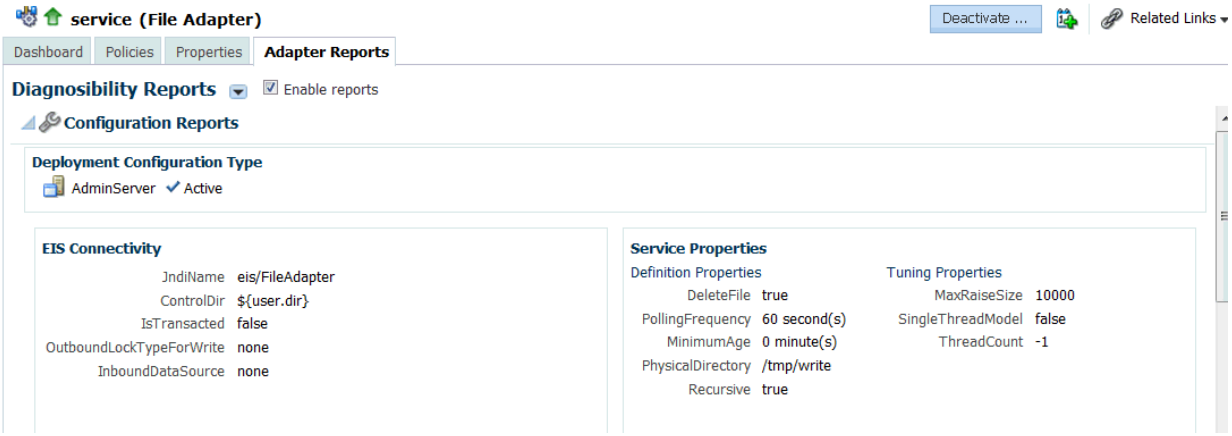
- Adapter Preprocessing
  - Preparing `InteractionSpec`
- Adapter Processing
  - Setting up Callable Statement
  - Invoking Database
  - Parsing Result
- Adapter Postprocessing

The adapter statistics can be viewed in the Fusion Middleware Control Console. The following are the steps to view the adapter statistics:

1. Navigate to `http://servername:portnumber/em`.
2. In the `SOA` folder in the Target Navigation tree (in the extreme left pane), click `soa_infra`.

   The soa-infra page is displayed.
3. From the SOA Infrastructure menu in the soa-infra page, click **Services and References**.

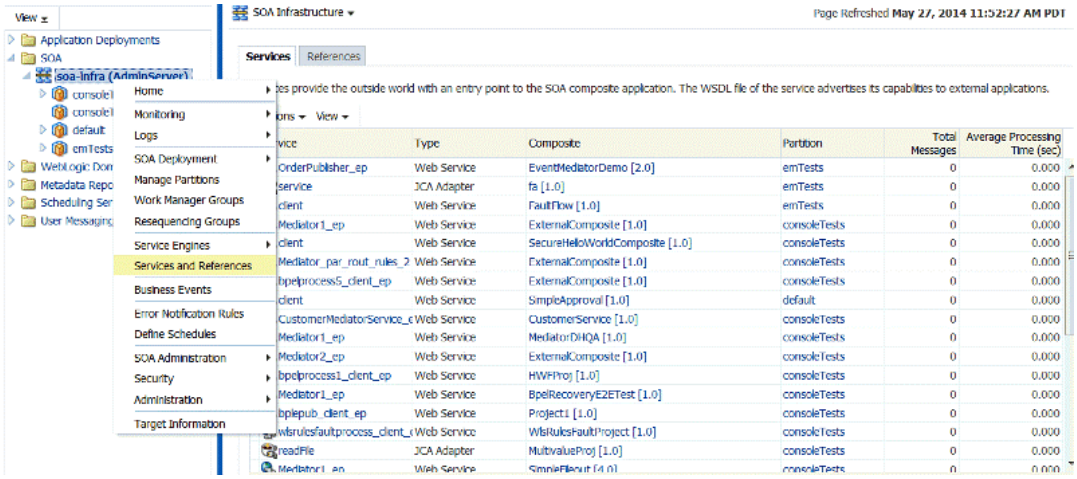
Figure 3-10 Viewing the Adapter Statistics in the Fusion Middleware Control Console using Diagnosability Reports



The SOA Infrastructure Home > Interfaces page is displayed, as shown in Figure 3-11.

This page shows a list of all currently active inbound (services) and outbound adapter interactions (references), and the average execution time for the various steps each adapter performs.

Figure 3-11 The SOA Infrastructure Home Page



For more information on monitoring and configuring SOA Adapters, including Adapter Reporting new to this release, see the Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite, in particular, the Monitoring Oracle JCA Adapters, and Configuring Oracle JCA Adapters chapters, and the Diagnosing Problems with SOA Composite Applications, which has information on using Adapter logs to assist in diagnosing SOA Composite application problems that are related to Adapters.

4

Oracle JCA Adapter for Files/FTP

This chapter describes how to use the Oracle File and FTP Adapters, which work with Oracle BPEL Process Manager and Oracle Mediator and Oracle Mediator. Information on concepts, features, configuration and use cases for the Oracle File and FTP Adapters is also provided.

This chapter includes the following sections:

Note:

The term *Oracle JCA Adapter for Files/FTP* is used for the Oracle File and FTP Adapters, which are separate adapters with very similar functionality.

Introduction to Oracle File and FTP Adapters

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

This section includes the following topics:

Oracle File and FTP Adapters Architecture

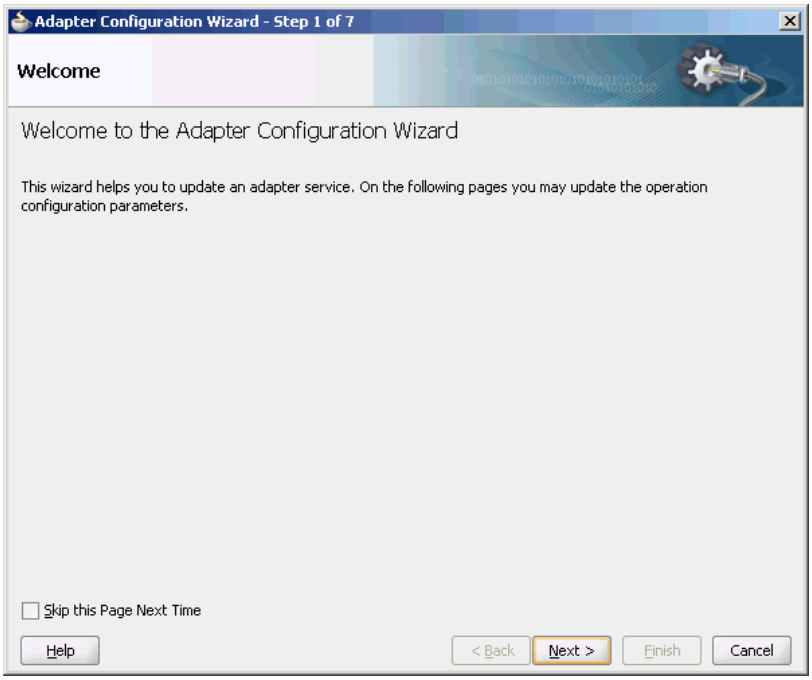
The Oracle File and FTP Adapters are based on JCA 1.5 architecture. JCA provides a standard architecture for integrating heterogeneous enterprise information systems (EIS). The JCA Binding Component of the Oracle File and FTP Adapters expose the underlying JCA interactions as services (`WSDL` with JCA binding) for Oracle BPEL PM

integration. For details about Oracle JCA Adapter architecture, see Introduction to Oracle JCA Adapters.

Oracle File and FTP Adapters Integration with Oracle BPEL PM

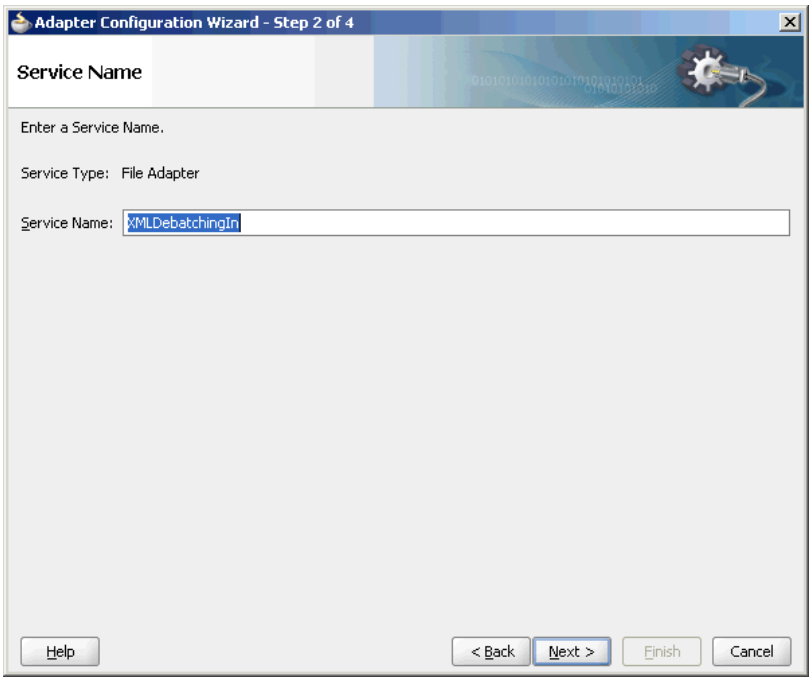
The Oracle File and FTP Adapters are automatically integrated with Oracle BPEL PM. When you drag and drop File Adapter for FTP Adapter from the Components window of JDeveloper BPEL Designer, the Adapter Configuration Wizard starts with a Welcome page, as shown in Figure 4-1.

Figure 4-1 The Adapter Configuration Wizard - Welcome Page



This wizard enables you to select and configure the Oracle File and FTP Adapters. The Adapter Configuration Wizard then prompts you to enter a service name, as shown in Figure 4-2.

Figure 4-2 The Adapter Configuration Wizard - Service Name Page



When configuration is complete, a `WSDL` and `JCA` file pair is created in the Application Navigator section of Oracle JDeveloper. (JDeveloper) This `JCA` file contains the configuration information you specify in the Adapter Configuration Wizard.

The Operation Type page of the Adapter Configuration Wizard prompts you to select an operation to perform. Based on your selection, different Adapter Configuration Wizard pages appear and prompt you for configuration information. Table 4-1 lists the available operations and provides references to sections that describe the configuration information you must provide.

Table 4-1 Supported Operations for Oracle BPEL Process Manager

Operation	Section
Oracle File Adapter	-
• Read File (inbound operation)	Read File Concepts
• Write File (outbound operation)	Write File Concepts
• Synchronous Read File (outbound operation)	Synchronous Read Concepts
• List Files (outbound operation)	File Listing Concepts
Oracle FTP Adapter	-
• Get File (inbound operation)	Get File Concepts
• Put File (outbound operation)	Put File Concepts
• Synchronous Get File (outbound operation)	Synchronous Get File Concepts
• List Files (outbound operation)	File Listing Concepts

For more information about Oracle JCA Adapter integration with Oracle BPEL PM, see Introduction to Oracle JCA Adapters.

Oracle File and FTP Adapters Integration with Mediator

The Oracle File and FTP Adapters are automatically integrated with Mediator. When you create an Oracle File or FTP Adapter service in JDeveloper Designer, the Adapter Configuration Wizard is started.

This wizard enables you to select and configure the Oracle File and FTP Adapters. When configuration is complete, a `WSDL, JCA` file pair is created in the Application Navigator section of JDeveloper. This `JCA` file contains the configuration information you specify in the Adapter Configuration Wizard.

The Operation Type page of the Adapter Configuration Wizard prompts you to select an operation to perform. Based on your selection, different Adapter Configuration Wizard pages appear and prompt you for configuration information. Table 4-2 lists the available operations and provides references to sections that describe the configuration information you must provide. For more information about Adapters and Mediator, see Introduction to Oracle JCA Adapters.

Table 4-2 Supported Operations for Oracle Mediator

Operation	Section
Oracle File Adapter	-
• Read File (inbound operation)	Read File Concepts
• Write File (outbound operation)	Write File Concepts
• Synchronous Read File (outbound operation)	Synchronous Read Concepts
• List Files (outbound operation)	File Listing Concepts
Oracle FTP Adapter	-
• Get File (inbound operation)	Get File Concepts
• Put File (outbound operation)	Put File Concepts
• Synchronous Get File (outbound operation)	Synchronous Get File Concepts
• List Files (outbound operation)	File Listing Concepts

Oracle File and FTP Adapters Integration with SOA Composite

A composite is an assembly of services, service components (Oracle BPEL PM and Mediator), wires, and references designed and deployed in a single application. The composite processes the information described in the messages. The details of the composite are stored in the `composite.xml` file. For more information about integration of the Oracle File and FTP Adapters with SOA composite, see Oracle SOA Composite Integration with Adapters.

Oracle File and FTP Adapters Features

The Oracle File and FTP Adapters enable you to configure a BPEL process or a Mediator to interact with local and remote file system directories. This section explains the following features of the Oracle File and FTP Adapters:

- File Formats
- FTP Servers
- Inbound and Outbound Interactions
- File Debatching
- File ChunkedRead
- File Sorting
- Dynamic Outbound Directory and File Name Specification
- Security
- Nontransactional
- Proxy Support
- No Payload Support
- Large Payload Support
- File-Based Triggers
- Pre-Processing and Post-Processing of Files
- Error Handling
- Threading Model
- Performance Tuning
- High Availability
- Multiple Directories
- Append Mode
- Recursive Processing of Files Within Directories in
- Securing Enterprise Information System Credentials

Note:

For composites with Oracle File and FTP Adapters, which are designed to consume very large number of concurrent messages, you must set the number of open files parameter for your operating system to a larger value. For example, to set the number of open files parameter to `8192` for Linux, use the `ulimit -n 8192` command.

File Formats

The Oracle File and FTP Adapters can read and write the following file formats and use the adapter translator component at runtime:

- XML (both XSD- and DTD-based)
- Delimited
- Fixed positional
- Binary data
- COBOL Copybook data

The Oracle File and FTP Adapters can also treat file contents as an opaque object and pass the contents in their original format (without performing translation). The opaque option handles binary data such as Jpgs and GIFs, whose structure cannot be captured in an XSD or data you do not want to have translated.

Note that opaque representation base-64 encodes the payload and increase the size of the payload in-memory by a third. Opaque/base-64 representation is usually used for passing binary data within XML. See also Large Payload Support, for a description of attachment support.

The translator enables the Oracle File and FTP Adapters to convert native data in various formats to XML, and from XML to other formats. The native data can be simple (just a flat structure) or complex (with parent-child relationships). The translator can handle both XML and non-XML (native) formats of data.

FTP Servers

Oracle FTP Adapter supports most RFC 959 compliant FTP servers on all platforms. It also provides a pluggable mechanism that enables Oracle FTP Adapter to support additional FTP servers. In addition, Oracle FTP Adapter supports FTP over SSL (FTPS) on Solaris and Linux. Oracle FTP Adapter also supports SFTP (Secure FTP) using SSH transport.

Note:

Oracle FTP Adapter supports SFTP server version 3 or later.

Inbound and Outbound Interactions

The Oracle File and FTP Adapters exchange files in the inbound and outbound directions. Based on the direction, the Oracle File and FTP Adapters perform different sets of tasks.

For inbound files sent to Oracle BPEL PM or Mediator, the Oracle File and FTP Adapters perform the following operations:

1. Poll the file system looking for matches.
2. Read and translate the file content. Native data is translated based on the native schema (NXSD) defined at design time.
3. Publish the translated content as an XML message.

This functionality of the Oracle File and FTP Adapters is referred to as the file read operation.

For outbound files sent from Oracle BPEL PM or Mediator, the Oracle File and FTP Adapters perform the following operations:

1. Receive messages from BPEL or Mediator.
2. Format the XML contents as specified at design time.
3. Produce output files. The output files can be created based on the following criteria: time elapsed, file size, and number of messages. You can also specify a combination of these criteria for output files.

This functionality of the Oracle File and FTP Adapters is referred to as the file write operation. This operation is known as a JCA outbound interaction.

For the inbound and outbound directions, the Oracle File and FTP Adapters use a set of configuration parameters. For example:

- The inbound Oracle File and FTP Adapters have parameters for the inbound directory where the input file appears and the frequency with which to poll the directory.
- The outbound Oracle File and FTP Adapters have parameters for the outbound directory in which to write the file and the file naming convention to use.

Note:

You must use the Adapter Configuration Wizard to modify the configuration parameters, such as publish size, number of messages, and polling frequency.

You *must not* manually change the value of these parameters in JCA files.

The file reader supports polling conventions and offers several postprocessing options. You can specify to delete, move, or leave the file as it is after processing the file. The file reader can split the contents of a file and publish it in batches, instead of as a single message. You can use this feature for performance tuning of the Oracle File and FTP Adapters. The file reader guarantees once and once-only delivery.

following sections for details about the read and write functionality of the Oracle File and FTP Adapters:

- Read File Concepts
- Write File Concepts
- Get File Concepts
- Put File Concepts

File Debatching

You can define the batch size using the `publishSize` parameter in the .jca file.

This property specifies if the file contains multiple messages and how many messages to publish to the BPEL process at a time.

For example, if a certain file has 11 records and this parameter is set to 2, then the file processes 2 records at a time and the final record is processed in the sixth iteration.

When a file contains multiple messages, you can choose to publish messages in a specific number of batches. This is referred to as debatching. During debatching, the file reader, on restart, proceeds from where it left off in the previous run, thereby avoiding duplicate messages. File debatching is supported for files in XML and native formats.

You can register a batch notification callback (Java class) which is invoked when the last batch is reached in a debatching scenario.

```
<service ...
  <binding.jca ...
    <property name="batchNotificationHandler">java://
oracle.sample.SampleBatchCalloutHandler </property>
```

where the property value must be `java://{custom_class}` and where `oracle.sample.SampleBatchCalloutHandler` must implement

```
package oracle.tip.adapter.api.callout.batch;
public interface BatchNotificationCallout extends Callout
{
    public void onInitiateBatch(String rootId,
                                String metaData)
        throws ResourceException;
    public void onFailedBatch(String rootId,
                              String metaData,
                              long currentBatchSize,
                              Throwable reason)
        throws ResourceException;
    public void onCompletedBatch(String rootId,
                                 String metaData,
                                            long finalBatchSize)
        throws ResourceException;
```

File ChunkedRead

The File Chunked Read operation enables you to process large files and uses a BPEL Invoke activity within a while loop to process the target file.

Specifically, the FileAdapter allows the BPEL process modeler to use an Invoke activity to retrieve a logical chunk from a huge file, enabling the file to stay within memory constraints. The process calls the chunked-interaction in a loop in order to process the entire file, one logical chunk at a time. The intent is to achieve de-batchability on a file's outbound processing.

You use the File Adapter Configuration Wizard to define a chunked file .jca file and the WSDL file.

Chunked Interaction File Adapter Processing

The FileAdapter translates the native data for a Chunked Read operation to XML and returns it as a BPEL variable.

To perform a Chunked Read, you typically create an Invoke activity within BPEL.

You also select `Chunked Synchronous Read` as the WSDL operation, using the File Adapter Configuration Wizard; you optionally use the Configuration Wizard to configure the file input directory and the filename which are placed in the `ChunkedInteractionSpec`.

Each call to the Invoke activity returns header values in addition to the payload.

These header values include: line number, column number, and indicators that specify if the End of File has been reached. You must ensure to copy the line/column numbers from the return header to the outbound headers for the subsequent call to the File Adapter. You can also specify the input directory/filename as header values if you want to.

File Chunked Interaction BPEL Invocation

The FileAdapter chunked interaction is invoked from BPEL. For native data files, line number and column number are additionally passed as header values.

The first time that the chunked interaction is called within the loop, the values for LineNumber and ColumnNumber are blank; for subsequent calls, these values come from the return values from the Invoke minus one (that is, the prior Invoke).

The BPEL Invoke calls ChunkedInteraction with the parameters provided in Table 4-3.

Table 4-3 BPEL Invoke Parameters for Chunked Interaction

Parameter	Where Obtained
`Physical/LogicalDirectory`	ChunkedInteractionSpec or from the BPEL header
`FileName`	ChunkedInteractionSpec or from BPEL header
`ChunkSizeBatchSize`	ChunkedInteractionSpec, analogous to PublishSize in de-batching
`LineNumber`	Header (optional)
`ColumnNumber`	Header (optional)
`RecordNumber`	Header (optional)

The ChunkSize Parameter

The ChunkSize parameter provides information related to the size of the file chunk for the Chunked Read operation; it defaults to 1 if you do not configure another value in `ChunkedInteractionSpec` (that is, using the File Adapter Configuration Wizard).

Specifically, the `ChunkSize` parameter governs the number of nodes or records (not lines) that are returned.

For example, if you have an address book as a native CSV file and you have specified a ChunkSize of 5, each call to the Invoke activity returns an XML file containing 5 address book nodes; that is, five rows of CSV records in XML format.

In that sense, the `ChunkSize` parameter is analogous to the `PublishSize` parameter used by the FileAdapter for an inbound transaction.

Example Rejection Handler Binding for Chunked Read

The following example shows how to configure a rejection handler for the chunked read reference binding.

Example - Rejection Handler Binding for Chunked Read

```
<reference name="ReadAddressChunk">
    <interface.wsdl
```

```
        interface="http://xmlns.oracle.com/pcbpel/adapter
          /file/ReadAddressChunk/
                #wsdl.interface(ChunkedRead_ptt)"/>
       <binding.jca config="ReadAddressChunk_file.jca">
         <property name="rejectedMessageHandlers" source=""
              type="xs:string" many="false" override="may"
                >file:///c:/temp</property>
       </binding.jca>
  </reference>
```

Using Line Number//Column Number or Record Number

After every Invoke, you must copy the return headers over to the outbound headers for the subsequent invoke. The LineNumber/ColumnNumber are used by the Adapter for book-keeping purposes only, and you must ensure that you copy these values from the return-headers back to the headers before the call to the chunked interaction.

RecordNumber, on the other hand, is used when the data is in XML format (as distinct from native data). In that sense, RecordNumber is mutually exclusive with LineNumber/ColumnNumber, which is used for native data book-keeping.

File Chunked Read Interaction Artifacts

See the below example for JCA file for Chunked Read.

Example - JCA File for Chunked Read

```
<interaction-spec className="oracle.tip.adapter.file.
                outbound.ChunkedInteractionSpec">
      <property name="PhysicalDirectory"
                    value="/tmp/chunked/in"/>
      <property name="FileName" value="dummy.txt"/>
      <property name="ChunkSize" value="10"/>
    </interaction-spec>
```

The below example shows the generated Adapter WSDL file for the Chunked Read interaction:

Example - Chunked Read WSDL File

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<definitions name="ReadAddressChunk"
   targetNamespace="http://xmlns.oracle.com/pcbpel
        /adapter/file/ReadAddressChunk/"
        xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://xmlns.oracle.com/
pcbpel/adapter/
               file/ReadAddressChunk/" xmlns:plt="http://schemas.xmlsoap.org/ws/
2003/05/partner-link/" xmlns:imp1="http://xmlns.oracle.com/pcbpel/demoSchema/csv">
    <documentation>Returns a finite chunk from the target
      file based on the chunk size parameter</documentation>
    <types>
        <schema targetNamespace="http://xmlns.oracle.com/
                  pcbpel/adapter/
             file/ReadAddressChunk/"
             xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://xmlns.oracle.com/pcbpel
             /demoSchema/csv"
               schemaLocation="xsd/address-csv.xsd"/>
             <element name="empty">
               <complexType/>
             </element>
```

```
        </schema>
    </types>
    <message name="Empty_msg">
        <part name="Empty" element="tns:empty"/>
    </message>
    <message name="Root-Element_msg">
        <part name=
         "Root-Element" element="imp1:Root-Element"/>
    </message>
    <portType name="ChunkedRead_ptt">
        <operation name="ChunkedRead">
            <input message="tns:Empty_msg"/>
            <output message="tns:Root-Element_msg"/>
        </operation>
    </portType>
    <plt:partnerLinkType name="ChunkedRead_plt">
        <plt:role name="ChunkedRead_role">
            <plt:portType
             name="tns:ChunkedRead_ptt"/>
        </plt:role>
    </plt:partnerLinkType>
</definitions>
```

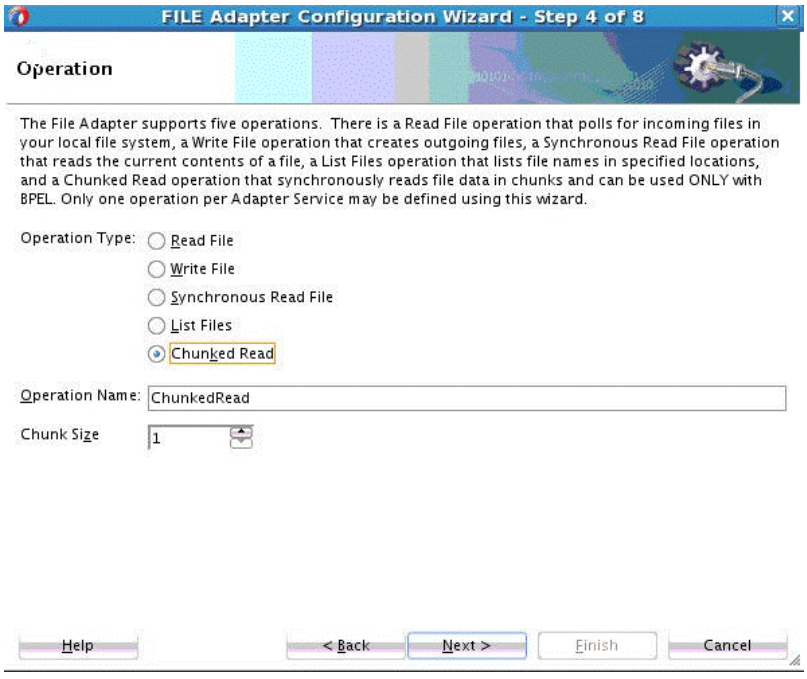
Using the File Adapter Configuration Wizard to Perform Chunked Read Interaction Modelling

You use the File Adapter Configuration Wizard to model chunked red interaction.

You can use the initial three screens of the File Adapter as you would to configure any other File Adapter operation.

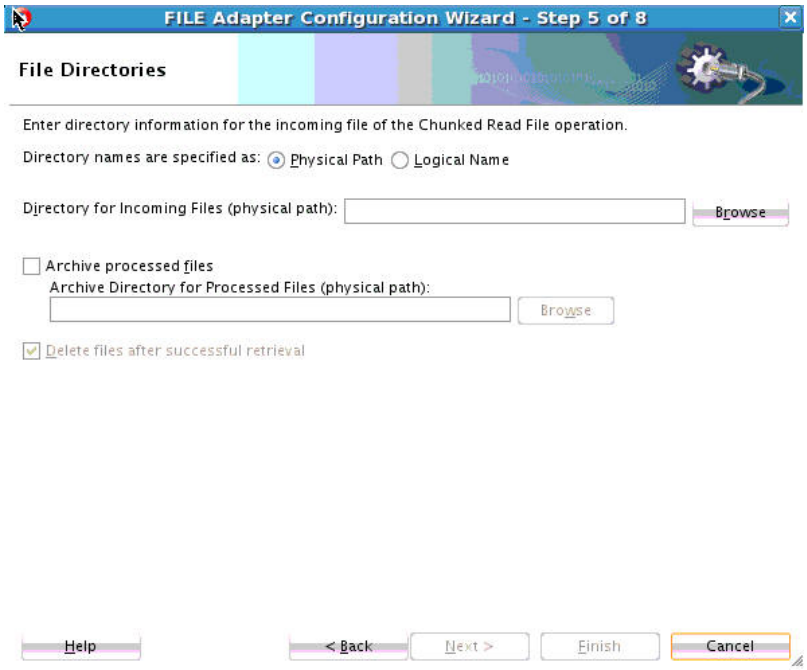
1. From the File Adapter Operation Screen, specify **Chunked Read**.

Figure 4-3 File Adapter Configuration Wizard Operation Page with Chunked Read Selected



2. On the File Directories, specifying the directory information for the Chunked Read operation.

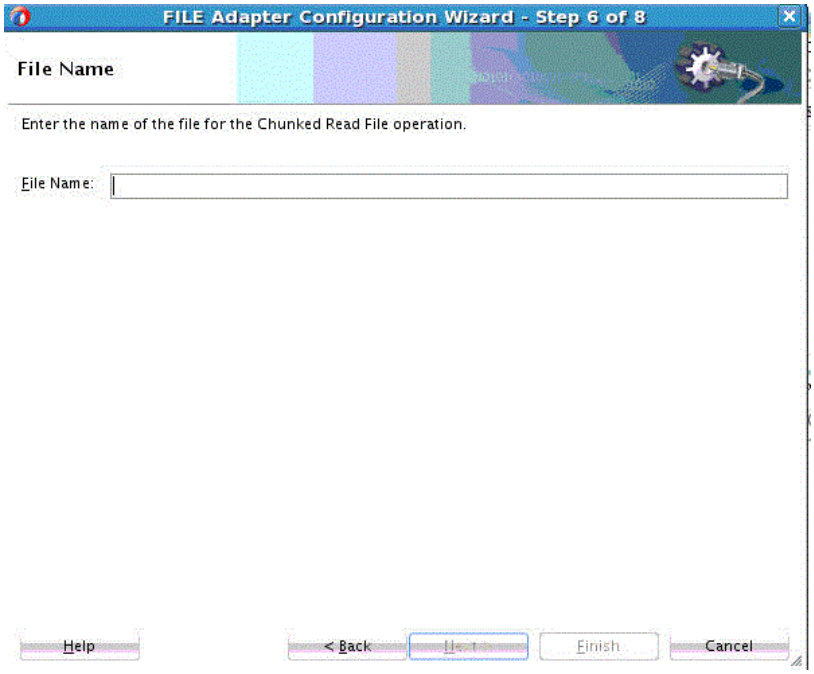
Figure 4-4 File Adapter Configuration Wizard Specifying the Directory Information for the Chunked Read Operation



3. On the **File Directory** page, indicate if the directory path you specify is physical or logical. Indicate or browse to find the file. Check and provide a directory to indicate the name of a physical path for the archived file. Deleting files after a successful retrieval is the default. Select **Next**.

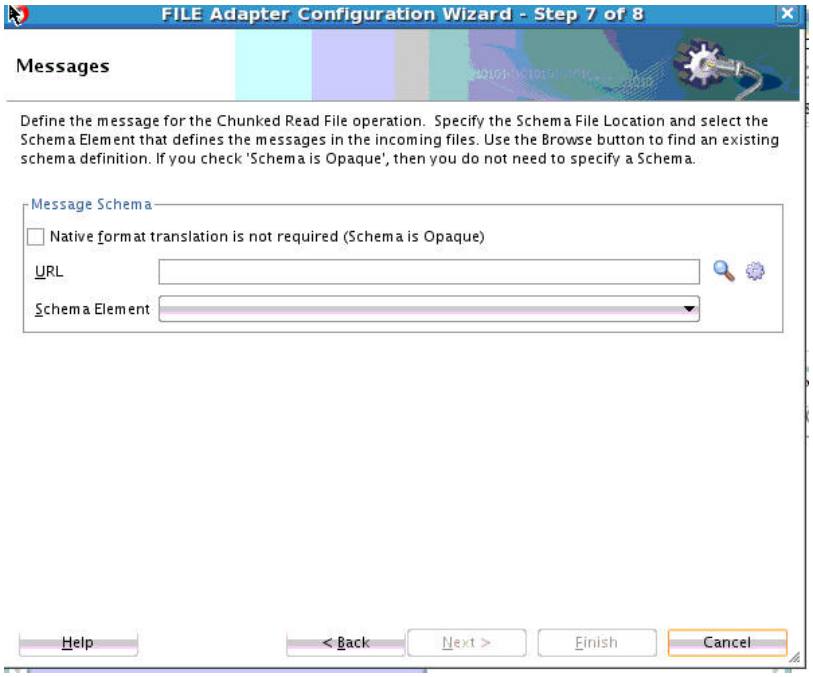
4. The File Adapter Configuration Wizard File Name Page is displayed. Indicate the name of the file for the Chunked Read operation. Select **Next**.

Figure 4-5 File Adapter Configuration Wizard File Name Page



5. The File Adapter Configuration Wizard Messages Screen appears.

Figure 4-6 File Adapter Configuration Wizard Messages Screen



6. The File Adapter Wizard Messages Screen enables you to define a message for the Chunked Read file operation. You can specify a file location and select the Schema Element that defines the messages for the incoming files. Use the browse button to find an existing schema definition. If you check **Schema is Opaque**, you do not must specify a Schema. Select **Next** or **Finish** to display the File Adapter Configuration Wizard and to display the directory path and name for the WSDL for the Chunked Read File operation.

Chunked Interaction Error Handling Summary

When a translation exception (that is, a bad record that violates the nXSD specification) is encountered, the return header is populated with the translation exception message that includes details such as the line/column where the error occurred.

However, a specific translation error does not result in a fault. Instead, it becomes a value in the return header. You must check the `jca.file.IsMessageRejected` and the `jca.file.RejectionReason` header values to check if rejection did happen. Additionally, you can also check the `jca.file.NoDataFound` header value

Skipping Bad Records

Using the `nxsd:uniqueMessageSeparator` construct enables the Adapter to skip bad records and continue processing the next set of records. (For more information on `uniqueMessageSeparator`, see Native Format Builder Wizard .)

If you do not use the `uniqueMessageSeparator`, the Adapter returns `EndOfFile` and thus causes the while loop to terminate.

Thus, the `uniqueMessageSeparator` construct is required if you want processing to continue and not assume an End of File situation. The absence of the `uniqueMessageSeparator` construct causes the rest of the file to be rejected as a single chunk-to reject the entire file.

Examples of Chunked Interaction Header and Rejected Chunked Interaction Messages

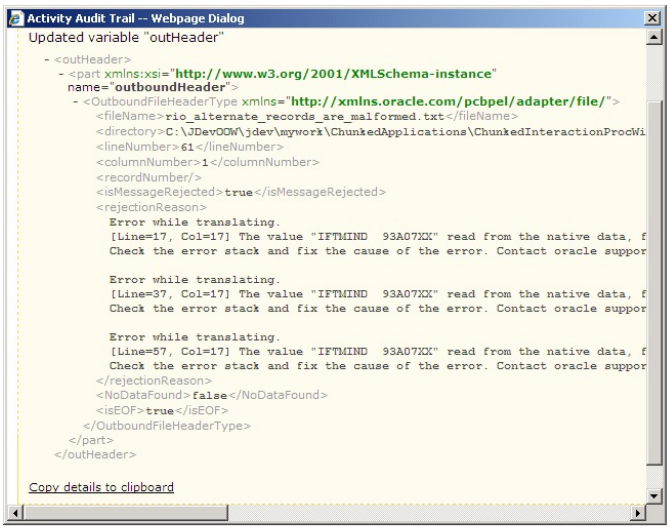
See Figure 4-7 for an example of the return header appearance in a scenario that employs the `nxsd:uniqueMessageSeparator` construct.

This scenario shows a chunked interaction with a file that had six records (each of which was complex) and each alternate record was malformed. In the scenario, the `ChunkSize` used was five.

The `returnHeader` shows that the messages have been rejected (`isMessageRejected=true`) and the rejection reason is populated for the three malformed records: specifically, the records at line 17, 37, and 57 were malformed.

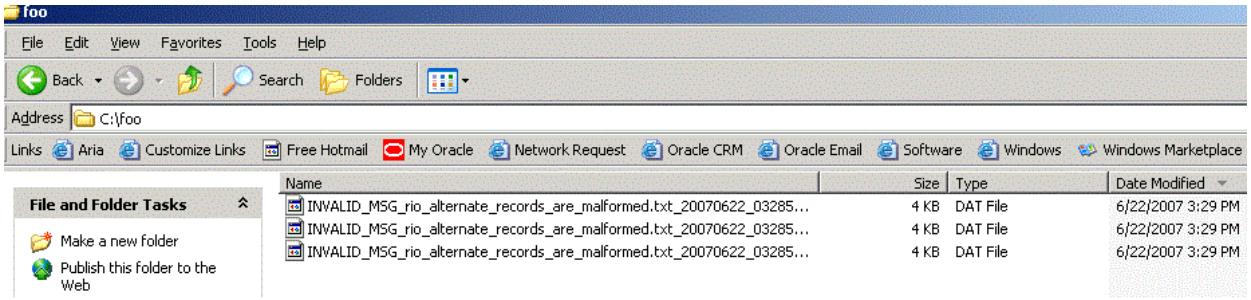
The `NoDataFound` parameter is set to false, which means that the data for the remaining three records is returned.

Figure 4-7 Return Outbound Header Appearance when nxsd:uniqueMessageSeparator is Used



The same records are also rejected to the user-configured rejection folder (C:\foo in this case). See Figure 4-8.

Figure 4-8 Chunked Read Interaction Rejected Messages in Rejection Folder



File Sorting

When files must be processed by Oracle File and FTP Adapters in a specific order, you can you use the File Sorting Functionality of the Adapter.. For example, you can configure the sorting parameters for Oracle File and FTP Adapters to process files in ascending or descending order by time stamps.

You must meet the following prerequisites for sorting scenarios of Oracle File and FTP Adapters:

- Use a synchronous operation

- Add the following property to the inbound JCA file:

```
<property name="ListSorter"
value="oracle.tip.adapter.file.inbound.listing.TimestampSorterAscending"/>
<property name="SingleThreadModel" value="true"/>
```

Dynamic Outbound Directory and File Name Specification

The Oracle File and FTP Adapters enable you to dynamically specify the logical or physical name of the outbound file or outbound directory. For information about how to specify dynamic outbound directory, see Outbound File Directory Creation.

Security

The Oracle FTP Adapter supports FTP over SSL (FTPS) and Secure FTP (SFTP) to enable secure file transfer over a network.

For more information, see Using Secure FTP with the and Using SFTP with .

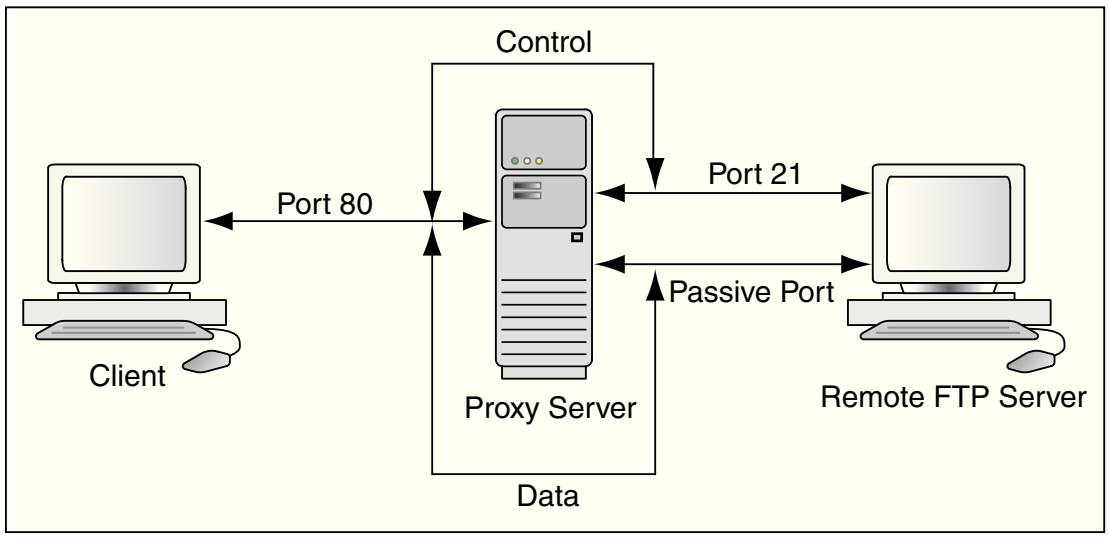
Nontransactional

The Oracle File Adapter picks up a file from an inbound directory, processes the file, and sends the processed file to an output directory. However, during this process if a failover occurs in the Oracle RAC back end or in an SOA managed server, then the file is processed twice because of the nontransactional nature of Oracle File Adapter. As a result, there can be duplicate files in the output directory.

Proxy Support

You can use the proxy support feature of the Oracle FTP Adapter to transfer and retrieve data to and from the FTP servers that are located outside a firewall or can only be accessed through a proxy server. A proxy server enables the hosts in an intranet to indirectly connect to hosts on the Internet. Figure 4-9 shows how a proxy server creates connections to simulate a direct connection between the client and the remote FTP server.

Figure 4-9 Remote FTP Server Communication Through a Proxy Server



To use the HTTP proxy feature, your proxy server must support FTP traffic through HTTP Connection. In addition, only passive data connections are supported with this

feature. For information about how to configure the Oracle FTP Adapter, see Configuring for HTTP Proxy.

No Payload Support

For Oracle BPEL PM and Mediator, the Oracle File and FTP Adapters provide support for publishing only file metadata such as file name, directory, file size, and last modified time to a BPEL process or Mediator and excludes the payload. The process can use this metadata for subsequent processing. For example, the process can call another reference and pass the file and directory name for further processing.You can use the Oracle File and FTP Adapters as a notification service to notify a process whenever a new file appears in the inbound directory. To use this feature, select the **Do not read file content** check box in the JDeveloper wizard while configuring the "Read operation."

Large Payload Support

For Oracle BPEL PM and Mediator, the Oracle File Adapter provides support for transferring large files as attachments. To use this feature, select the **Read File As Attachment** check box in the JDeveloper Configuration wizard while configuring the Read operation.

This option opaquely transfers a large amount of data from one place to another as attachments. For example, you can transfer large MS Word documents, images, and PDFs without processing their content within the composite application. For information about how to pass large payloads as attachments, see Read File As Attachments.

Additionally, the Oracle File Adapter provides you with the ability to write files as an attachment. When you write files as attachments, and also have a normal payload, it is the attached file that is written, and the payload is ignored.

Note:

You must not pass large payloads as opaque schemas.

File-Based Triggers

You can use the Oracle File and FTP Adapters, which provide support for file-based triggers, to control inbound adapter endpoint activation. For information about how to use file-based triggers, see File Polling.

Pre-Processing and Post-Processing of Files

The process modeler may encounter situations where files must be pre-processed before they are picked up for processing or post-processed before the files are written out to the destination folder. For example, the files that the Oracle File and FTP adapters receive may be compressed or encrypted and the adapter must decompress or decrypt the files before processing. In this case, you must use a custom code to decompress or decrypt the files before processing. The Oracle File and FTP Adapters supports the use of custom code that can be plugged in for pre-processing or post-processing of files.

The implementation of the pre-processing and post-processing of files is restricted to the following communication modes of the Oracle File and FTP Adapters:

- Read File or Get File
- Write File or Put File
- Synchronous Read File
- Chunked Read

This section contains the following topics:

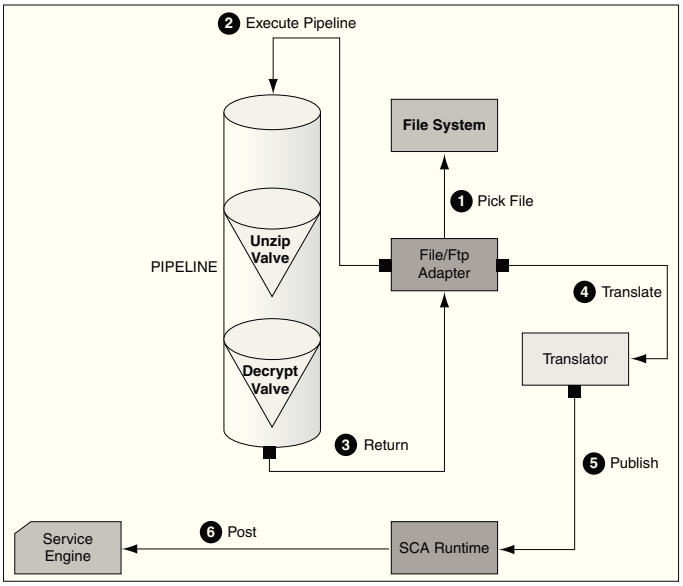
Mechanism For Pre-Processing and Post-Processing of Files

The mechanism for pre-processing and post-processing of files is configured as pipelines and valves. This section describes the concept of pipelines and valves.

A **pipeline** consists of a series of custom-defined valves. A pipeline loads a stream from the file system, subjects the stream to processing by an ordered sequence of valves, and after the processing returns the modified stream to the adapter.

A **valve** is the primary component of execution in a processing pipeline. A valve processes the content it receives and forwards the processed content to the next valve. For example, in a scenario where the Oracle File and FTP Adapters receive files that are encrypted and zipped, you can configure a pipeline with an unzip valve followed by a decryption valve. The unzip valve extracts the file content before forwarding it to the decryption valve, which decrypts the content and the final content is made available to the Oracle File or FTP Adapter as shown in Figure 4-10.

Figure 4-10 A Sample Pre-Processing Pipeline



Configuring a Pipeline

Configuring the mechanism for pre-processing and post-processing of files requires defining a pipeline and configuring it in the corresponding JCA file.

To configure a pipeline, you must perform the following steps:

- Implementing and Extending Valves
- Compiling the Valves
- Creating a Pipeline
- Adding the Pipeline to the SOA Project Directory
- Registering the Pipeline

Implementing and Extending Valves

- All valves must implement `Valve` or `StagedValve` interface.

  Tip:

  You can extend either the `AbstractValve` or the `AbstractStagedValve` class based on business requirement rather than implementing a valve from the beginning.

The below example is a sample valve interface.

Example - The Valve Interface

```
package oracle.tip.pc.services.pipeline;

import java.io.IOException;
/** <p>
 *  Valve component is resposible for processing the input stream
 *  and returning a modified input stream.
 *  The <code>execute()</code> method of the valve gets invoked
 *  by the caller (on behalf) of the pipeline. This method must
 *  return the input stream wrapped within an InputStreamContext.
 *  The Valve is also responsible for error handling specifically
 *
 *  The Valve can be marked as reentrant in which case the caller
 *  must call the <code>execute()</code> multiple times and each
 *  invocation must return a new input stream. This is useful, if
 *  you are writing an UnzipValve since each iteration of the valve
 *  must return the input stream for a different zipped entry.
 *  <b> You must note that only the first Valve in the pipeline can
 *  be reentrant </b>
 *
 *  The Valve has another flavor <code>StagedValve</code> and if
 *  the valve implements StagedValve, then the valve must store
 *  intermediate content in a staging file and return it whenever
 *  required.
 * </p>
 */
public interface Valve
{
    /**
     * Set the Pipeline instance. This parameter can be
     * used to get a reference to the PipelineContext instance.
     * @param pipeline
     */
    public void setPipeline(Pipeline pipeline);

    /** Returns the Pipeline instance.
```

```
        * @return
        */
       public Pipeline getPipeline();

       /** Returns true if the valve has more input streams to return
        *  For example, if the input stream is from a zipped file, then
        *  each invocation of <code>execute()</code> returns a different
        *  input stream once for each zipped entry. The caller calls
        *  <code>hasNext()</code> to check if more entries are available
        * @return true/false
        */
       public boolean hasNext();

       /** Set to true if the caller can call the valve
            multiple times
        *  e.g. in case of ZippedInputStreams
        * @param reentrant
        */
       public void setReentrant(boolean reentrant);

       /** Returns true if the valve is reentrant.
      * @return
      */
     public boolean isReentrant();

     /** The method is called by pipeline to
         return the modified input stream
      * @param in
      * @return InputStreamContext that wraps
      * the input stream along with required metadata
      * @throws PipelineException
      */
     public InputStreamContext execute(InputStreamContext
          in) throws
             PipelineException, IOException;

       /**
        * This method is called by the pipeline after the caller
        * publishes the
        * message to the SCA container.
        * In the case of a zipped file, this method
        * gets called repeatedly, once
        * for each entry in the zip file.
        * This should be used by the Valve to do
        * additional tasks such as
        * delete the staging file that has been processed
        * in a reentrant
        * scenario.
        * @param in The original InputStreamContext returned from
<code>execute()</code>
        */
       public void finalize(InputStreamContext in);

     /**Cleans up intermediate staging files, input streams
      * @throws PipelineException, IOException
      */
     public void cleanup() throws PipelineException, IOException;
  }
```

The `StagedValve` stores intermediate content in staging files. The example below shows the `StagedValve` interface extending the `Valve` interface.

Example - The StagedValve Interface Extending the Valve Interface

```
package oracle.tip.pc.services.pipeline;

import java.io.File;

/**
 * A special valve that stages the modified
 * input stream in a staging file.
 * If such a <code>Valve</code> exists, then
 * it must return the staging file containing
 * the intermediate data.
 */
public interface StagedValve extends Valve {

      /**
       * @return staging file where the valve
       * stores its intermediate results
       */
      public File getStagingFile();
}
```

The below example is a sample of an `AbstractValve` class implementing the `Valve` interface.

Example - The AbstractValve Class Implementing the Valve Interface

```
package oracle.tip.pc.services.pipeline;

import java.io.IOException;

/**
 * A bare bone implementation of Valve. The user should
 * extend from
 * AbstractValve rather than implementing a Valve from scratch
 *
 */
public abstract class AbstractValve implements Valve {

      /**
       * The pipeline instance is stored as a member
       */
      private Pipeline pipeline = null;

      /**
       * If reentrant is set to true, then the Valve must adhere to the
following:
       * i) It must the first valve in the pipeline ii)
       *  Must implement hasNext
       * method and return true if more input
       * streams are available A reentrant
       * valve will be called by the pipeline
       * more than once and each time the
       * valve must return a different input stream,
       * for example Zipped entries
       * within a zip file
       */
      private boolean reentrant = false;

      /*
       * Save the pipeline instance.
       *
```

```
 * @see
 * oracle.tip.pc.services.pipeline.Valve#setPipeline
 * (oracle.tip.pc.services.pipeline.Pipeline)
 */
public void setPipeline(Pipeline pipeline) {
      this.pipeline = pipeline;
}

/*
 * Return the pipeline instance (non-Javadoc)
 *
 * @see oracle.tip.pc.services.pipeline.
 * Valve#getPipeline()
 */
public Pipeline getPipeline() {
      return this.pipeline;
}

/*
 * Return true if the valve is reentrant (non-Javadoc)
 *
 * @see oracle.tip.pc.services.pipeline.
 * Valve#isReentrant()
 */
public boolean isReentrant() {
      return this.reentrant;
}

/*
 * If set to true, the valve is reentrant (non-Javadoc)
 *
 * @see oracle.tip.pc.services.pipeline.
 * Valve#setReentrant(boolean)
 *
 */
public void setReentrant(boolean reentrant) {
      this.reentrant = reentrant;
}

/*
 * By default, set to false For valves
 * that can return more than one
 * inputstreams to callers, this parameter
 *  must return true/false depending
 * on the availability of input streams (non-Javadoc)
 *
 * @see oracle.tip.pc.services.pipeline.Valve#hasNext()
 */
public boolean hasNext() {
      return false;
}

/*
 * Implemented by concrete valve (non-Javadoc)
 *
 * @see oracle.tip.pc.services.pipeline.
 * Valve#execute(InputStreamContext)
 */
public abstract InputStreamContext
       execute(InputStreamContext in)
            throws PipelineException, IOException;
```

```
    /*
     * Implemented by concrete valve (non-Javadoc)
     *
     * @see
     *       oracle.tip.pc.services.pipeline.Valve#finalize
     *       (oracle.tip.pc.services.pipeline.In
     *                       putStreamContext)
     *        /
    public abstract void finalize(InputStreamContext in);

    /*
     * Implemented by concrete valve (non-Javadoc)
     *
     * @see oracle.tip.pc.services.pipeline.
     * Valve#cleanup()
     */
    public abstract void cleanup()
            throws PipelineException, IOException;

}
```

The below example shows the `AbstractStagedValve` class extending the `AbstractValve` class.

Example - The AbstractStagedValve Class Extending the AbstractValve Class

```
package oracle.tip.pc.services.pipeline;

import java.io.File;
import java.io.IOException;

public abstract class AbstractStagedValve
           extends AbstractValve implements
            StagedValve {

    public abstract File getStagingFile();

    public abstract void cleanup() throws IOException,
            PipelineException;

    public abstract InputStreamContext
                 execute(InputStreamContext in)
               throws IOException, PipelineException;

}
```

For more information on valves, see Oracle JCA Adapter Valves.

Compiling the Valves

- You must use the `bpm-infra.jar` file to compile the valves. The `bpm-infra.jar` file is located at `$MW_HOME/AS11gR1SOA/soa/modules/oracle.soa.fabric_11.1.1/bpm-infra.jar`.

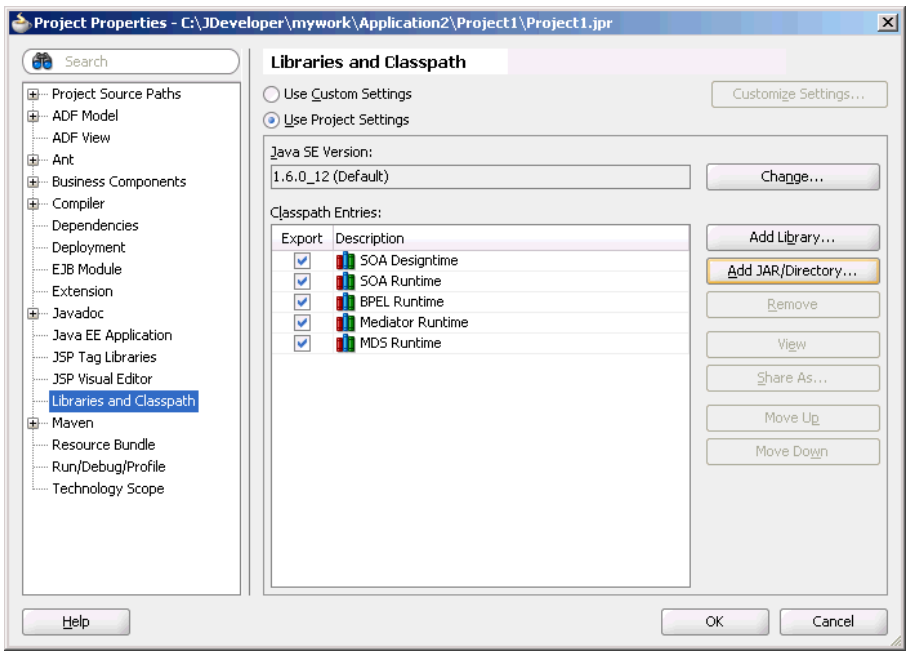
  1. Reference the SOA project to the `bpm-infra.jar` file, by using the following procedure:

     a. In the Application Navigator, right-click the SOA project.

     b. Select **Project Properties**. The Project Properties dialog is displayed.

c. Click **Libraries and Classpath**. The Libraries and Classpath pane is displayed as shown in Figure 4-11.

Figure 4-11 The Project Properties Dialog



d. Click **Add Jar/Directory**. The Add Archive or Directory dialog is displayed.

e. Browse to select the `bpm-infra.jar` file. The `Bpm-infra.jar` file is located at `$MW_HOME/AS11gR1SOA/soa/modules/oracle.soa.fabric_11.1.1/bpm-infra.jar`.

f. Click **OK**. The `bpm-infra.jar` file is listed under Classpath Entries.

2. Compile the valves using the `bpm-infra.jar` file.

3. Make the JAR file containing the compiled valves available to the Oracle WebLogic Server classpath by adding the jar file to the `soainfra` domain classpath. For example, `$MW_HOME/user_projects/domains/soainfra/lib`.

Note:

Ensure that you compile `bpm-infra.jar` with JDK 6.0 to avoid compilation error such as `class file has wrong version 50.0, should be 49.0`.

Creating a Pipeline

- To configure a pipeline, you must create an XML file that conforms to the following schema:

  Example - XML for Pipeline Creation

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://
```

```
www.oracle.com/adapter/pipeline">
  <xs:element name="pipeline">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="valves">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="valve"
                       maxOccurs="unbounded">
                <xs:complexType
mixed="true">
                <xs:attribute name="reentrant"
                   type="xs:NMTOKEN"
                         use="optional" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:attribute name="useStaging"
              type="xs:NMTOKEN" use="optional" />
    <xs:attribute name="batchNotificationHandler"
            type="xs:NMTOKEN" use="
optional" />
  </xs:element>
</xs:schema
```

The following is a sample XML file configured for a pipeline with two valves, `SimpleUnzipValve` and `SimpleDecryptValve`:

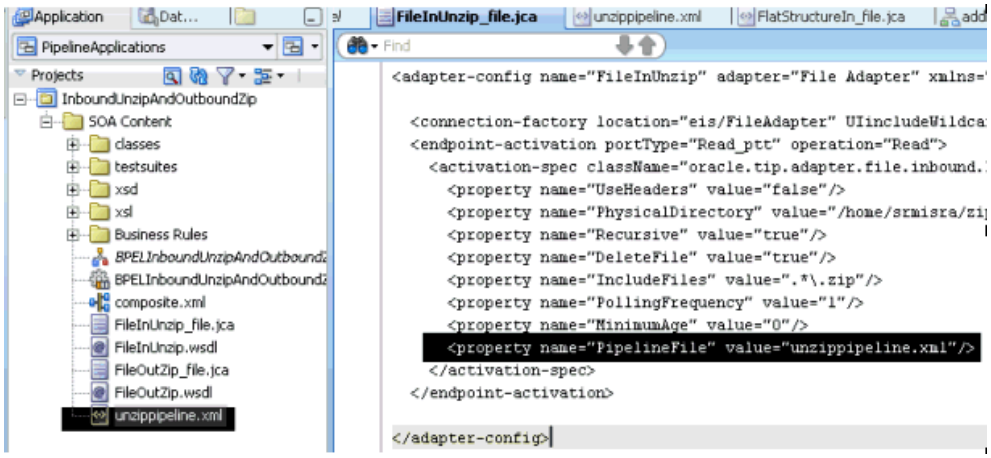
Example - XML file configured for a Pipeline with two valves

```
<?xml version="1.0"?>
<pipeline xmlns=
          "http://www.oracle.com/adapter/pipeline">
<valves>
        <valve>valves.SimpleUnzipValve</valve>
        <valve> valves.SimpleDecryptValve </valve>
</valves>
</pipeline>
```

Adding the Pipeline to the SOA Project Directory

- You must add the *`pipeline`*`.xml` file to the SOA project directory. This step is required to integrate the pipeline with the Oracle File or FTP Adapter. Figure 4-12 shows a sample *`pipeline`*`.xml` file (`unzippipeline.xml`) added to the `InboundUnzipAndOutboundZip` project.

Figure 4-12 Project with unzippipeline.xml File



Registering the Pipeline

- The pipeline that is a part of the SOA project must be registered by modifying the inbound JCA file, by adding the following property:

  ```
  <property name="PipelineFile" value="pipeline.xml"/>
  ```

  For example, in the JCA file shown in Figure 4-12, `FileInUnzip_file.jca`, the following property has been added to register an `Unzip` pipeline with an Oracle File Adapter:

  ```
  <property name="PipelineFile" value="unzippipeline.xml"/>
  ```

  There may be scenarios involving simple valves. A simple valve is one that does not require additional metadata such as re-entrancy, and `batchNotificationHandlers`. If the scenario involves simple valves, then the pipeline can be configured as an `ActivationSpec` or an `InteractionSpec` property as shown in the following sample:

  Example - Pipeline Configuration with Simple Valves

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <adapter-config name="FlatStructureIn"
                  adapter="File Adapter"
  xmlns="http://platform.integration.
                oracle/blocks/adapter/fw/metadata">

    <connection-factory location="eis/FileAdapter"
                UIincludeWildcard="*.txt"
  adapterRef=""/>
    <endpoint-activation operation="Read">
      <activation-spec
  className="oracle.tip.adapter.file.
                      inbound.FileActivationSpec">
        <property name="UseHeaders" value="false"/>
        <property name=
                "LogicalDirectory" value="InputFileDir"/>
        <property name="Recursive" value="true"/>
        <property name="DeleteFile" value="true"/>
        <property name="IncludeFiles" value=".*\.txt"/>
        <property name="PollingFrequency" value="10"/>
        <property name="MinimumAge" value="0"/>
        <property name="OpaqueSchema" value="false"/>
  ```

```
      </activation-spec>
    </endpoint-activation>

  </adapter-config>
```

Note:

There is no space after the comma (`,`) in the `PipelineValves` property `value`.

Note:

If you configure a pipeline using the `PipelineValve` property, then you cannot configure additional metadata such as Re-entrant Valve and Batch Notification Handler. Additional metadata can be configured only with `PipelineFile` that is used for the XML-based approach.

Using a Re-Entrant Valve For Processing Zip Files

The re-entrant valve enables you to process individual entries within a zip file. In a scenario that involves processing all entries within a zip file, wherein each entry is encrypted using the Data Encryption Standard (DES), you can configure the valve by adding the `reentrant="true"` attribute to the unzip valve as follows:

Example - Configuring the reentrant=true Attribute

```
<?xml version="1.0"?>
<pipeline xmlns="http://www.oracle.com/adapter/pipeline">
<valves>
        <valve reentrant="true">valves.ReentrantUnzipValve</valve>
        <valve> valves.SimpleDecryptValve </valve>
</valves>
</pipeline>
```

In this example, the pipeline invokes the `ReentrantUnzipValve` and then the `SimpleDecryptValve` repeatedly in the same order until the entire zip file has been processed. In other words, the `ReentrantUnzipValve` is invoked first to return the data from the first zipped entry, which is then fed to the `SimpleDecryptValve` for decryption, and the final content is returned to the Adapter. The process repeats until all the entries within the zip file are processed.

Additionally, the valve must set the message key using the `setMessageKey()` API. For more information refer to An Unzip Valve for processing Multiple Files.

Error Handling For Zip Files

If there are *translation errors* for individual entries within the zip file, entries with the translation errors are rejected and the other entries are processed.

If there are *errors during the publish operation*, the publish operation is retried and the retry semantic holds. If the retry semantic does not hold, then the original file is rejected and the pipeline ends.

Configuring the Batch Notification Handler

The `BatchNotificationHandler` API is used with the Oracle File and FTP Adapter inbound de-batchability. In a de-batching scenario, each file contains multiple messages, and some sort of bookkeeping is required for crash-recovery. This is

facilitated by the `BatchNotificationHandler` API, which lets you receive notification from the pipeline whenever a batch begins, occurs, or ends. The below example is the `BatchNotificationHandler` interface:

Example - BatchNotification Handler

```
package oracle.tip.pc.services.pipeline;

/*
 * Whenever the caller processes de-batchable files,
 * each file can
 * have multiple messages and this handler
 * allows the user to plug in
 * a notification mechanism into the pipeline.
 *
 * This is particularly useful in crash recovery
 * situations
 */
 public interface BatchNotificationHandler {

      /*
       * The Pipeline instance is set by the
       * PipelineFactory when the
       * BatchNotificationHandler instance is created
       */
      public void setPipeline(Pipeline pipeline);

      public Pipeline getPipeline();
      /*
       * Called when the BatchNotificationHandler
       * is instantiated
       */

      public void initialize();
      /*
       * Called by the adapter when a batch begins,
       * the implementation must
       * return
         * a BatchContext instance with the
         * following information:
         * i) batchId: a unique
         * id that will be returned
         * every time onBatch is
         * invoked by called
         * ii)line/col/record/offset:
         * for error recovery cases
       */
      public BatchContext onBatchBegin();

      /*
       * Called by the adapter
       * when a batch is submitted.
       * The parameter holds the
       * line/column/record/offset for the successful batch
       * that is published.
       * Here the implementation
       * must save these in
       * order to recover from
       * crashes
       */
```

```
      public void onBatch(BatchContext ctx);

      /*
       * Called by the adapter when a batch
       * completes.
      * This must be used to clean up
       */
      public void onBatchCompletion
           (boolean success);
}
```

To use a pipeline with de-batching, you must configure the pipeline with a `BatchNotificationHandler` instance. See the below example.

Example - Configuring the Pipeline with a BatchNotificationHandler Instance

```
<?xml version="1.0"?>
<pipeline xmlns="http://www.oracle.com
           /adapter/pipeline"
batchNotificationHandler="oracle.tip.pc.services.
pipeline.ConsoleBatchNotificationHandler">
<valves>
      <valve reentrant="true">valves.
                  SimpleUnzipValve</valve>
      <valve>valves.SimpleDecryptValve</valve>
</valves>
</pipeline>
```

Error Handling

The Oracle File Adapter and Oracle FTP Adapter provide inbound error handling capabilities, such as the `uniqueMessageSeparator` property.

In the case of debatching (multiple messages in a single file), messages from the first bad message to the end of the file are rejected. If each message has a unique separator and that separator is not part of any data, then rejection can be more fine-grained. In these cases, you can define a `uniqueMessageSeparator` property in the schema element of the native schema to have the value of this unique message separator. This property controls how the adapter translator works when parsing through multiple records in one file (debatching). This property enables recovery even when detecting bad messages inside a large batch file. When a bad record is detected, the adapter translator skips to the next unique message separator boundary and continues from there. If you do not set this property, then all records that follow the record with errors are also rejected.

The below example provides an example of using the `uniqueMessageSeparator` property:

Example - Schema File Showing Use of uniqueMessageSeparator Property

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/
                2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/
                        pcbpel/nxsd"
            targetNamespace=
                  "http://TargetNamespace.com/Reader"
            xmlns:tns=
                "http://TargetNamespace.com/Reader"
            elementFormDefault="qualified"
```

```
                attributeFormDefault="unqualified"
            nxsd:encoding="US-ASCII" nxsd:stream="chars"
            nxsd:version="NXSD"
                nxsd:uniqueMessageSeparator="${eol}">
  <xsd:element name="emp-listing">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="emp" minOccurs="1"
                        maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="GUID"
                           type="xsd:string"
                           nxsd:style="terminated"
                           nxsd:terminatedBy=","
                           nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Designation"
                           type="xsd:string"
                           nxsd:style="terminated"
                           nxsd:terminatedBy=","
                           nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Car" type="xsd:string"
                           nxsd:style="terminated"
                           nxsd:terminatedBy=","
                           nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Labtop" type="xsd:string"
                           nxsd:style="terminated"
                           nxsd:terminatedBy=","
                           nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Location"
                              type="xsd:string"
                           nxsd:style="terminated"
                                 nxsd:terminatedBy=","
                           nxsd:quotedBy="&quot;">
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
<!--NXSDWIZ:D:\work\jDevProjects\Temp_BPEL_process
                \Sample2\note.txt:-->
<!--USE-HEADER:false:-->
```

For information about handling rejected messages, connection errors, and message errors, see Handling Rejected Messages .

Sending a Malformed XML File to a Local File System Folder

During an Inbound Read operation, if a malformed XML file is read, the malformed file results in an error. The errored file is by default sent to the remote file system for archival.

The errored file can be archived at a *local* file system by specifying the `useRemoteErrorArchive` property in the `jca` file and setting that property to `false`.

The default value for this property is `true`.

Threading Model

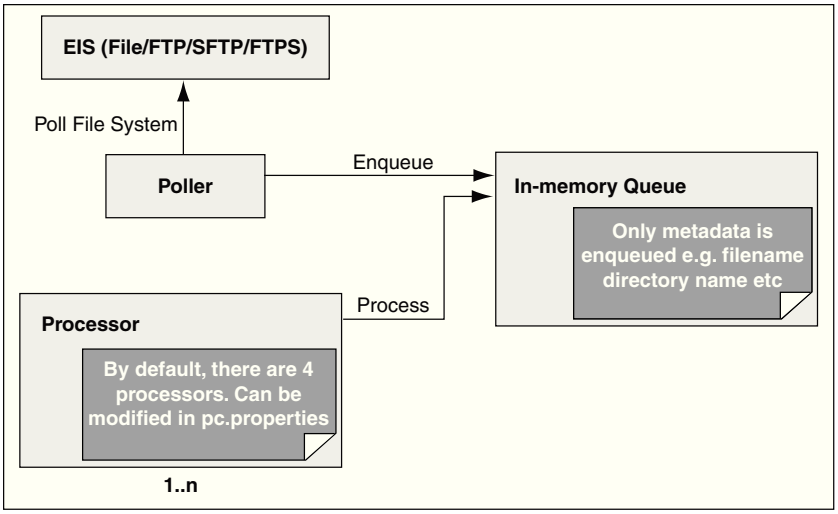
This section describes the threading models that Oracle File and FTP Adapters support. An understanding of the threading models is required to throttle or de-throttle the Oracle File and FTP Adapters. The Oracle File and FTP Adapters use the following threading models:

- Default Threading Model
- Modified Threading Model

Default Threading Model

In the default threading model, a poller is created for each inbound Oracle File or FTP Adapter endpoint. The poller enqueues file metadata into an in-memory queue, which is processed by a global pool of processor threads. Figure 4-13 shows a default threading model.

Figure 4-13 Default Threading Model



The following steps highlight the functioning of the default threading model:

1. The poller periodically looks for files in the input directory. The interval at which the poller looks for files is specified using the `PollingFrequency` parameter in the inbound `JCA` file.

2. For each new file that the poller detects in the configured inbound directory, the poller enqueues information such as file name, file directory, modified time, and file size into an internal in-memory queue.

Note:

New files are ones that are not being processed.

---

3. A global pool of processor worker threads wait to process from the in-memory queue.
4. Processor worker threads pick up files from the internal queue, and perform the following actions:
   a. Stream the file content to an appropriate translator (for example, a translator for reading text, binary, XML, or opaque data.)
   b. Publish the XML result from the translator to the SCA infrastructure.
   c. Perform the required postprocessing, such as deletion or archival after the file is processed.

Modified Threading Model

You can modify the default threading behavior of Oracle File and FTP Adapters. Modifying the threading model results in a modified throttling behavior of the Oracle File and FTP Adapters. The following sections describe the modified threading behavior of the Oracle File and FTP Adapters:

- Single Threaded Model
- Partitioned Threaded Model

Single Threaded Model

The single threaded model is a modified threaded model that enables the poller to assume the role of a processor. The poller thread processes the files in the same thread. The global pool of processor threads is not used in this model. You can define the property for a single threaded model in the inbound JCA file as follows:

Example - Defining the Property for a Single-Threaded Model

```
<activation-spec className="oracle.tip.adapter.file.
   inbound.FileActivationSpec">
   <property../>
   <property name="SingleThreadModel" value="true"/>
   <property../>
</activation-spec>
```

Partitioned Threaded Model

The partitioned threaded model is a modified threaded model in which the in-memory queue is partitioned and each composite application receives its own in-memory queue. The Oracle File and FTP Adapters are enabled to create their own processor threads rather than depend on the global pool of processor worker threads for processing the enqueued files. You can define the property for a partitioned model in the inbound JCA file. See the example below.

Example - Defining the Property for a Partitioned Model in the Inbound JCA File

```
<activation-spec
className="oracle.tip.adapter.file.inbound.
   FileActivationSpec">
  <property../>
  <property name="ThreadCount" value="4"/>
  <property../>
</activation-spec>
```

In the preceding example for defining the property for a partitioned model:

- If the `ThreadCount` property is set to `0`, the threading behavior is like that of the single threaded model.
- If the `ThreadCount` property is set to `-1`, the global thread pool is used, as in the default threading model.
- The maximum value for the `ThreadCount` property is `40`.

Performance Tuning

The Oracle File and FTP Adapters support the performance tuning feature by providing knobs to throttle the inbound and outbound operations. The Oracle File and FTP Adapters also provide parameters that you can use to tune the performance of outbound operations.

For more information about performance tuning, see Oracle JCA Adapter Tuning Guide in this document.

High Availability

The Oracle File and FTP Adapters support the high availability feature for the active-active topology with Oracle BPEL Process Manager and Mediator service engines. They support this feature for both inbound and outbound operations.

Multiple Directories

The Oracle File and FTP Adapters support polling multiple directories within a single activation. You can specify multiple directories in JDeveloper as distinct from a single directory. This is applicable to both physical and logical directories.

Note:

If the inbound Oracle File Adapter is configured for polling multiple directories for incoming files, then all the top-level directories (inbound directories where the input files appear) must exist before the file reader starts polling these directories.

After selecting the inbound directory or directories, you can also specify whether the subdirectories must be processed recursively. If you check the Process Files Recursively option, then the directories would be processed recursively. By default, this option is selected, in the File Directories page, as shown in Figure 4-14.

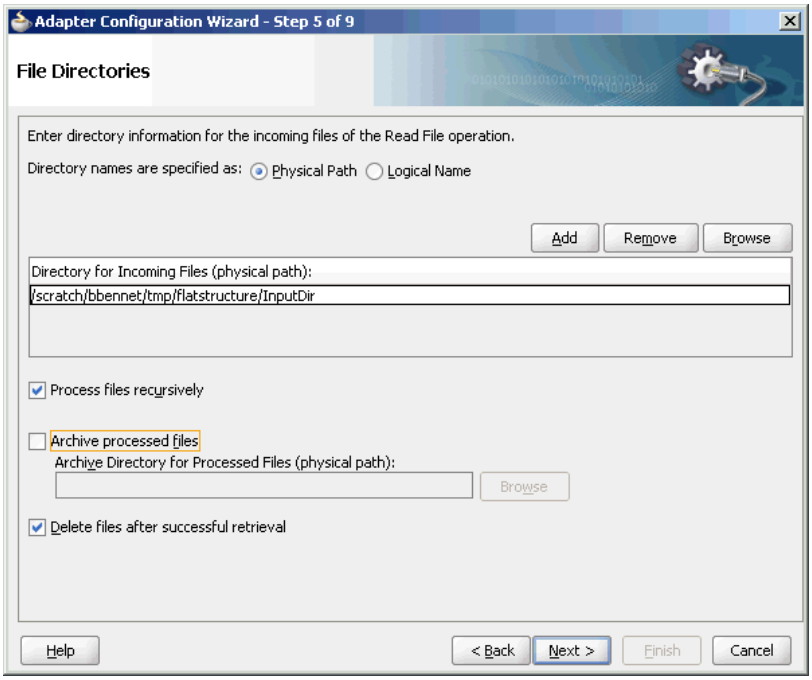
When you choose multiple directories, the generated JCA files use semicolon(;) as the separator for these directories. However, you can change the separator to something else. If you do so, manually add `DirectorySeparator="`*`chosen separator`*`"` in the generated JCA file. For example, to use comma (,) as the separator, you must first change the separator to "," in the Physical directory and then add `<property name="DirectorySeparator" value=","/>`, in the JCA file.

Additionally, if you choose to process directories recursively and one or more subdirectories do not have the appropriate permissions, the inbound adapter throws an exception during processing. To ignore this exception, you must define a binding property with the name `ignoreListingErrors` in your `composite.xml` as shown in the example below.

Example - Defining a Binding Property with the name ignoreLIstingErrors

```
<service name="FlatStructureIn">
<interface.wsdl
interface="http://xmlns.oracle.com/
       pcbpel/adapter/file/
       FlatStructureIn/#wsdl.inte
rface(Read_ptt)"/>
<binding.jca config="FlatStructureIn_file.jca">
<property name="ignoreListingErrors"
                    type="xs:string"
many="false">true</property>
</binding.jca>
</service>
```

Figure 4-14 The Adapter Configuration Wizard - File Directories Page



Append Mode

The Oracle File and FTP Adapters enable you to configure outbound interactions that append to an existing file. The Append to Existing File option enables the outbound invoke to write to the same file. There are two ways in which you can append to a file name:

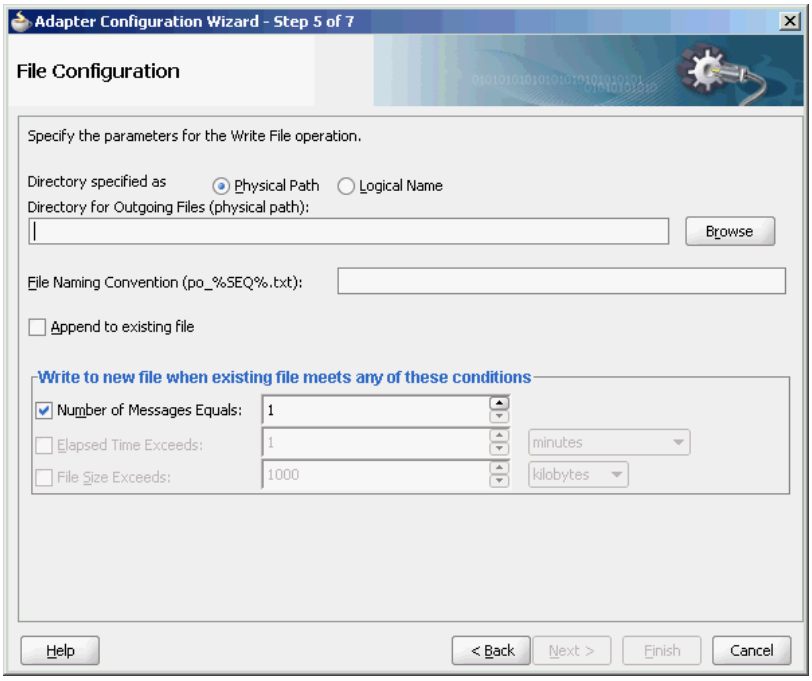
- Statically — in the JCA file for the outbound Oracle File Adapter.
- Dynamically — using the header mechanism.

Note:

The append mode is not supported for SFTP scenarios, where instead of appending to the existing file, the file is overwritten.

When you select the **Append to existing file** option in the File Configuration page, the batching options such as Number of Messages Equals, Elapsed Time Exceeds, File Size Exceeds options are disabled. Figure 4-15 displays the **Append to existing file** option.

Figure 4-15 The Adapter Configuration Wizard - File Configuration Page



Batching option is disabled if "Append" is chosen in the wizard. In addition, the following error message is displayed if the user specifies a dynamic file naming convention as opposed to a static file naming convention:

```
You cannot choose to Append Files and use a dynamic file naming convention at the
same time
```

If you are using the "Append" functionality in Oracle FTP Adapter, ensure that your FTP server supports the "APPE" command.

Recursive Processing of Files Within Directories in Oracle FTP Adapter

In earlier versions of the Oracle SOA Suite, the inbound Oracle FTP Adapter used the `NLST` (Name List) FTP command to read a list of file names from the FTP server. However, the NLST command does not return directory names and therefore does not allow recursive processing within directories. Currently, the Oracle FTP Adapter uses the `LIST` command, instead.

However, the response from the `LIST` command is different for different FTP servers. To incorporate the subtle differences in results from the `LIST` command in a standard manner, the following parameters are added to the deployment descriptor for Oracle FTP Adapter:

- `defaultDateFormat`: This parameter specifies the default date format value. On the FTP server, this is the value for files that are older. The default value for this parameter is `MMM d yyyy` as most UNIX-type FTP servers return the last modified time stamp for older files in the `MMM d yyyy` format. For example, `Jan 31 2006`.

  You can find the default date format for your FTP server by using the `ls -l` command by using a FTP command-line client. For example, `ls -l` on a vsftpd server running on Linux returns the following:

  ```
  -rw-r--r--    1 500      500           377 Jan 22 2005 test.txt
  ```

For Microsoft Windows NT FTP servers, the `defaultDateFormat` is `MM-dd-yy hh:mma`, for example, `03-24-09 08:06AM <DIR> oracle`.

- `recentDateFormat`: This parameter specifies the recent date format value. On the FTP server, this is the value for files that were recently created.

  The default value for this parameter is `MMM d HH:mm` as most UNIX-type FTP servers return the last modified date for recently created files in `MMM d HH:mm` format, for example, `Jan 31 21:32`.

  You can find the default date format for your FTP server by using the `ls -l` command from an FTP command-line client. For example, `ls -l` on a vsftpd server running on Linux returns the following:

```
150 Here comes the directory listing.
-rw-r--r--    1 500      500           377 Jan 30 21:32 address.txt
-rw-r--r--    1 500      500           580 Jan 3121:32 container.txt
.....................................................................
...
```

  For Microsoft Windows NT FTP servers, the `recentDateFormat` parameter is in the `MM-dd-yy hh:mma`, format, for example, `03-24-09 08:06AM <DIR> oracle`.

- `serverTimeZone`: The server time zone, for example, *America/Los_Angeles*. If this parameter is set to blank, then the default time zone of the server running the Oracle FTP Adapter is used.

- `listParserKey:` Directs the Oracle FTP Adapter on how it should parse the response from the `LIST` command. The default value is UNIX, in which case the Oracle FTP Adapter uses a generic parser for UNIX-like FTP servers. Apart from `UNIX`, the other supported values are `WIN` and `WINDOWS`, which are specific to the Microsoft Windows NT FTP server.

  Note:

  The locale language for the FTP server can be different from the locale language for the operating system. Do not assume that the locale for the FTP server is the same locale for the operating system it is running on. You must set the `serverLocaleLanguage`, `serverLocaleCountry`, and `serverLocaleVariant` parameters in such cases.

- `serverLocaleLanguage`: This parameter specifies the locale construct for language.

- `serverLocaleCountry`: This parameter specifies the locale construct for country.

- `serverLocaleVariant`: This parameter specifies the locale construct for variant.

Configure the Parameters in the Deployment Descriptor

The standard date formats of an FTP server are usually configured when the FTP server is installed. If your FTP server uses a format "MMM d yyyy" for defaultDateFormat and "MMM d HH:mm" for recentDateFormat, then your Oracle FTP Adapter must use the same formats in its corresponding deployment descriptor.

If you enter "ls -l" from a command-line FTP, then you can see the following:

```
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r--    1 500      500           377 Jan 22 21:32 1.txt
-rw-r--r--    1 500      500           580 Jan 22 21:32 2.txt
.....................................................................
```

This is the `recentDateFormat` parameter for your FTP server, for example MMM d HH:mm (Jan 22 21:32). Similarly, if your server has an old file, the server does not show the hour and minute part and it shows the following:

```
-rw-r--r--    1 500      500           377 Jan 22 2005 test.txt
```

This is the default date format, for example MMM d yyyy (Jan 22 2005).

Additionally, the `serverTimeZone` parameter is used to by the Oracle FTP Adapter to parse time stamps for FTP server running in a specific time zone. The value for this is either an abbreviation such as "PST" or a full name such as "America/Los_Angeles".

Additionally, the FTP server might be running on a different locale. The `serverLocaleLanguage`, `serverLocaleCountry`, and `serverLocaleVariant` parameters are used to construct a locale from language, country, variant where

- language is a lowercase two-letter ISO-639 code, for example, en,
- country is an uppercase two-letter ISO-3166 code, for example, US.
- variant is a vendor and browser-specific code.

If these locale parameters are absent, then the Oracle FTP Adapter uses the system locale to parse the time stamp.

Additionally, if the FTP server is running on a different system than the SOA suite, then you must handle the time zone differences between them. You must convert the time difference between the FTP server and the system running the Oracle FTP Adapter to milliseconds and add the value as a binding property: `timestampOffset` in the `composite.xml`.

For example, if the FTP server is six hours ahead of your local time, you must add the following endpoint property to your service or reference. See the example below.

Example - Endpoint Property to Add if FTP Server is Ahead of Local

```
<service name="FTPDebatchingIn">
    <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel
             /adapter/ftp/FTPDebatchingIn/#wsdl.
interface(Get_ptt)"/>
    <binding.jca config="DebatchingIn_ftp.jca">
<property name=" timestampOffset"
               type="xs:string"
                        many="false" source=""
override="may"> 21600000</property>
    </binding.jca>
  </service>
```

Some FTP servers do not work well with the `LIST` command. In such cases, use the `NLST` command for listing, but you cannot process directories recursively with `NLST`.

To use the `NLST` command, then you must add the following property to the JCA file. See the example below.

Example - Adding the NLST Property

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FTPDebatchingIn"
                   adapter="Ftp Adapter"
 xmlns="http://platform.integration.oracle/
                  blocks/adapter/fw/metadata">
<connection-factory location="eis/Ftp/FtpAdapter"
                     UIincludeWildcard="*.txt"
 adapterRef=""/>
    <activation-spec
className="oracle.tip.adapter.ftp.
                  inbound.FTPActivationSpec">
       ……..
       ……..
      <property name="UseNlst" value="true"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

Securing Enterprise Information System Credentials

When a resource adapter makes an outbound connection with an Enterprise Information System (EIS), it must sign on with valid security credentials. In accordance with the J2CA 1.5 specification, Oracle WebLogic Server supports both container-managed and application-managed sign-on for outbound connections. At runtime, Oracle WebLogic Server determines the chosen sign-on mechanism, based on the information specified in either the invoking client component's deployment descriptor or the `res-auth` element of the resource adapter deployment descriptor. This section describes the procedure for securing the user name and password for Oracle JCA Adapters by using Oracle WebLogic Server container-managed sign-on.

Both Oracle WebLogic Server and EIS maintain independent security realms. A container-managed sign-on enables you to sign on to Oracle WebLogic Server and also be able to use applications that access EIS through a resource adapter without having to sign on separately to the EIS. Container-managed sign-on in Oracle WebLogic Server uses credential mappings. The credentials (user name/password pairs or security tokens) of Oracle WebLogic security principals (authenticated individual users or client applications) are mapped to the corresponding credentials required to access EIS. You can configure credential mappings for applicable security principals for any deployed resource adapter.

To use container-managed sign-on first you must ensure that the connection pool you use supports container-managed sign-on. You can follow these steps to turn on container-managed sign-on for an existing connection pool or create a new pool which supports container-managed sign-on.

1. Go to the Oracle SOA installation directory where the SOA adapter .rar files are located (for example, `</fmwhome/AS11gR1SOA>/soa/connectors` or `<C:\Oracle\Middleware\Oracle_SOA1>\soa\connectors)` where you must replace the path in the brackets with an appropriate path.

2. Make a backup copy of the adapter .rar file, for example, `FtpAdapter.rar`, for which the `weblogic-ra.xml` will be modified.

3. Extract the `weblogic-ra.xml` from the .rar file and open the extracted `weblogic-ra.xml` with an editor.

4. For an existing connection pool in weblogic-ra.xml, if `<wls:res-auth>Container</wls:res-auth>` does not exist under the <wls:connection-properties> for the connection pool, add the property to the connection pool.

As an example, the pre-configured eis/Ftp/FtpAdapter connection pool in the FTP adapter is defined as in the example below.

Example - Defining the Pre-Configured Connection Pool

```
<wls:connection-instance>
<wls:description>Ftp Adapter</wls:description>
<wls:jndi-name>eis/Ftp/FtpAdapter</wls:jndi-name>
<wls:connection-properties>
 <wls:res-auth>Container</wls:res-auth>
</wls:connection-properties>
```

This example indicates that this pool supports container-managed sign-on.

5. To create a new connection pool that supports container-managed sign-on, add a new connection pool instance by using an existing instance as an example. You must replace the JNDI name and/or the connection factory name with appropriate names. The following is a sample configuration of a JMS connection pool that supports container-managed sign-on.

   (The presence of `java.naming.factory.initial` and `java.naming.provider.url`are required to turn on container-managed sign-on.

   Example - Sample Configuration of a JMS Connection Pool Supporting Container-Managed Sign-on

```
 <connection-instance>
  <jndi-name>eis/jms/MyConnectionPool</jndi-name>
    <connection-properties>
      <properties>
        <property>
         <name>ConnectionFactoryLocation</name>
           <value>jms/MyConnectionFactory</value>
        </property>
        <property>
          <name>FactoryProperties</name>
          <value>java.naming.factory.initial
              =weblogic.jndi.
                     WLInitialContextFactory;java.
               naming.provider.url=
                 t3://ncosmidi-us5.us.
                      mydomain.com:8001</value>
         </property>
         <property>
           <name>AcknowledgeMode</name>
            <value>AUTO_ACKNOWLEDGE</value>
         </property>
         <property>
           <name>IsTopic</name>
           <value>true</value>
         </property>
         <property>
          <name>IsTransacted</name>
            <value>false</value>
         </property>
         <property>
          <name>Username</name>
          <value></value>
         </property>
         <property>
```

```
            <name>Password</name>
            <value></value>
          </property>
            </properties>
          <res-auth>Container</res-auth>
    </connection-properties>
 </connection-instance>
```

6. Add the modified `weblogic-ra.xml` back to `adapter .rar` and make sure it replaces the existing `weblogic-ra.xml` in the `.rar` file.

7. Proceed to the **Weblogic Server Console > Deployments** and select the adapter, for example, FTP Adapter or JMS Adapter. Click **Update**.

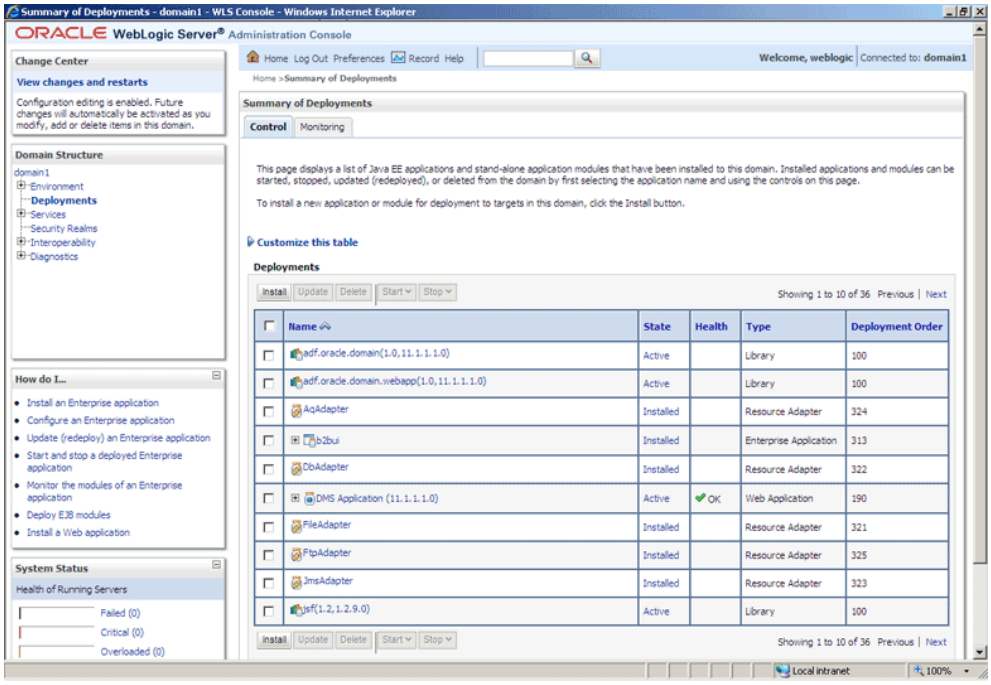
8. For a new connection pool, for example, a JMS pool, open D**eployments > JmsAdapter > Configuration > Outbound Connection Pools** and expand `oracle.tip.adapter.jms.IJmsConnectionFactory` to verify the new connection pool is present.

   After these steps, you can configure credential mapping for the corresponding pool by performing the following procedure using the Oracle WebLogic Server console. To do so:

9. Log in to the Oracle WebLogic Server Administration Console.

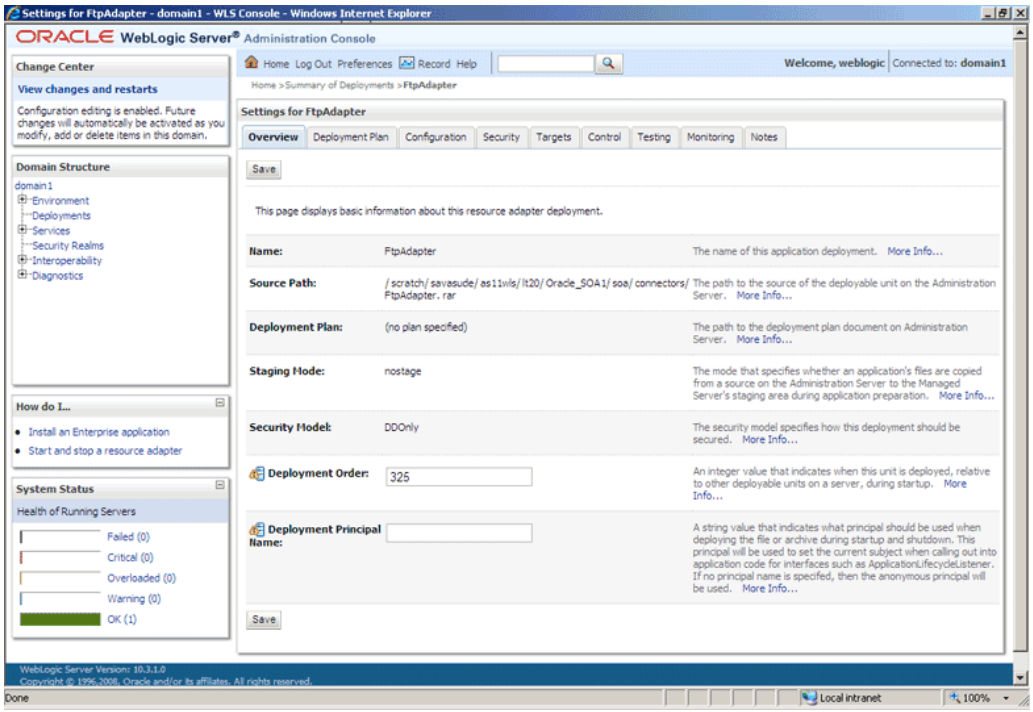
   Click **Deployments** in the Domain Structure pane. The deployed applications and adapters are displayed, as shown in Figure 4-16.

Figure 4-16 The Oracle WebLogic Server Administration Console - Summary of Deployments Page



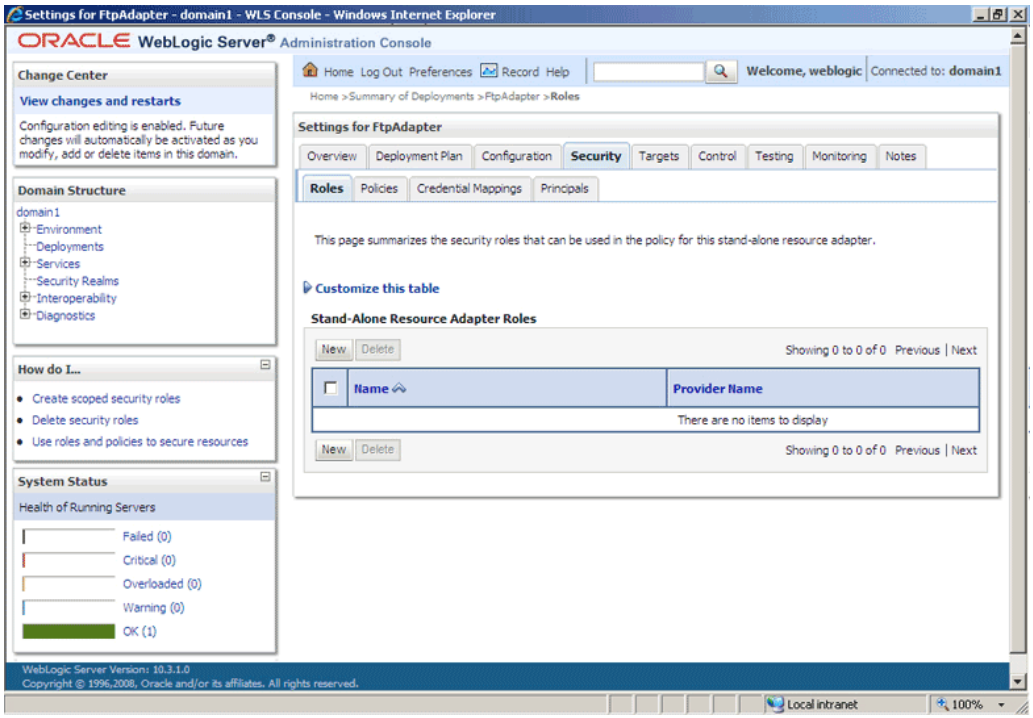
10. Click the adapter for which you must create the security credentials. For example, click `FtpAdapter`. The Settings for FtpAdapter page is displayed, as shown in Figure 4-17.

Figure 4-17 The Oracle WebLogic Server Administration Console - Settings for FTPAdapter Page



11. Click the **Security** tab. The Settings for FTP Adapter page with the Stand-Alone Resource Adapter Roles pane displayed, as shown in Figure 4-18

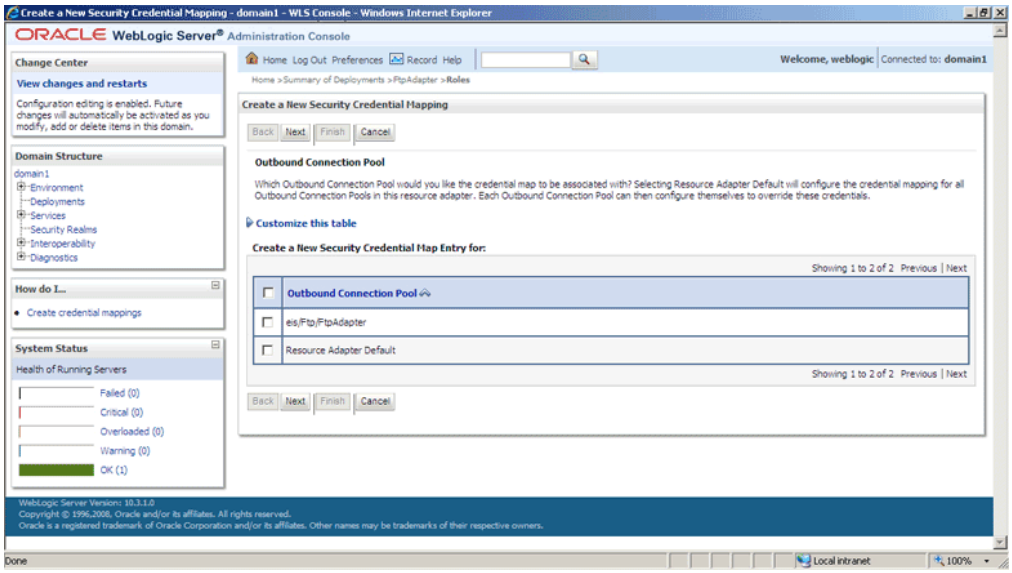
Figure 4-18 The Oracle WebLogic Server Administration Console - Settings for FTPAdapter Page



12. Click the **Credential Mappings** tab.

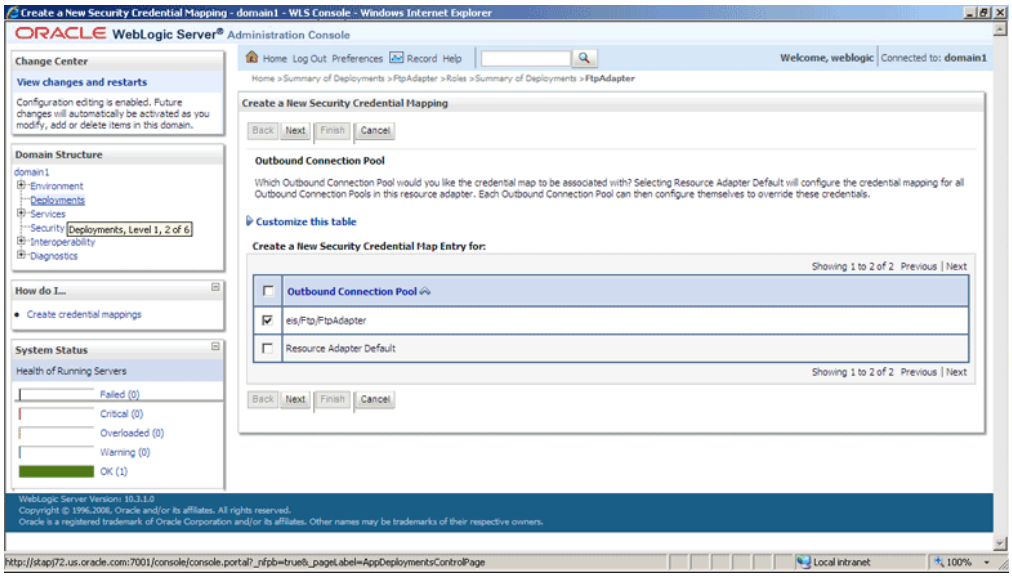
13. Click **New** in the **Credential Mappings** pane. The **Create a New Security Credential Mapping** page is displayed, as shown in Figure 4-19.

Figure 4-19 The Oracle WebLogic Server Administration Console - Create a New Security Credential Mapping Page



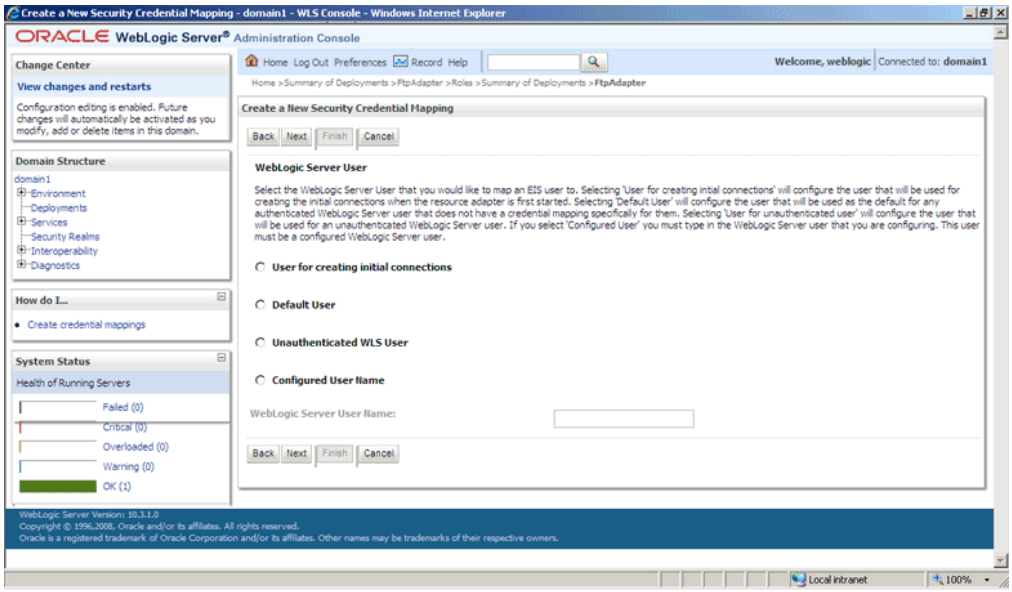
14. Select **eis/Ftp/FtpAdapter** (JNDI for Oracle FTP Adapter) to create a security credential map entry for Oracle FTP Adapter, as shown in Figure 4-20.

Figure 4-20 The Oracle WebLogic Server Administration Console - Create a New Security Credential Mapping Page



15. Click **Next**. The Create a New Security Credential Mapping – WebLogic Server User page is displayed, as shown in Figure 4-21.

Figure 4-21 The Oracle WebLogic Server Administration Console - Create a New Security Credential Mapping Page

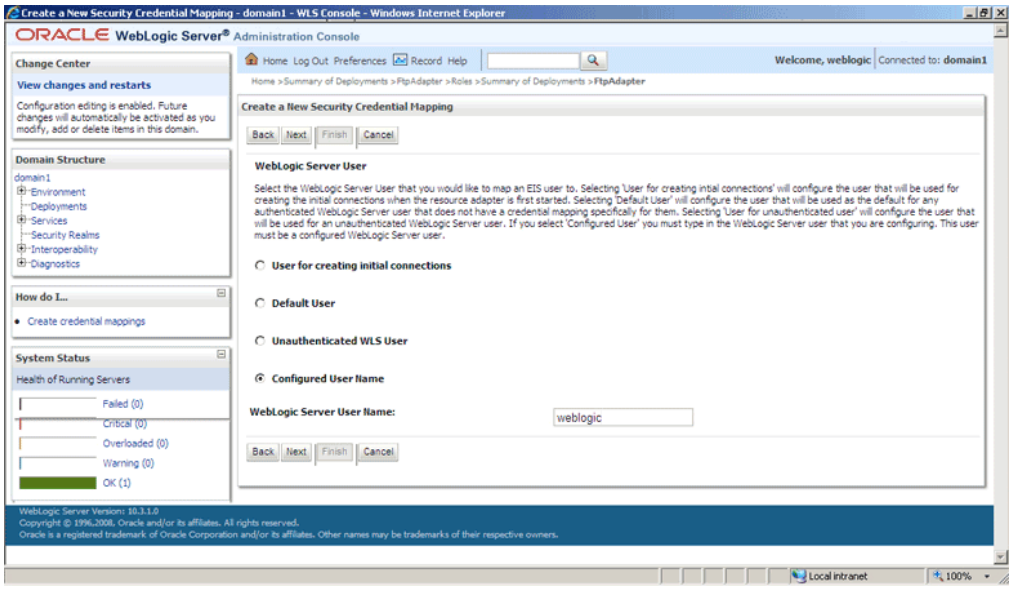


Note:

Credential mapping is not supported for the **User for creating initial connections** and **Unauthenticated WLS User** options.

16. Select **Configured User Name** and enter the Oracle WebLogic Server user name in the **WebLogic Server User Name** field, as shown in Figure 4-22. For example, enter `weblogic`, which is the default user name. Depending on your configuration, in some cases, you might select **Default User** if **Configured User Name** does not work.

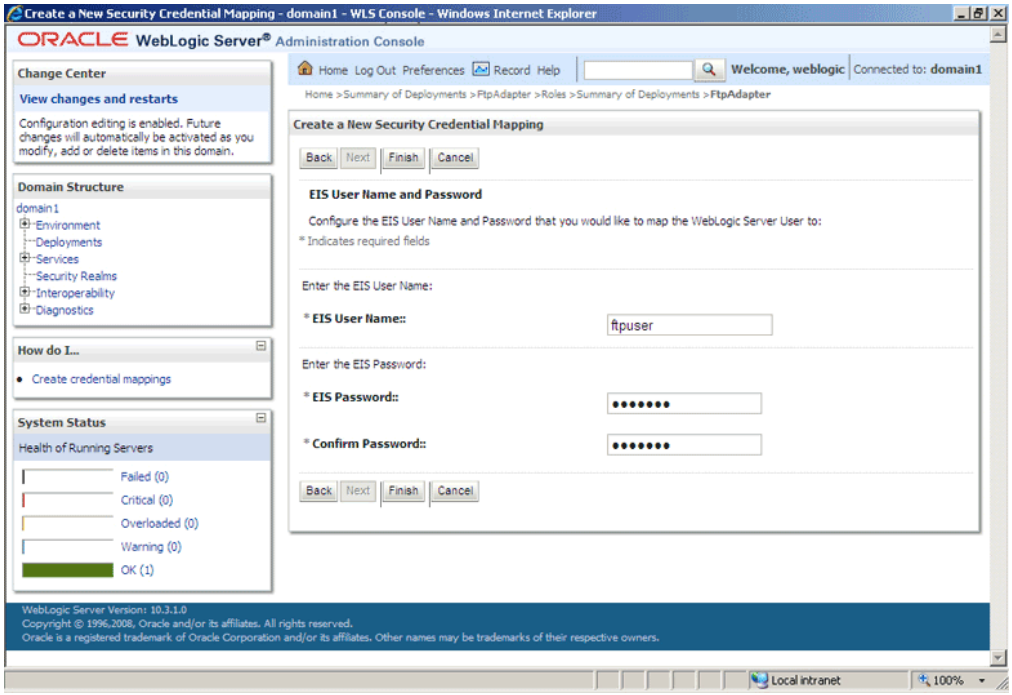
Figure 4-22 The Oracle WebLogic Server Administration Console - Create a New Security Credential Mapping Page



17. Click **Next**. The **Create a New Security Credential Mapping** – EIS User Name and Password page is displayed.

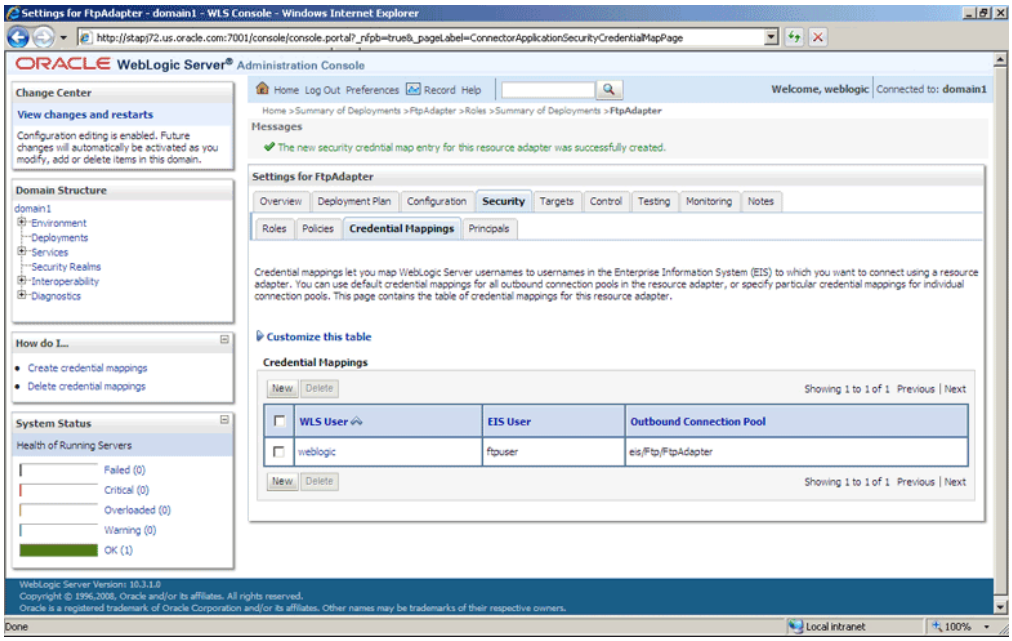
18. Enter the EIS user name in the **EIS User Name** field, the EIS password in the **EIS Password** field, and then reenter the EIS password in the **Confirm Password** field to confirm the password, as shown in Figure 4-23.

Figure 4-23 The Oracle WebLogic Server Administration Console - Create a New Security Credential Mapping Page



19. Click **Finish**. The new security credential mapping is created, as shown in Figure 4-24.

Figure 4-24 The Oracle WebLogic Server Administration Console - Settings for FTPAdapter Page



Oracle File and FTP Adapter Concepts

The Oracle File and FTP Adapters concepts are discussed in the following sections:

- Read File Concepts
- Write File Concepts
- Synchronous Read Concepts
- File Listing Concepts
- Get File Concepts
- Put File Concepts
- Synchronous Get File Concepts
- File Listing Concepts
- File and FTP Adapter Extensions

Oracle File Adapter Read File Concepts

In the inbound direction, the Oracle File Adapter polls and reads files from a file system for processing. This section provides an overview of the inbound file reading capabilities of the Oracle File Adapter. You use the Adapter Configuration Wizard to configure the Oracle File Adapter for use with a BPEL process or a Mediator. Configuring the Oracle File Adapter creates an inbound `WSDL` and `JCA` file pair.

The following sections describe the Oracle File Adapter read file concepts:

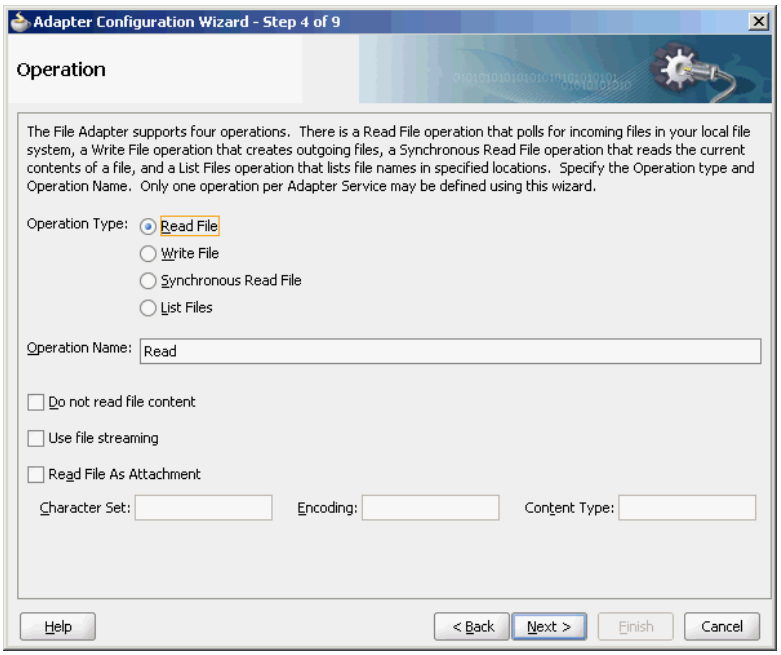
- Inbound Operation
- Inbound File Directory Specifications

- File Matching and Batch Processing
- File Polling
- Postprocessing
- Native Data Translation
- Inbound Service
- Inbound Headers

Inbound Operation

For inbound operations with the Oracle File Adapter, select the **Read File** operation, as shown in Figure 4-25.

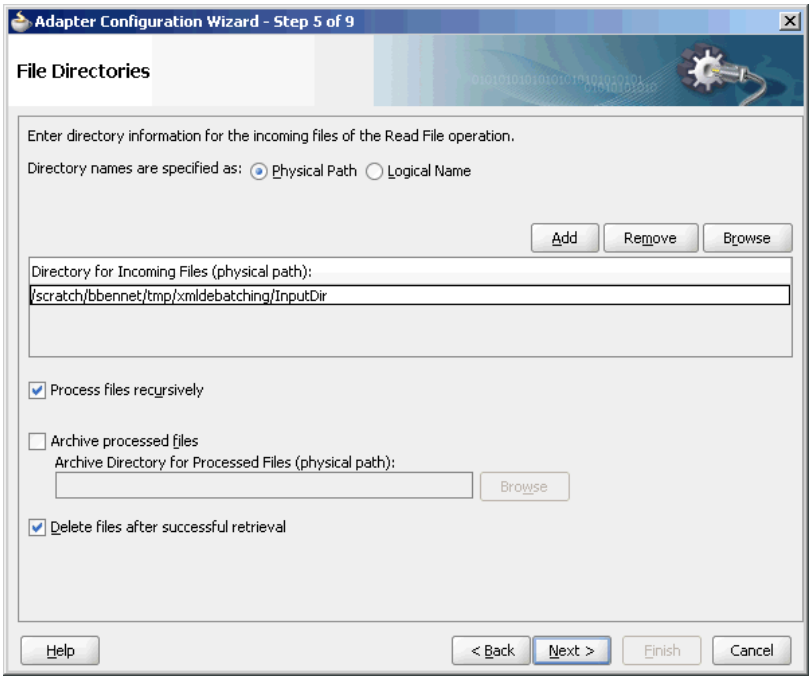
Figure 4-25 Selecting the Read File Operation



Inbound File Directory Specifications

The **File Directories** page of the Adapter Configuration Wizard shown in Figure 4-26 enables you to specify information about the directory to use for reading inbound files and the directories in which to place successfully processed files. You can choose to process files recursively within directories. You can also specify multiple directories.

Figure 4-26 The Adapter Configuration Wizard - Specifying Incoming Files



The following sections describe the file directory information to specify:

- Specifying Inbound Physical or Logical Directory Paths in SOA Composite
- Archiving Successfully Processed Files
- Deleting Files After Retrieval

Specifying Inbound Physical or Logical Directory Paths in SOA Composite

You can specify inbound directory names as physical or logical paths in the composite involving Oracle BPEL PM and Mediator. Physical paths are values such as `c:\inputDir`.

Note:

If the inbound Oracle File Adapter is configured for polling multiple directories for incoming files, then all the top-level directories (inbound directories where the input file appears) must exist before the file reader starts polling these directories.

In the composite, logical properties are specified in the inbound `JCA` file and their logical-physical mapping is resolved by using binding properties. You specify the logical parameters once at design time, and you can later modify the physical directory name as required.

For example, the generated inbound `JCA` file looks as follows for the logical input directory name `InputFileDir`.

Example - Generated Inbound .jca File

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FlatStructureIn"
```

```
                adapter="File Adapter"
xmlns="http://platform.integration.oracle/
            blocks/adapter/fw/metadata">

  <connection-factory location="eis/FileAdapter"
           UIincludeWildcard="*.txt"
adapterRef=""/>
  <endpoint-activation operation="Read">
    <activation-spec
className="oracle.tip.adapter.file.
                    inbound.FileActivationSpec">
      <property name="UseHeaders" value="false"/>
      <property name="LogicalDirectory"
                 value="InputFileDir"/>
      <property name="Recursive" value="true"/>
      <property name="DeleteFile" value="true"/>
      <property name="IncludeFiles" value=".*\.txt"/>
      <property name="PollingFrequency" value="10"/>
      <property name="MinimumAge" value="0"/>
      <property name="OpaqueSchema" value="false"/>
    </activation-spec>
  </endpoint-activation>

</adapter-config>
```

In the `composite.xml` file, you then provide the physical parameter values (in this case, the directory path) of the corresponding logical `ActivationSpec` or `InteractionSpec`. This resolves the mapping between the logical directory name and actual physical directory name. See the example below.

Example - Providing the Directory Path of the Corresponding ActivationSpec or InteractionSpec

```
<service name="FlatStructureIn">
    <interface.wsdl
     interface="http://xmlns.oracle.com/pcbpel/
               adapter/file/FlatStructureIn/#wsdl.
     interface(Read_ptt)"/>
    <binding.jca config="FlatStructureIn_file.jca">
<property name=" InputFileDir" type="xs:string"
                        many="false" source=""
      override="may"> /home/user/inputDir</property>

 </binding.jca>
</service>
```

Archiving Successfully Processed Files

This option enables you to specify a directory in which to place successfully processed files. You can also specify the archive directory as a logical name. In this case, you must follow the logical-to-physical mappings described in Specifying Inbound Physical or Logical Directory Paths in SOA Composite.

Deleting Files After Retrieval

This option enables you to specify whether to delete files after a successful retrieval. If this check box is not selected, processed files remain in the inbound directory but are ignored. Only files with modification dates more recent than the last processed file are retrieved. If you place another file in the inbound directory with the same name as a

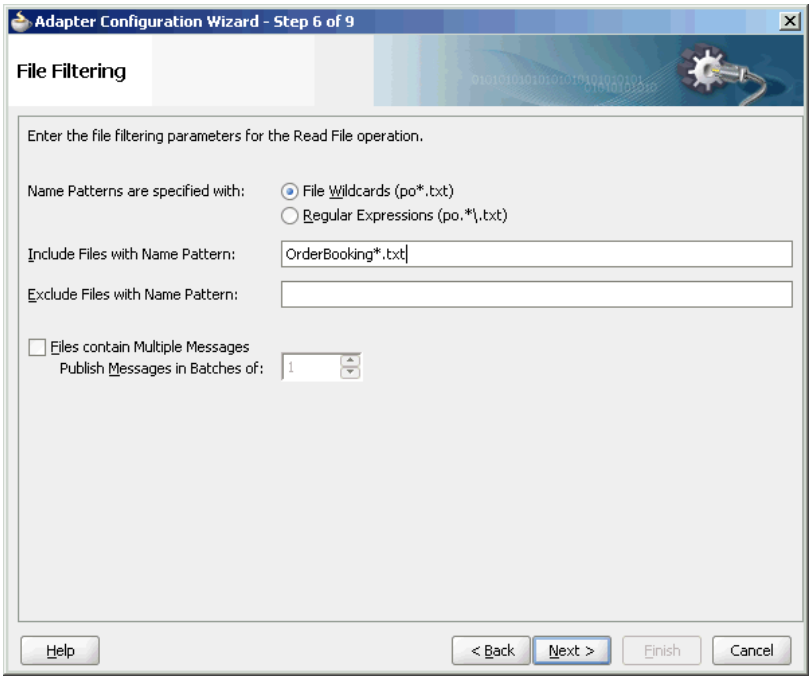
file that has been processed but the modification date remains the same, then that file is not retrieved.

File Matching and Batch Processing

The File Filtering page of the Adapter Configuration Wizard shown in Figure 4-27 enables you to specify details about the files to retrieve or ignore.

The Oracle File Adapter acts as a file listener in the inbound direction. The Oracle File Adapter polls the specified directory on a local or remote file system and looks for files that match specified naming criteria.

Figure 4-27 The Adapter Configuration Wizard-File Filtering Page



The following sections describe the file filtering information to specify:

- Specifying a Naming Pattern
- Including and Excluding Files
- Debatching Messages

Specifying a Naming Pattern

Specify the naming convention that the Oracle File Adapter uses to poll for inbound files. You can also specify the naming convention for files you do not want to process. Two naming conventions are available for selection. The Oracle File Adapter matches the files that appear in the inbound directory.

- File wildcards (`po*.txt`)

  Retrieves all files that start with `po` and end with `.txt`. This convention conforms to Windows operating system standards.

- Regular expressions (`po.*\.txt`)

Retrieves all files that start with `po` and end with `.txt`. This convention conforms to Java Development Kit (JDK) regular expression (regex) constructs.

Note:

- If you later select a different naming pattern, ensure that you also change the naming conventions you specify in the Include Files and Exclude Files fields. The Adapter Configuration Wizard does not automatically make this change for you.
- Do *not* specify ***.*** as the convention for retrieving files.
- Be aware of any file length restrictions imposed by your operating system. For example, Windows operating system file names cannot be more than 256 characters in length (the file name, plus the complete directory path). Some operating systems also have restrictions on the use of specific characters in file names. For example, Windows operating systems do not allow characters such as `backslash(\)`, slash (`/`), colon (`:`), asterisk (`*`), left angle bracket (`<`), right angle bracket (`>`), or vertical bar(`|`).

Including and Excluding Files

If you use regular expressions, the values you specify in the Include Files and Exclude Files fields must conform to JDK regular expression (regex) constructs. For both fields, different regex patterns must be provided separately. The Include Files and Exclude Files fields correspond to the `IncludeFiles` and `ExcludeFiles` parameters, respectively, of the inbound `WSDL` file.

Note:

The regex pattern complies with the JDK regex pattern. According to the JDK regex pattern, the correct connotation for a pattern of any characters with any number of occurrences is a period followed by a plus sign (`.+`). An asterisk (*) in a JDK regex is not a placeholder for a string of any characters with any number of occurrences.

For the inbound Oracle File Adapter to pick up all file names that start with `po` and which have the extension `txt`, you must specify the Include Files field as `po.*\.txt` when the name pattern is a regular expression. In this regex pattern example:

- A period (`.`) indicates any character.
- An asterisk (*) indicates any number of occurrences.
- A backslash followed by a period (\.) indicates the character period (.) as indicated with the backslash escape character.

The Exclude Files field is constructed similarly.

If you have Include Files field and Exclude Files field expressions that have an overlap, then the exclude files expression takes precedence. For example, if Include Files is set to `abc*.txt` and Exclude Files is set to `abcd*.txt`, then no `abcd*.txt` files are received.

Note:

You *must* enter a name pattern in the **Include Files with Name Pattern** field and not leave it empty. Otherwise, the inbound adapter service reads all the files present in the inbound directory, resulting in incorrect results.

Table 4-4 lists details of Java regex constructs.

Note:

Do not begin JDK regex pattern names with the following characters: plus sign (+), question mark (?), or asterisk (*).

Table 4-4 Java Regular Expression Constructs

Matches	Construct
Characters	-
The character `x`	`x`
The backslash character	`\\`
The character with octal value `0n (0 <= n <= 7)`	`\0n`
The character with octal value `0nn (0 <= n <= 7)`	`\0nn`
The character with octal value `0mnn (0 <= m <= 3, 0 <= n <= 7)`	`\0mnn`
The character with hexadecimal value `0xhh`	`\xhh`
The character with hexadecimal value `0xhhhh`	`\uhhhh`
The tab character `('\u0009')`	`\t`
The new line (line feed) character `('\u000A')`	`\n`
The carriage-return character `('\u000D')`	`\r`
The form-feed character `('\u000C')`	`\f`
The alert (bell) character `('\u0007')`	`\a`
The escape character `('\u001B')`	`\e`
The control character corresponding to `x`	`\cx`
-	-
Character classes	-
`a`, `b`, or `c` (simple class)	`[abc]`
Any character except `a`, `b`, or `c` (negation)	`[^abc]`
`a` through `z` or `A` through `Z`, inclusive (range)	`[a-zA-Z]`

Table 4-4 (Cont.) Java Regular Expression Constructs

Matches	Construct
a through d, or m through p: [a-dm-p] (union)	[a-d[m-p]]
d, e, or f (intersection)	[a-z&&[def]]
a through z, except for b and c: [ad-z] (subtraction)	[a-z&&[^bc]]
a through z, and not m through p: [a-lq-z](subtraction)	[a-z&&[^m-p]]
-	-
Predefined character classes	-
Any character (may or may not match line terminators)	-
A digit: [0-9]	\d
A nondigit: [^0-9]	\D
A white space character: [\t\n\x0B\f\r]	\s
A nonwhitespace character: [^\s]	\S
A word character: [a-zA-Z_0-9]	\w
A nonword character: [^\w]	\W
Greedy quantifiers	-
X, once or not at all	X?
X, zero or more times	X*
X, one or more times	X+
X, exactly *n* times	X{n}
X, at least *n* times	X{n,}
X, at least *n*, but not more than *m* times	X{n,m}

For details about Java regex constructs, see

http://java.sun.com/j2se/1.5.0/docs/api

File Include and Exclude

The `FileList` operation does not expose the `java.file.IncludeFiles` property. This property is configured while designing the adapter interaction and cannot be overridden through headers, as shown in the example tomorrow.

Example - Overriding the FileList Operation

```
<adapter-config name="ListFiles" adapter="File Adapter"    xmlns="http://
platform.integration.oracle
                  /blocks/adapter/fw/metadata">
   <connection-factory location="eis/FileAdapter"
```

```
                  UIincludeWildcard="*.txt"
                      adapterRef=""/>
    <endpoint-interaction portType="FileListing_ptt"
                              operation="FileListing">
    <interaction-spec className=
                  "oracle.tip.adapter.file.outbound.
                   FileListInteractionSpec">
        <property name="PhysicalDirectory"
                             value="%INP_DIR%"/>
        <property name="PhysicalDirectory"
                           value="%INP_DIR%"/>
        <property name="Recursive" value="true"/>
             <property name="Recursive"
                           value="true"/>
             <property name="IncludeFiles"
                        value=".*\.txt"/>
   </interaction-spec>
   </endpoint-interaction>
</adapter-config>
```

In this example, after you set the `IncludeFiles`, they cannot be changed.

Debatching Messages

You can select whether incoming files have multiple messages, and specify the number of messages in one batch file to publish. When the file contains message with repeating elements, you can choose to publish the message in a specific number of batches. Refer to Figure 4-27.

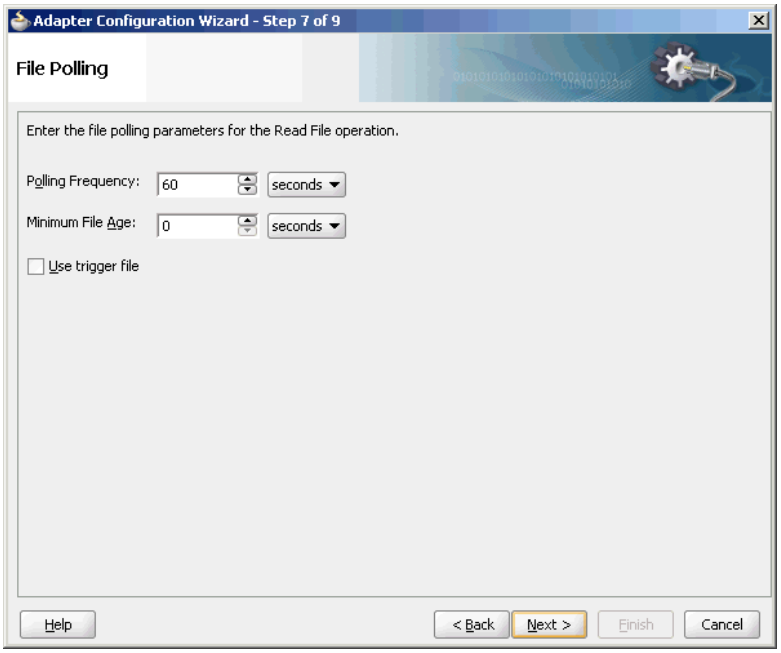
When a file contains multiple messages and this check box is selected, this is referred to as debatching. Nondebatching is applied when the file contains only a single message and the check box is not selected. Debatching is supported for native and XML files.

File Polling

The File Polling page of the Adapter Configuration Wizard shown in Figure 4-28 enables you to specify the following inbound polling parameters:

- The frequency with which to poll the inbound directory for new files to retrieve.
- The minimum file age of files to retrieve. For example, this polling parameter enables a large file to be completely copied into the directory before it is retrieved for processing. The age is determined by the last modified time stamp. For example, if you know that it takes three to four minutes for a file to be written, then set the minimum age to five minutes. If a file is detected in the input directory and its modification time is less than five minutes older than the current time, then the file is not retrieved because it is still potentially being written to.

Figure 4-28 The Adapter Configuration Wizard-File Polling Page



Note:

You *must not* manually change the value of polling parameters in JCA files. You must use the Adapter Configuration Wizard to modify this parameter.

Using Trigger Files

By default, polling by inbound Oracle File and FTP Adapters start as soon as the endpoint is activated. However, to obtain more control over polling, you can use a file-based trigger. Once the Oracle File or FTP Adapter finds the specified trigger file in a local or remote directory, it starts polling for the files in the inbound directory.

For example, a BPEL process is writing files to a directory and a second BPEL process is polling the same directory for files. To have the second process start polling the directory only after the first process has written all the files, you can use a trigger file. You can configure the first process to create a trigger file at the end. The second process starts polling the inbound directory after it finds the trigger file.

Note: The lifecycle of the trigger file is not managed by the adapter. The trigger file must be managed externally. For example, un-trigger the endpoint, delete the trigger file using the external application, and either specify TriggerFileStrategy as EndpointActivation or EveryTime.

The trigger file directory can be the same as the inbound polling directory or different from the inbound polling directory. However, if your trigger file directory and the inbound polling directory are the same, then you should ensure that the name of the trigger file is not similar to the file filter specified in the Adapter Configuration page shown in Figure 4-27.

The content of a trigger file is never read and therefore should not be used as payload for an inbound receive activity.

Table 4-5 lists the parameters that you must specify in the inbound service JCA file:

Table 4-5 Trigger File Parameters

Parameter	Description	Example
`TriggerFilePhysicalDirectory` or `TriggerFileLogicalDirectory`	The physical or logical name of the directory in which the Oracle File and FTP Adapters look for the trigger file. The `TriggerFilePhysicalDirectory` and `TriggerFileLogicalDirectory` parameters are optional. These parameters must be used only if the trigger file directory is different from the inbound polling directory. By default, the Oracle File and FTP Adapters looks for the trigger file in the inbound polling directory.	`TriggerFilePhysicalDirectory="C:\foo"` `TriggerFileLogicalDirectory= "TriggerFileDir"`
`TriggerFile`	The name of the trigger file.	`TriggerFile="Purchaseorder.trg"`
`TriggerFileStrategy`	Strategy that is used as the triggering mechanism. The value can be: *EndpointActivation:* The adapter looks for the trigger file every time the composite is activated. **Note:** The composite gets activated every time you start the container or redeploy the application, or retire or activate the composite application from Fusion Middleware Control. Every time you restart the container, the composite application is not triggered until it sees the trigger file in the specified directory. *OnceOnly:* The adapter looks for the trigger file only once in its lifetime. After it finds the trigger file, it remember that across restarts and redeployments. *EveryTime:* The adapter looks for the trigger file on each polling cycle.The default value for `TriggerFileStrategy` is EndpointActivation.	`TriggerFileStrategy="EndpointActivation "`

The following is a sample JCA file for the inbound service:

Example - Sample .jca File for the Inbound Service

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FlatStructureIn" adapter="File Adapter"
  xmlns="http://platform.integration.oracle/
                 blocks/adapter/fw/metadata">

  <connection-factory
                location="eis/FileAdapter"
    UIincludeWildcard="*.txt"
adapterRef=""/>
```

```
  <endpoint-activation operation="Read">
    <activation-spec className=
                "oracle.tip.adapter.file.
       inbound.FileActivationSpec">
       <property.../>
       <property name=
                "TriggerFilePhysicalDirectory"
             value="/tmp/flat/ArchiveDir"/>
    </activation-spec>
</endpoint-activation>

</adapter-config>
```

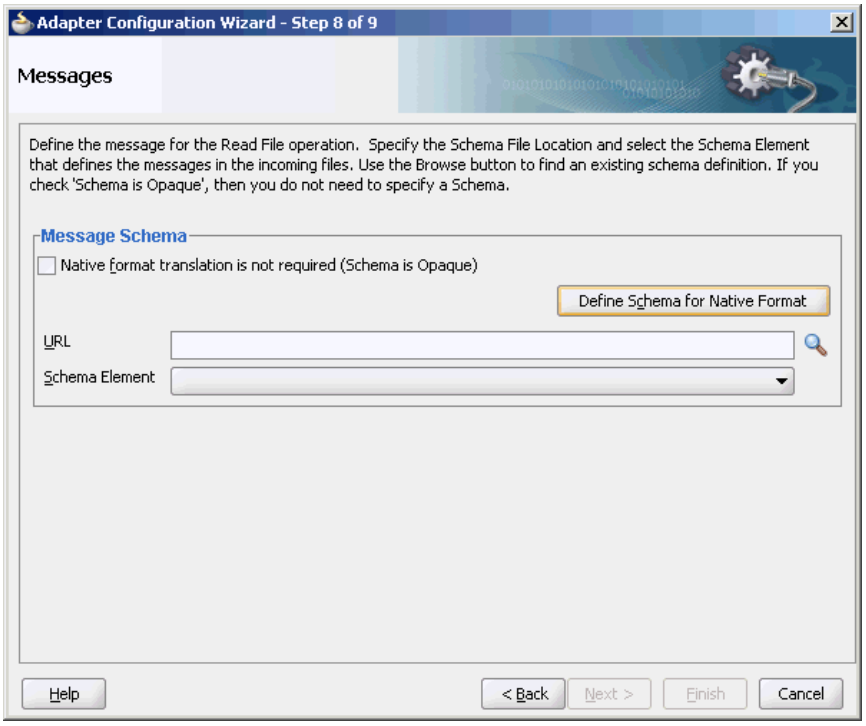
Postprocessing

The Oracle File Adapter supports several postprocessing options. After processing the file, files are deleted if specified in the File Polling page shown in Figure 4-28. Files can also be moved to a completion (archive) directory if specified in the File Directories page shown in Figure 4-26.

Native Data Translation

The next Adapter Configuration Wizard page that appears is the Messages page shown in Figure 4-29. This page enables you to select the XSD schema file for translation.

Figure 4-29 Specifying the Schema - Messages Page



If native format translation is not required (for example, a JPG or GIF image is being processed), then select the **Native format translation is not required** check box. The file is passed through in base-64 encoding.

XSD files are required for translation. To define a new schema or convert an existing data type definition (DTD) or COBOL Copybook, then select **Define Schema for Native Format**. This starts the Native Format Builder wizard. This wizard guides you

through the creation of a native schema file from file formats such as comma-delimited value (CSV), fixed-length, DTD, and COBOL Copybook. After the native schema file is created, the Messages page is displayed, with the Schema File URL and Schema Element fields filled in. For more information, see Supported File Formats.

Note:

Ensure that the schema you specify includes a namespace. If your schema does not have a namespace, then an error message is displayed.

Inbound Service

When you finish configuring the Oracle File Adapter, a `JCA` file is generated for the inbound service. The file is named after the service name you specified on the **Service Name** page of the Adapter Configuration Wizard. You can rerun the wizard later to change your operation definitions.

The `ActivationSpec` parameter holds the inbound configuration information. The `ActivationSpec` and a set of inbound Oracle File Adapter properties are part of the inbound `JCA` file.

Table 4-6 lists the properties of a sample inbound JCA file.

Table 4-6 Sample JCA Properties for Inbound Service

Property	Sample Value
`UseHeaders`	`true`
`PhysicalDirectory`	`/tmp/opaque/in`
`Recursive`	`true`
`DeleteFile`	`false`
`IncludeFiles`	`.*\.xml`
`PollingFrequency`	`1`
`MinimumAge`	`0`

The `ActivationSpec` property values are specified in the Adapter Configuration Wizard during design time and, as shown in Table 4-6. The inbound Oracle File Adapter uses the following configuration properties:

- `PollingFrequency`
- `MinimumAge`
- `PhysicalDirectory`
- `LogicalDirectory`
- `PublishSize`
- `PhysicalArchiveDirectory`
- `LogicalArchiveDirectory`

- `IncludeFiles`
- `ExcludeFiles`
- `UseHeaders`
- `ListSorter`
- `ThreadCount`
- `Recursive`
- `MaxRaiseSize`

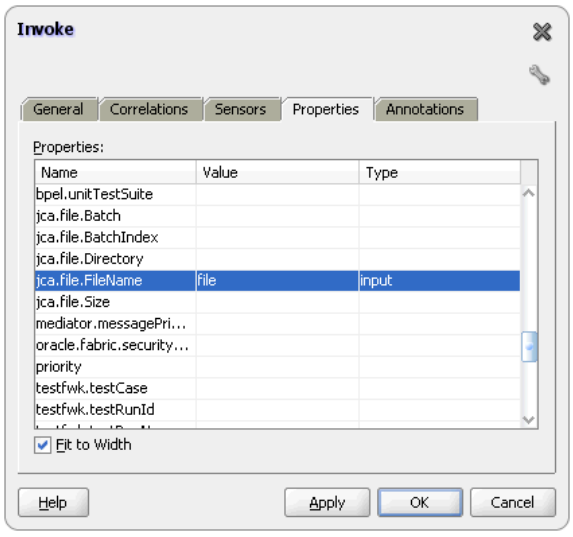
For a description of these configuration properties, see Appendix A of this book.

Inbound Headers

Apart from the payload, Oracle File Adapter publishes the following header metadata, from the inbound service, as shown in Figure 4-30:

- `jca.file.FileName`: file name
- `jca.file.Directory`: directory name
- `jca.file.Batch`: a unique name for a batch in case of debatching
- `jca.file.BatchIndex`: the batch index for each message within the batch for debatching
- `jca.file.Size`: the file size
- `jca.file.LastModifiedTime`: the last modified time for the file

Figure 4-30 The Invoke Dialog



Oracle File Adapter Write File Concepts

In the outbound direction, the Oracle File Adapter receives messages from the service engine and writes the messages to a file in a file system. This section provides an overview of the outbound file writing capabilities of the Oracle File Adapter. You use the Adapter Configuration Wizard to configure the Oracle File Adapter for use with a

BPEL process or a Mediator Service. This creates an outbound WSDL and a JCA file pair.

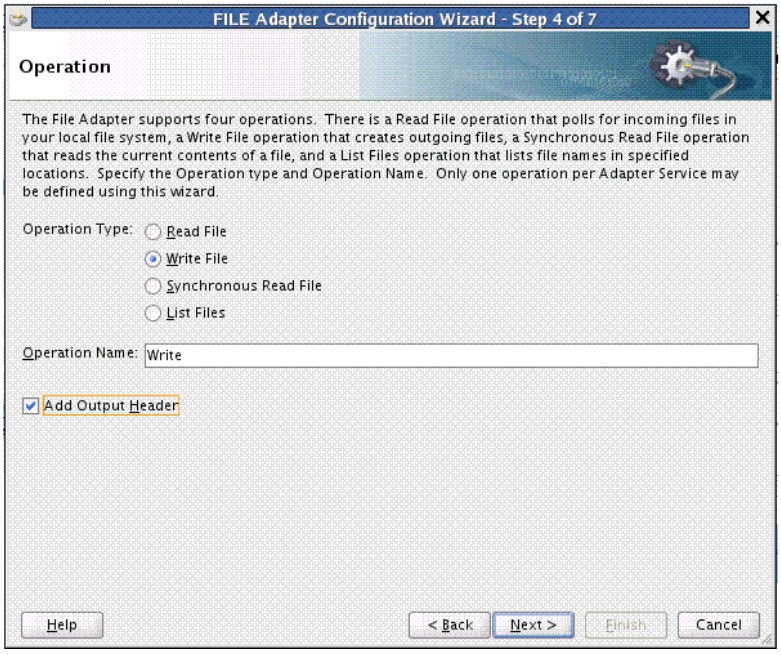
This section includes the following topics:

- Outbound Operation
- Outbound File Directory Creation
- Native Data Translation
- Outbound Service Files
- Outbound Headers

Outbound Operation

For outbound operations with the Oracle File Adapter, select the **Write File** operation, as shown in Figure 4-31.

Figure 4-31 Selecting the Write File Operation



The Add Output Header check box is visible when you select File Write. When you select this check box, the adapter WSDL has an output message pointing to a header schema, shown by the bold highlight.

Example - Adapter WSDL with Output Message Pointing to the Schema

```
 <wsdl:definitions name="fileout3"
  targetNamespace="http://xmlns.oracle.com/pcbpel
           /adapter/file/SOAApp1/NewJCAFmwk/
                   fileout3"
    xmlns:jca="http://xmlns.
                   oracle.com/pcbpel/wsdl/jca/"
xmlns:FILEAPP="http://xmlns.oracle.com
                    /pcbpel/adapter/file/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://xmlns.oracle.com/pcbpel
             /adapter/file/SOAApp1/NewJCAFmwk/
```

```
                fileout3"
xmlns:plt="http://schemas.xmlsoap.org/ws
             /2003/05/partner-link/">"
xmlns:opaque=
    "http://xmlns.oracle.com/
    pcbpel/adapter/opaque/"
<plt:role name="Write_role" >
  <plt:portType name="tns:Write_ptt" />
  </plt:role>
</plt:partnerLinkType>"
<wsdl:types>
<schema TargetNamespace=
         "http://xlmns.oracle.com/pcbpel/
              adapter/opaque/"
   xmlns:opaque="http://xmlns.oracle.com
                       /pcbpel/adapter/opaque/"
      xmlns="http://www.w3.org/2001/XMLSchema" >
   <element name="opaqueElement"
                   type="base64Binary" />
</schema>
<schema targetNamespace=
        "http://xmlns.oracle.com/pcbpel/
          adapter/file/"
xmlns="http://www.w3.org/2001/XMLSchema"
  attributeFormDefault="qualified"
  <element name="OutboundFileHeaderType" >
   <complexType>
    <sequence>
     <element name="filename" type="string" />
     <element name="directory" type="string" />
   </sequence>
   </complexType>
   </element>
</schema>
</wsdl:types>
   <wsdl:message name="Write_msg">
   <wsdl:part name="opaque" element=
         "opaque:opaqueElement"/>
</wsdl:message>
   <wsdl:message name="Output_msg">
   <wsdl:part name="body"
             element=
       "FILEAPP:OutboundFileHeaderType"/>
</wsdl:message>
   <wsdl:portType name="Write_ptt">
   <wsdl:operation name="Write">
     <wsdl:input message="tns:Write_msg"/>
     <wsdl:output message="tns:Output_msg"/>
       </wsdl:operation>
     </wsdl:portType>
</wsdl:definitions>
```

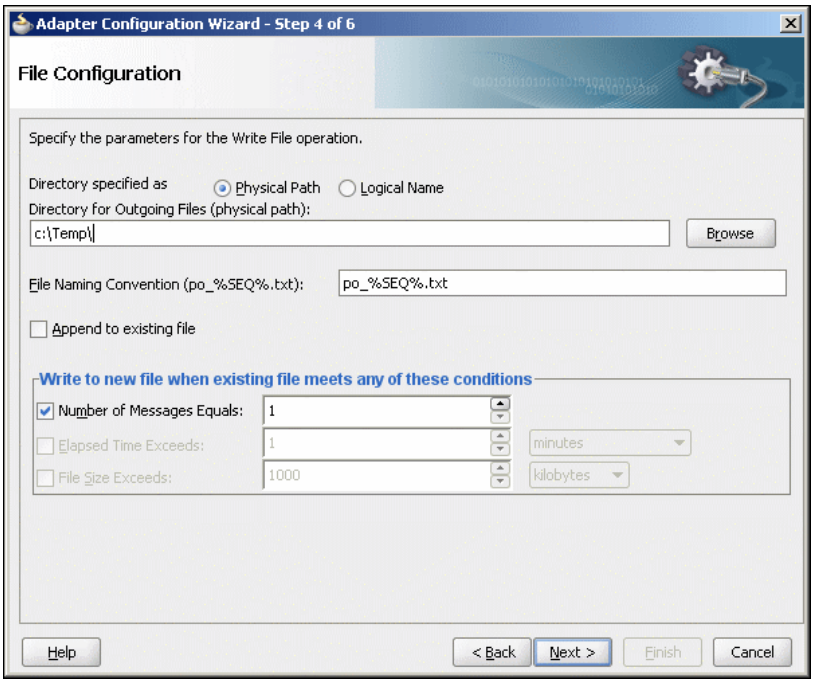
You can select the **Update Output Header** checkbox in edit mode, and the output message/header schema is removed from the adapter WSDL.

Outbound File Directory Creation

For the outbound operation, you can specify the outbound directory, outbound file naming convention to use, and, if necessary, the batch file conventions to use.

The File Configuration page of the Adapter Configuration Wizard shown in Figure 4-32 enables you to specify the directory for outgoing files and the outbound file naming convention.

Figure 4-32 The Adapter Configuration Wizard-Parameters for Outgoing Files



The following sections describe the file configuration information to specify:

- Specifying Outbound Physical or Logical Directory Paths in
- Specifying the Outbound File Naming Convention
- Specifying a Dynamic Outbound File Name
- Batching Multiple Outbound Messages

Specifying Outbound Physical or Logical Directory Paths in Oracle BPEL PM

You can specify outbound directory names as physical or logical paths. Physical paths are values such as `c:\outputDir`.

If you specify logical parameters, then the generated `JCA` file looks as follows for the logical outbound directory name `OutputFileDir`. See the example below.

Example - Generated JCA File for Sample Logical Outbound Directory

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FlatStructureOut"
          adapter="File Adapter"
  xmlns="http://platform.integration.
                oracle/blocks/
    adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter"
                      adapterRef=""/>
  <endpoint-interaction operation="Write">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.
                      FileInteractionSpec">
```

```
      <property name="LogicalDirectory"
                    value="OutputFileDir"/>
      <property name="FileNamingConvention"
             value="%yyMMddHHmmssSS%_%SEQ%_
                      %yyyyMMdd%_%SEQ%.out.%SEQ%"/>
      <property name="Append" value="false"/>
      <property name="NumberMessages" value="1"/>
      <property name="OpaqueSchema" value="false"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

Select the outbound adapter in the **External References** swim lane in JDeveloper wizard (it is present in the composite.xml tab). Create a **Binding Property** in the Property Inspector for the outbound adapter (you must scroll down to find it). Once the **Create Property** box appears, enter `OutputFileDir` in the Name field and the actual output directory name, example, `C:\outputDir` in the Value field. The `composite.xml` file appears. See the example below.

Example - Creating a Property with An Outbound Directory Specified

```
<reference name="FlatStructureOut">
    <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter
           /file/FlatStructureOut/
                 #wsdl.interface(Write_ptt)"/>
    <binding.jca config="FlatStructureOut_file.jca">
      <property name="OutputFileDir"
             type="xs:string" many="false"
                override="may">C:\outputDir
                        </property>
    </binding.jca>
  </reference>
```

Note:

Ensure that you limit the length of outbound file names (the file name, plus the complete directory path) to 200 characters. This is not an exact limit but rather a recommendation. When an outbound file name is long (for example, 215 characters), a blank file with that name is created in the outbound directory.

Specifying Outbound Physical or Logical Directory Paths in Mediator

You can specify outbound directory names as physical or logical paths in Mediator. Physical paths are values such as `c:\inputDir`.

You can specify the logical names at design time in the File Directories page shown in Figure 4-26 and then provide logical-physical mapping by using the Endpoint properties. For example, `WriteFile` is an outbound adapter service. You have specified `OutDir` as the logical directory name during design time.

Specifying a Dynamic Outbound Directory Name

For outbound operation, you can specify a dynamic outbound directory name. You can set variables to specify dynamic outbound directory names.

Example - Setting Variables to Specify Dynamic Outbound Directory Names

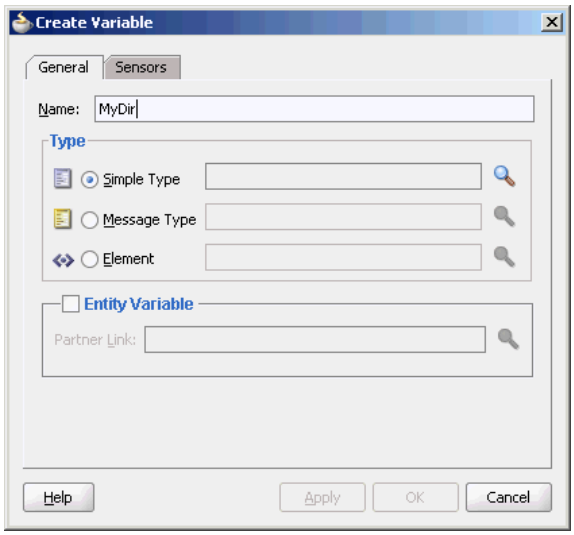
```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="ReadAddressChunk"
                  adapter="File Adapter"
xmlns="http://platform.integration.oracle
                    /blocks/adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter"
                    adapterRef=""/>
  <endpoint-interaction operation="ChunkedRead">
    <interaction-spec
              className=
                 "oracle.tip.adapter.file.outbound.
                   ChunkedInteractionSpec">
      <property name=
              "PhysicalDirectory" value="C:\foo"/>
      <property name="FileName" value="dummy.txt"/>
      <property name="ChunkSize" value="1"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

In the preceding example, in the JCA file, the physical directory is set to `"C:\foo"` but during runtime it is dynamically changed to the assigned value. In this example, the physical directory is dynamically changed to `"C:\out".`

You must perform the following steps to specify the dynamic outbound directory name:

1. Double-click the **Invoke** activity.
2. Select the **Browse Variables** icon.
3. In the **Variable Chooser** dialog, select the **Create an Object** icon.
4. Create a variable `MyDir` of type `xsd:string`, as shown in Figure 4-33.

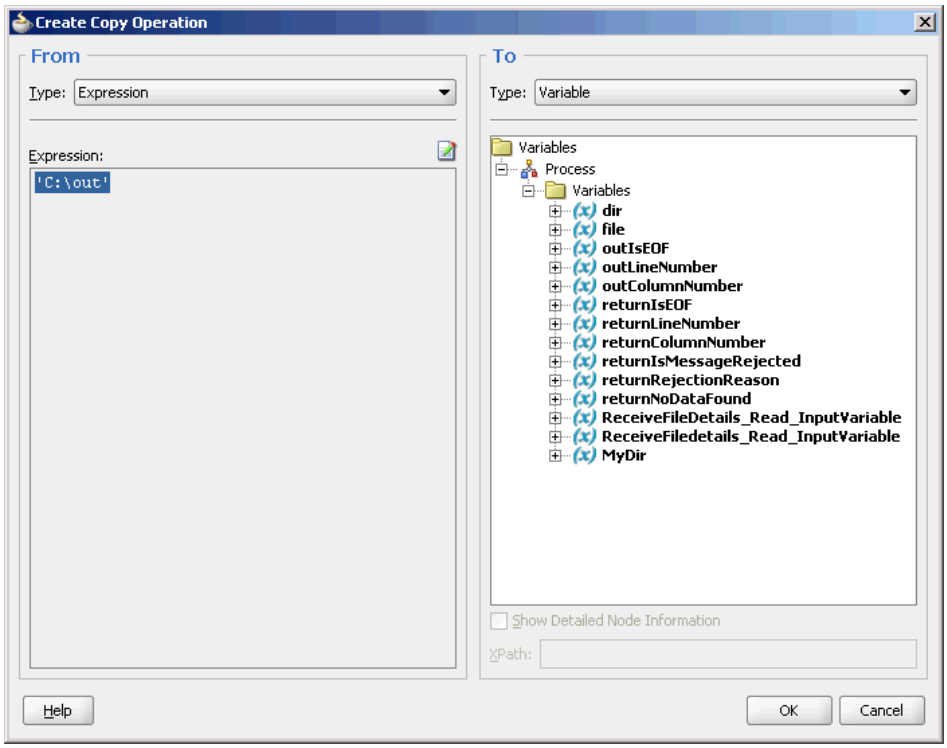
Figure 4-33 Create Variable Dialog



5. Drag and drop an **Assign** activity from the **Components** window in between the **Receive** and **Invoke** activities in the design area.

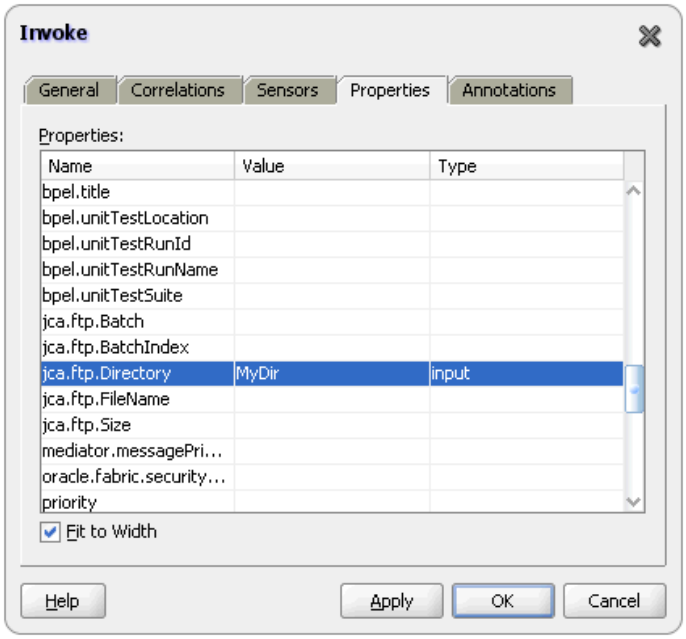
6. Double-click the **Assign** activity and click the **Copy Operation** tab.
7. Click **Create** and then **Copy Operation**. The **Create Copy Operation** dialog is displayed.
8. In the **Create Copy Operation** dialog, select **Expression** from **Type** and specify the directory name and path, as shown in Figure 4-34. The operation writes the output file to this directory.

Figure 4-34 Create Copy Operation Dialog



9. Click **OK** in the Create Copy Operation dialog and then click **OK** in the Assign dialog. The `.bpel` page is displayed.
10. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.
11. Click the **Properties** tab.
12. Select the `jca.file.Directory` property from the **Properties** column and set the **Value** as `MyDir` (the directory that you created in Step 4.) Ensure that the **Type** column is set to `input`, as shown in Figure 4-35.

Figure 4-35 The Invoke Dialog



Note:

When using dynamic directories, ensure that parameters such as `NumberMessages`, `ElapsedTime`, and `FileSize` are not defined in the outbound adapter service `WSDL` file. These parameters are not supported with dynamic directories.

Specifying the Outbound File Naming Convention

Specify the naming convention to use for outgoing files. You cannot enter completely static names such as `po.txt`. This is to ensure the uniqueness in names of outgoing files, which prevents files from being inadvertently overwritten. Instead, outgoing file names must be a combination of static and dynamic portions.

The prefix and suffix portions of the file example shown in Figure 4-32 are static (for example, `po_` and `.xml`). The `%SEQ%` variable of the name is dynamic and can be a sequence number or a time stamp (for example, `po_%yyMMddHHmmss%.xml` to create a file with a time stamp).

If you choose a name starting with `po_`, followed by a sequence number and the extension `txt` as the naming convention of the outgoing files, then you must specify `po_%SEQ%.txt`.

If you choose a name starting with `po_`, followed by a time stamp with the pattern `yyyy.MM.dd` and the extension `txt` as the naming convention of the outgoing file, then you must specify `po_%yyyy.MM.dd%.txt`. For example, the outgoing file name can be `po_2004.11.29.txt`.

Additionally, you can combine file naming conventions. For example, you can specify the file naming convention as `po_%SEQ%_%yyyy.MM.dd%_%SEQ%.txt`.

Note:

When you use the time stamp pattern, the same time stamp may be generated on subsequent calls and you may lose messages. The workaround is to combine the time-stamp pattern with a sequence pattern. Alternatively, you can use a time-stamp pattern closest to a millisecond, in which case the adapter handles the uniqueness of the file names.

You cannot use a regular expression for outbound synchronous reads. In these cases, the exact file name must be known.

A time stamp is specified by date and time pattern strings. Within date and time pattern strings, unquoted letters from `'A'` to `'Z'` and from `'a'` to `'z'` are interpreted as pattern letters representing the components of a date or time string. Text can be quoted using single quotation marks (`'`) to avoid interpretation. The characters `"''"` represent single quotation marks. All other characters are not interpreted.

The Java pattern letters are defined in Table 4-7.

Table 4-7 Java Pattern Letters

Letter	Date or Time Component	Presentation	Examples
`G`	Era designator	Text	`AD`
`y`	Year	Year	`1996`; `96`
`M`	Month in year	Month	`July`; `Jul`; `07`
`w`	Week in year	Number	`27`
`W`	Week in month	Number	`2`
`D`	Day in year	Number	`189`
`d`	Day in month	Number	`10`
`F`	Day of week in month	Number	`2`
`E`	Day in week	Text	`Tuesday`; `Tue`
`a`	AM/PM marker	Text	`PM`
`H`	Hour in day (0-23)	Number	`0`
`k`	Hour in day (1-24)	Number	`24`
`K`	Hour in AM/PM (0-11)	Number	`0`
`h`	Hour in AM/PM (1-12)	Number	`12`
`m`	Minute in hour	Number	`30`
`s`	Second in minute	Number	`55`
`S`	Millisecond	Number	`978`

Table 4-7 (Cont.) Java Pattern Letters

Letter	Date or Time Component	Presentation	Examples
z	Time zone	General Time Zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 Time Zone	-0800

Different presentations in the pattern are as follows:

- Text

  For formatting, if the number of pattern letters is four or more, then the full form is used; otherwise, a short or abbreviated form is used if available. For parsing, both forms are accepted, independent of the number of pattern letters.

- Number

  For formatting, the number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this number. For parsing, the number of pattern letters is ignored unless it is required to separate two adjacent fields.

- Year

  For formatting, if the number of pattern letters is two, then the year is truncated to two digits; otherwise, it is interpreted as a number.

For parsing, if the number of pattern letters is more than two, then the year is interpreted literally, regardless of the number of digits. Using the pattern `MM/dd/yyyy`, `01/11/12` parses to `Jan 11, 12 A.D`.

For parsing with the abbreviated year pattern (`y` or `yy`), the abbreviated year is interpreted relative to some century. The date is adjusted to be within 80 years before and 20 years after the time instance is created. For example, using a pattern of `MM/dd/yy` and `Jan 1, 1997` is created; the string `01/11/12` is interpreted as `Jan 11, 2012`, while the string `05/04/64` is interpreted as `May 4, 1964`. During parsing, only strings consisting of exactly two digits are parsed into the default century. Any other numeric string, such as a one-digit string, a three-or-more-digit string, or a two-digit string that is not all digits (for example, `-1`), is interpreted literally. So, `01/02/3` or `01/02/003` is parsed using the same pattern as `Jan 2, 3 AD`. Likewise, `01/02/-3` is parsed as `Jan 2, 4 BC`.

- Month

  If the number of pattern letters is `3` or more, then the month is interpreted as text; otherwise, it is interpreted as a number.

- General time zone

  Time zones are interpreted as text if they have names. For time zones representing a `GMT` offset value, the following syntax is used:

```
GMTOffsetTimeZone:

   GMT Sign Hours : Minutes

Sign: one of
```

```
      + -

   Hours:

      Digit

      Digit Digit

   Minutes:

      Digit Digit

   Digit: one of

      0 1 2 3 4 5 6 7 8 9
```

`Hours` must be between `0` and `23`, and `Minutes` must be between `00` and `59`. The format is locale-independent and digits must be taken from the Basic Latin block of the Unicode standard.

For parsing, RFC 822 time zones are also accepted.

For formatting, the RFC 822 4-digit time zone format is used:

```
RFC822TimeZone:

      Sign TwoDigitHours Minutes

   TwoDigitHours:

      Digit Digit
```

`TwoDigitHours` must be between `00` and `23`. Other definitions are the same as for general time zones.

For parsing, general time zones are also accepted.

Specifying a Dynamic Outbound File Name

For outbound operation, you can specify a dynamic outbound file name. You can set variables to specify dynamic outbound file names.

Example - Setting Variables to Specify Dynamic Outbound File Names

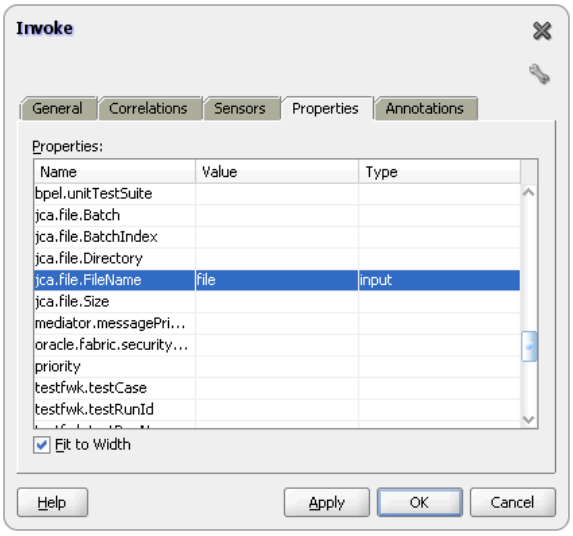
```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="ReadAddressChunk"
              adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks
          /adapter/fw/metadata">
  <connection-factory location=
                    "eis/FileAdapter"
          adapterRef=""/>
  <endpoint-interaction operation
                  ="ChunkedRead">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.
          ChunkedInteractionSpec">
      <property name="PhysicalDirectory"
                         value="C:\foo"/>
      <property name="FileName"
                       value="dummy.txt"/>
      <property name="ChunkSize" value="1"/>
```

```
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

In the preceding example, in the JCA file, the physical directory is set to `"C:\foo"` but during runtime it is dynamically changed to the assigned value. In this example, the physical directory is dynamically changed to `"C:\out".` You must perform the following steps to specify the dynamic outbound directory name:

1. Double-click the **Invoke** activity.
2. Select the **Browse Variables** icon.
3. In the Variable Chooser dialog, click the **Create an Object** icon.
4. Create a variable `file` of type `xsd:string`, as shown in Figure 4-33.
5. Drag and drop an **Assign** activity from the Components window in between the Receive and Invoke activities in the design area.
6. Double-click the **Assign** activity and click the **Copy Operation** tab.
7. Click **Create** and then **Copy Operation**. The **Create Copy Operation** dialog is displayed.
8. In the **Create Copy Operation** dialog, select **Expression** from **Type** and specify the file name, as shown in Figure 4-34. The file operation writes the output file to this file.
9. Click **OK** till you exit the **Assign** activity dialog.
10. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.
11. Click the **Properties** tab.
12. Select the `jca.file.FileName` property from the **Properties** column and set the **Value** as `file` (the file that you created in Step 4.) Ensure that the **Type** column is set to `input`, as shown in Figure 4-36.

Figure 4-36 The Invoke Dialog



Note:

When using dynamic files, ensure that parameters such as `NumberMessages`, `ElapsedTime`, and `FileSize` are not defined in the outbound adapter service `WSDL` file. These parameters are not supported with dynamic files.

Batching Multiple Outbound Messages

In the simplest scenario, you specify writing a single file to a single message. You can also specify the outbound method for batch file writing. This method enables you to specify the number of messages to publish in one batch file. The following batch file settings are provided in the File Configuration page shown in Figure 4-32:

- **Number of Messages Equals**

  Specify a value which, when equaled, causes a new outgoing file to be created.

- **Elapsed Time Exceeds**

  Specify a time which, when exceeded, causes a new outgoing file to be created.

  Note:

  The Elapsed Time Exceeds batching criteria is evaluated and a new outgoing file is created, only when an invocation happens.

  For example, if you specify that elapsed time exceeds 15 seconds, then the first message that is received is not written out, even after 15 seconds, as batching conditions are not valid. If a second message is received, then batching conditions become valid for the first one, and an output file is created when the elapsed time exceeds 15 seconds.

- **File Size Exceeds**

  Specify a file size which, when equaled, causes an outgoing file to be created. For example, assume that you specify a value of `3` for the number of messages received and a value of 1 MB for the file size. When you receive two messages that when combined equal or exceed 1 MB, or three messages that are less than 1 MB, an output file is created.

  Note:

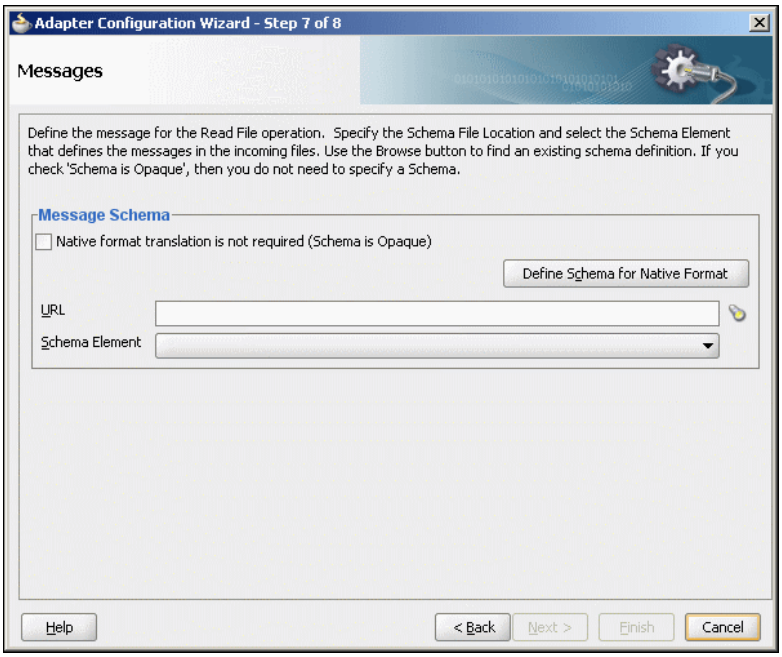
  You *must not* manually change the file configurations specified in the preceding list in the `JCA` files. You must use the Adapter Configuration Wizard to modify these configurations.

If the Oracle File Adapter encounters a problem during batching, it starts batching at the point at which it left off on recovery.

Native Data Translation

The next Adapter Configuration Wizard page that appears is the **Messages** page shown in Figure 4-37. This page enables you to select the XSD schema file for translation.

Figure 4-37 Specifying the Schema



As with specifying the schema for the inbound direction, you can perform the following tasks in this page:

- Specify whether native format translation is not required.
- Select the XSD schema file for translation.
- Start the Native Format Builder wizard to create an XSD file from file formats such as CSV, fixed-length, DTD, and COBOL Copybook.

For more information about Messages page, see Native Data Translation.

Outbound Service Files

When you complete configuration of the Oracle File Adapter with the Adapter Configuration Wizard, a `WSDL` and a `JCA` file pair is generated for the outbound operation. The files are named after the service name you specified on the Service Name page of the Adapter Configuration Wizard shown in Figure 2-8. You can rerun the wizard later to change your operation definitions.

A sample outbound `JCA` file includes the information listed in Table 4-8:

Table 4-8 Sample JCA Properties for Outbound Service

Property	Sample Value
`PhysicalDirectory`	`/tmp/flat/OutputDir`
`FileNamingConvention`	`address-csv%SEQ%.txt`
`Append`	`true`
`NumberMessages`	`1`
`ConcurrentThreshold`	`0`

Table 4-8 (Cont.) Sample JCA Properties for Outbound Service

Property	Sample Value
`OpaqueSchema`	`false`

The outbound Oracle File Adapter uses the following configuration parameters:

- `PhysicalDirectory`
- `LogicalDirectory`
- `NumberMessages`
- `ElapsedTime`
- `FileSize`
- `FileNamingConvention`
- `Append`

For a description of these configuration properties, see Oracle JCA Adapter Properties.

Outbound Headers

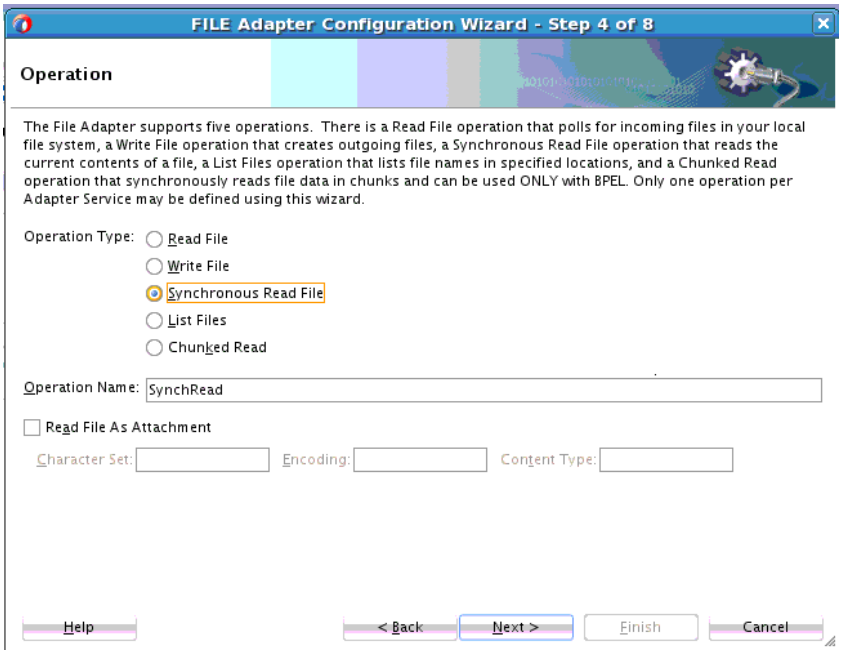
Apart from the payload, the Oracle File Adapter receives the following headers from the component:

- `jca.file.FileName`: file name
- `jca.file.Directory`: directory name

Oracle File Adapter Synchronous Read Concepts

In the outbound direction, the Oracle File or FTP Adapter can read the content of a single file. This section provides an overview of the outbound synchronous file reading capabilities of the Oracle File Adapter. For reading a file synchronously, you select Synchronous Read File operation, as shown in Figure 4-38.

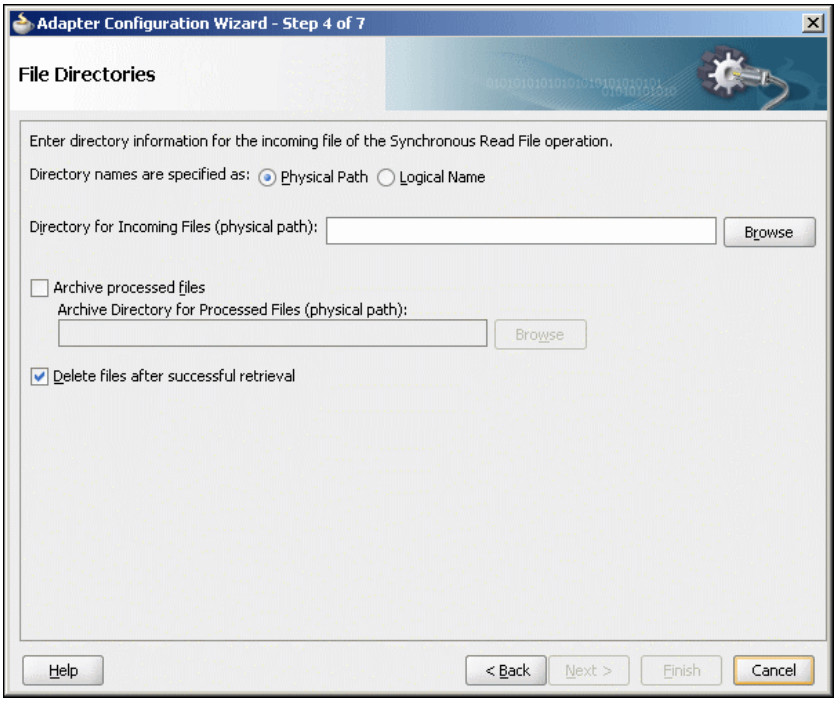
Figure 4-38 Synchronous Read Operation Page



In the outbound direction, the Oracle File/FTP Adapter enables you to read the content of a single file using the Synchronous File Read Operation. This operation now enables you to read the file as an attachment. Select the checkbox to read the file as an attachment,. The rest of the options are optional for attachments and are useful in cases where the information is required by the service engine.

Many of the pages of the Adapter Configuration Wizard are similar to the Read File operation except the File Name page. You can specify the name of the file to be read in the **File Name** field, as shown in Figure 4-39.

Figure 4-39 File Directories Page



Oracle File Adapter File Listing Concepts

This feature of the Oracle File Adapter lets you use a BPEL activity to retrieve a list of files from a target directory. This list of files is returned as an XML document and contains information such as file name, directory name, file size, and last modified time. This section provides an overview of the file listing capabilities of the Oracle File Adapter. You use the Adapter Configuration Wizard to configure the Oracle File Adapter for use with a BPEL process or a Mediator service. This creates an outbound `WSDL` and `JCA` file pair.

Note:

The file creation time property, `creationTime`, is not supported because the standard Java APIs do not provide a mechanism to retrieve the creation time. The value of the `creationTime` property is always displayed as `0`.

For example,

```
<creationTime xmlns="http://xmlns.oracle.com
/pcbpel/adapter/file/FAListFiles/FAListFilesTest/ReadS/">
0</creationTime>
```

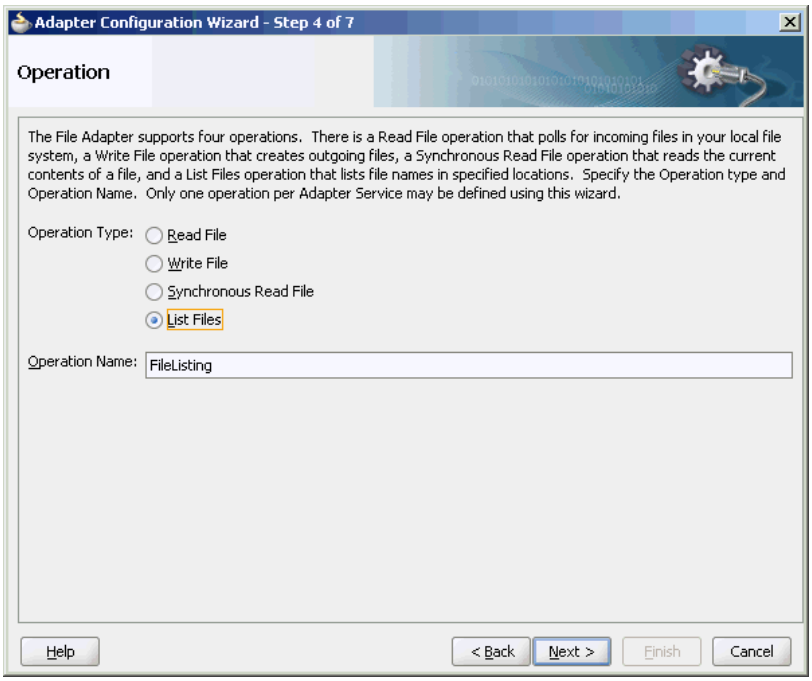
This section includes the following topics:

- Listing Operation
- File Directory Specifications
- File Matching

Listing Operation

For listing files, you must select the List Files operation, as shown in Figure 4-40.

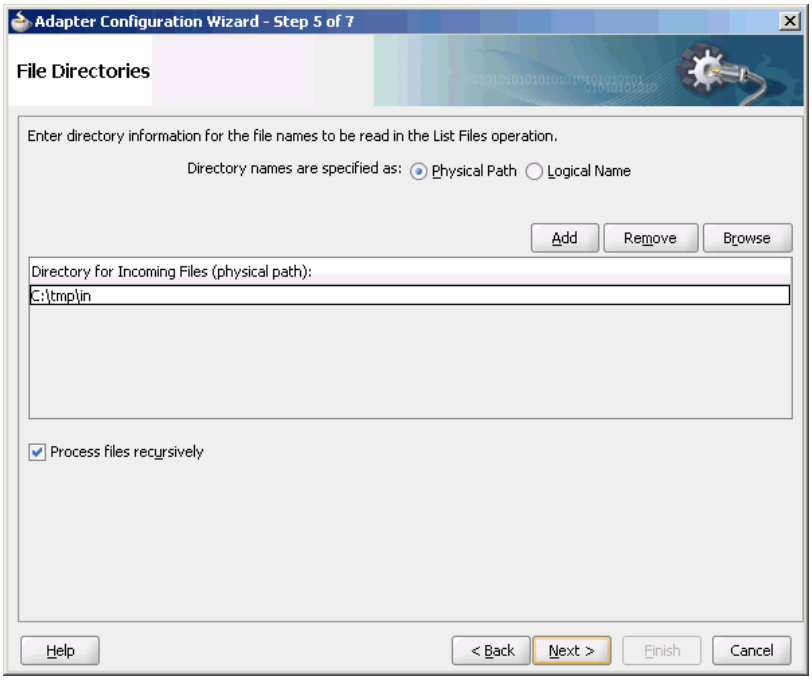
Figure 4-40 List Files Operation Page



File Directory Specifications

The File Directories page of the Adapter Configuration Wizard shown in Figure 4-41 enables you to specify information about the directory to use for reading files names for the list operation. You can choose to list files recursively within directories.

Figure 4-41 The Adapter Configuration Wizard-Specifying Incoming Files



The following section describes the file directory information to specify:

Specifying Inbound Physical or Logical Directory Paths in SOA Composite

You can specify directory names as physical or logical paths for composites involving Oracle BPEL PM and Mediator. Physical paths are values such as `C:\inputDir`.

In the composite, logical properties are specified in the `JCA` file, and their logical-physical mapping is resolved by using binding properties. You specify the required directory once at design time, and you can later modify the directory name as required.

For example, the generated `JCA` file looks as follows for the logical input directory name `C:\inputDir`:

Example - Generated .jca file for Logical Input Directory

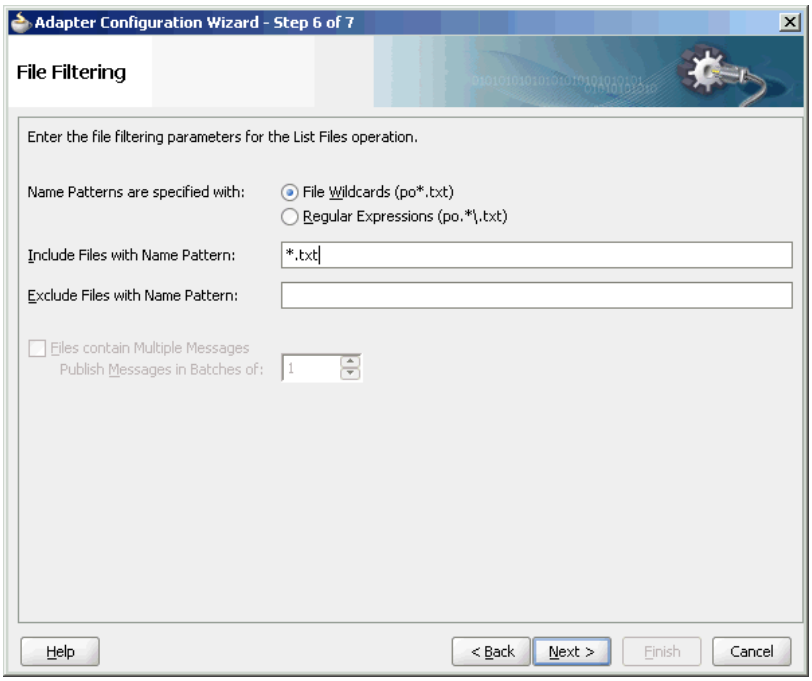
```
<adapter-config name="ListFiles"
       adapter="File Adapter"
   xmlns="http://platform.integration.oracle
   /blocks/adapter/fw/metadata">

  <connection-factory location="eis/FileAdapter"
         UIincludeWildcard="*.txt"
adapterRef=""/>
  <endpoint-interaction portType="FileListing_ptt"
         operation="FileListing">
    <interaction-spec
className="oracle.tip.adapter.file.
         outbound.FileListInteractionSpec">
      <property name="PhysicalDirectory"
           value="C:\inputDir"/>
      <property name="Recursive" value="true"/>
      <property name="IncludeFiles" value=".*\.txt"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

File Matching

The **File Filtering** page of the Adapter Configuration Wizard shown in Figure 4-42 enables you to specify details about the files to retrieve or ignore.

The Oracle File Adapter acts as a file listener and polls the specified directory on a local or remote file system and looks for files that match specified naming criteria.

Figure 4-42 The Adapter Configuration Wizard - File Filtering



The following sections describe the file filtering information to specify:

- Specifying a Naming Pattern
- Including and Excluding Files

Specifying a Naming Pattern

Specify the naming convention that the Oracle File Adapter uses to poll for inbound files. You can also specify the naming convention for files you do not want to process. Two naming conventions are available for selection. The Oracle File Adapter matches the files that appear in the inbound directory.

- File wildcards (`po*.txt`)

  Retrieve all files that start with `po` and end with `.txt`. This convention conforms to operating system standards.

- Regular expressions (`po.*\.txt`)

  Retrieve all files that start with `po` and end with `.txt`. This convention conforms to Java Development Kit (JDK) regular expression (regex) constructs.

Note:

- If you later select a different naming pattern, ensure that you also change the naming conventions you specify in the Include Files and Exclude Files fields. The Adapter Configuration Wizard does not automatically make this change for you.
- Do *not* specify ***.*** as the convention for retrieving files.
- Be aware of any file length restrictions imposed by your operating system. For example, Windows operating system file names cannot be more than 256 characters in length (the file name, plus the complete directory path). Some operating systems also have restrictions on the use of specific characters in file names. For example, Windows operating systems do not allow characters such as `backslash(\)`, slash (`/`), colon (`:`), asterisk (`*`), left angle bracket (`<`), right angle bracket (`>`), or vertical bar(`|`).

Including and Excluding Files

If you use regular expressions, the values you specify in the Include Files and Exclude Files fields must conform to JDK regular expression (regex) constructs. For both fields, different regex patterns must be provided separately. The Include Files and Exclude Files fields correspond to the `IncludeFiles` and `ExcludeFiles` parameters, respectively, of the inbound `WSDL` file.

Note:

The regex pattern complies with the JDK regex pattern. According to the JDK regex pattern, the correct connotation for a pattern of any characters with any number of occurrences is a period followed by a plus sign (`.+`). An asterisk (*) in a JDK regex is not a placeholder for a string of any characters with any number of occurrences.

To have the inbound Oracle File Adapter to pick up all file names that start with `po` and which have the extension `txt`, you must specify the Include Files field as `po.* \.txt` when the name pattern is a regular expression. In this regex pattern example:

- A period (`.`) indicates any character.
- An asterisk (`*`) indicates any number of occurrences.
- A backslash followed by a period (\.) indicates the character period (.) as indicated with the backslash escape character.

The **Exclude Files** field is constructed similarly.

If you have **Include Files** field and **Exclude Files** field expressions that have an overlap, then the exclude files expression takes precedence. For example, if Include Files is set to `abc*.txt` and Exclude Files is set to `abcd*.txt`, then you receive any files prefixed with `abcd*`.

Note:

Do not begin JDK regex pattern names with the following characters: plus sign (+), question mark (?), or asterisk (*).

For details about Java regex constructs, go to

`http://java.sun.com/j2se/1.5.0/docs/api`

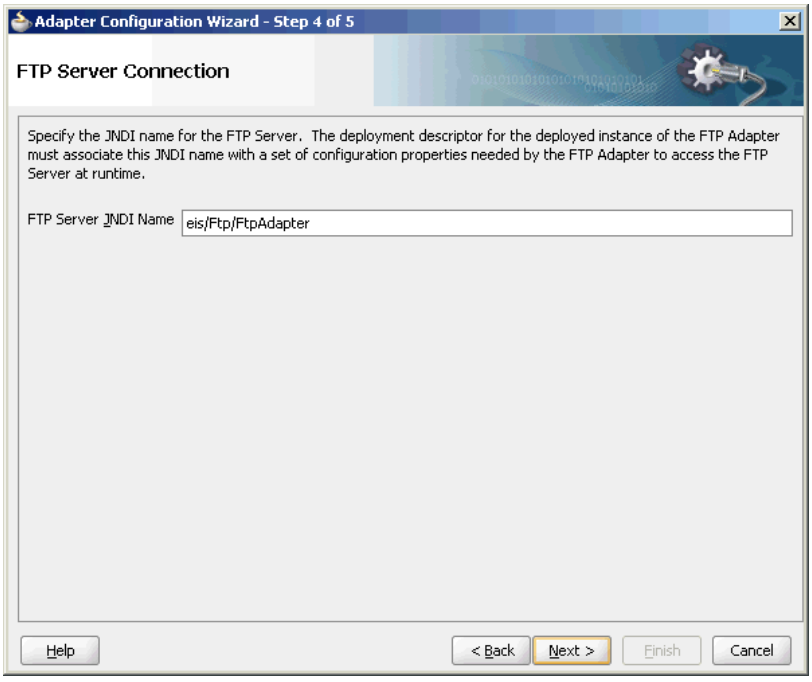
Note:

Files are not read and therefore there is no native data translation.

Oracle FTP Adapter Get File Concepts

In the inbound direction, the Oracle FTP Adapter works the same way as the Read File operations of the Oracle File Adapter in that it polls and gets files from a file system for processing. The major difference is that the Oracle FTP Adapter is used for remote file exchanges. To configure the FTP adapter for remote file exchanges, the Adapter Configuration Wizard asks for connection information to an FTP server to be used later, as shown in Figure 4-43.

Figure 4-43 Specifying FTP Server Connection Information



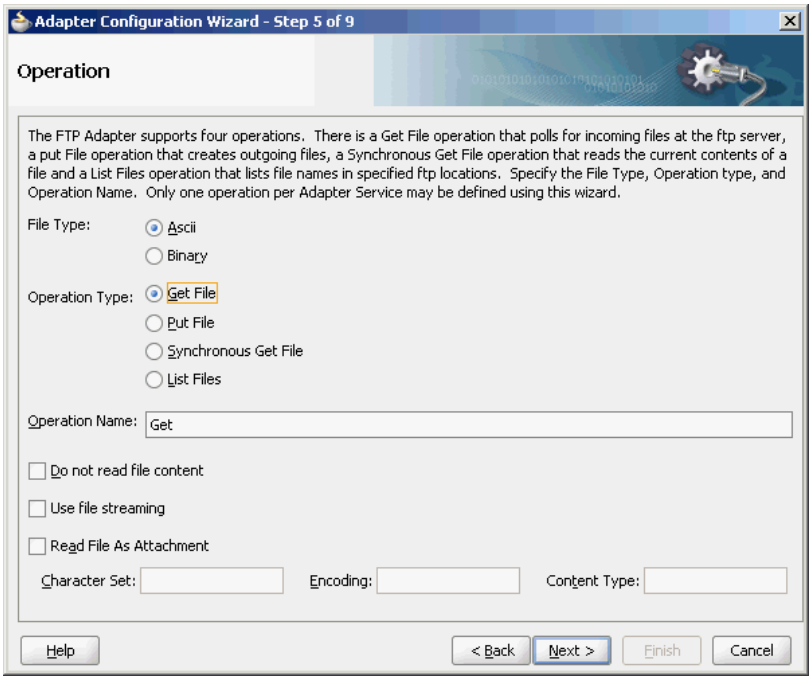
The default adapter instance JNDI name is **`eis/Ftp/FtpAdapter`**, or use a custom name. This name connects to the FTP server during runtime.

Note:

The Oracle FTP Adapter does not support the FTP commands `RESTART` and `RECOVERY` during the transfer of large files.

After logging in, you select the **Get File** (read) operation and the type of file to deliver. Figure 4-44 shows this selection.

Figure 4-44 Selecting the Get File Operation



The `serverType` property in the deployment descriptor is used to determine line separators when you transfer data. You can specify `unix`, `win`, or `mac` as property values. These values represent the operating system on which the FTP server is running. By default, the serverType property contains `unix`.

When you specify `mac` as the value, `\r` is used as line separator. For `unix`, `\n` is used and for `win`, `\r\n` is used. You must note that this property is used by the NXSD translator component to write the line separator during an outbound operation.

From this point onwards, pages of the Adapter Configuration Wizard for the Get File operation are the same as those for the Read File operation of the file. Table 4-9 lists the pages that are displayed and provides references to sections that describe their functionality.

Table 4-9 Adapter Configuration Wizard Windows for Get File Operation

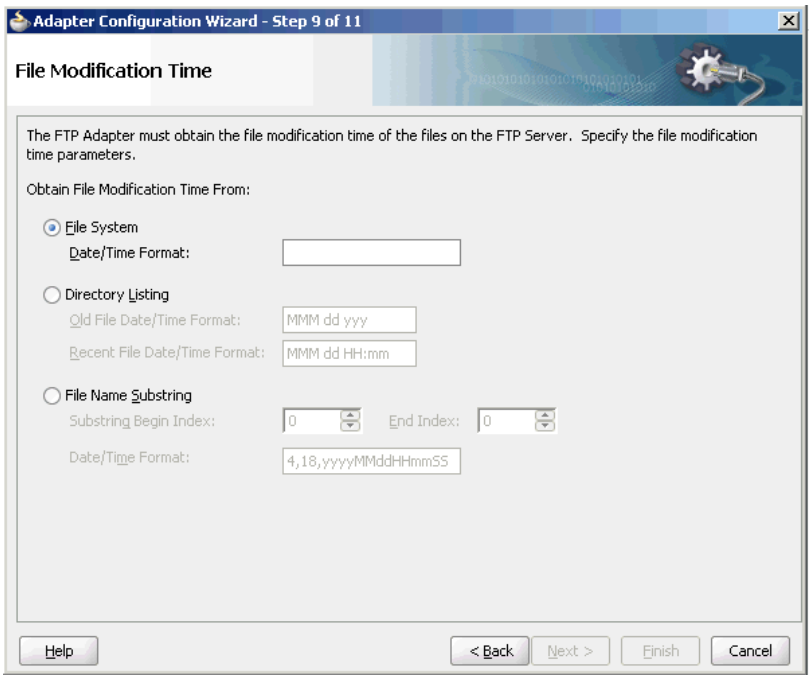
Page	See Section...
File Directories (Figure 4-26)	Inbound File Directory Specifications
File Filtering (Figure 4-27)	File Matching and Batch Processing
File Polling (Figure 4-28)	File Polling
Messages (Figure 4-29)	Native Data Translation

An additional Adapter Configuration Wizard page is also available for advanced users. This page is shown in Figure 4-45 and appears only after you make either or both of the following selections on the File Polling page shown in Figure 4-28:

- Do not select the **Delete Files After Successful Retrieval** check box.

- Set the value of the **Minimum File Age** field to a value greater than **0**.

Figure 4-45 File Modification Time



This page enables you to specify a method for obtaining the modification time of files on the remote FTP server:

Note:

The Oracle FTP Adapter uses the `LIST` command as opposed to `NLST` for listing and retrieves the time stamps because of which you must not specify the time formats. However, you must specify the time formats as shown if you do any of the following:

- If you specify `NLST` as the listing command (either through the mapping file or the `UseNlst="true"` parameter in the inbound`JCA` file)
- To use the **File Name** Substring option

This note is not applicable if your case does not fall under neither of these categories.

- **File System**

  This option enables you to obtain the date/time format of the file modification time with the file system listing command. However, this option is rarely used and is not supported by all FTP servers. See your FTP server documentation to determine whether your server supports the file system listing command, which command-line syntax to use, and how to interpret the output.

  For example, if the file system listing command `quote mdtm` *`filename`* is supported and returns the following information:

```
213 20050602102633
```

specify the start index, end index, and date/time format of the file modification time in the **Data/Time Format** field as a single value separated by commas (for example, **4,18,yyyyMMddHHmmss**).

Where:

– 4 is the start index of the file modification time.

– **18** is the end index of the file modification time.

– yyyyMMddHHmmss is the data/time format of the file modification time obtained with the `quote mdtm` *`filename`* command.

The resulting JCA file includes the following parameters and values:

```
<property name=" FileModificationTime " value=" FileSystem "/>
<property name=" ModificationTimeFormat" value=" 4,18,yyyyMMddHHmmss "/>
```

To handle the time zone issue, you must also be aware of the time stamp difference. The time zone of the FTP server is determined by using the Windows date/time properties (for example, by double-clicking the time being displayed in the Windows task bar). You must then convert the time difference between the FTP server and the system on which the Oracle FTP Adapter is running to milliseconds and add the value as a binding property in the `composite.xml` file:

```
<binding.jca config="FlatStructureIn_file.jca">
   <property name="timestampOffset" source="" type="xs:string" many="false"
override="may">238488888</property-->
</binding.jca>
```

- **Directory Listing**

  This option enables you to obtain the date/time format from the file modification time with the FTP directory listing command. For example, if the directory listing command (`ls -l`) returns the following information:

  ```
  12-27-04  07:44AM                2829 NativeData2.txt
  ```

  specify the start index, end index, and date/time format of the file modification time as a single value separated by commas in either the Old File Date/Time Format field or the Recent File Date/Time Format field (for example, `0,17,MM-dd-yy hh:mma`).

  Where:

  – `0` is the start index of the file modification time.

  – `17` is the end index of the file modification time.

  – MM-dd-yy hh:mma is the date/time format of the file modification time obtained with the `ls -l` command. For this example, the value is entered in the Recent File Date/Time Format field. This field indicates that the format is obtained from the most recent file adhering to the naming convention, whereas the Old File Date/Time Format field obtains the format from the oldest file.

  The resulting JCA file includes the following parameters and values:

  ```
  <property name=" FileModificationTime " value=" DirListing"/>
  <property name=" ModificationTimeFormat" value="0,17, MM-dd-yy hh:mma "/>
  ```

  To handle the time zone issue, you must also be aware of the time stamp difference. The time zone of the FTP server is determined by using the Windows

date/time properties (for example, by double-clicking the time being displayed in the Windows task bar). You must then convert the time difference between the FTP server and the system on which the Oracle FTP Adapter is running to milliseconds and add the value as a binding property in the `composite.xml` file:

```
<binding.jca config="FlatStructureIn_file.jca">
   <property name="timestampOffset" source="" type="xs:string" many="false"
override="may">238488888</property-->
</binding.jca>
```

- **File Name Substring**

  This option enables you to obtain the modification time from the file name. For example, if the name of the file is `fixedLength_20050324.txt`, you can specify the following values:

  – The start index in the S**ubstring Begin Index** field (for example, `12`)

  – The end index in the **End Index** field (for example, `20`)

  – The date and time format in the **Date/Time Format** field conforming to the Java `SimpleDateFormat` to indicate the file modification time in the file name (for example, `yyyyMMdd`)

  The resulting JCA file includes the following parameters and values:

```
<property name=" FileModificationTime " value=" Filename"/>
<property name=" FileNameSubstringBegin " value="12 "/>
<property name=" FileNameSubstringEnd " value="20"/>
<property name=" ModificationTimeFormat " value=" yyyyMMdd "/>
```

After the completion of the Adapter Configuration Wizard, configuration files are created in the Applications section of JDeveloper.

See Figure 2-28 for more information about error handling.

You must also add the `DefaultDateFormat` and the `RecentDateFormat` parameters to the deployment descriptor for Oracle FTP Adapter, as shown in the following sample:

```
<non-managed-connection
managedConnectionFactoryClassName=
          "oracle.tip.adapter.ftp.
             FTPManagedConnectionFactory">
   <property name="host" value="localhost"/>
   <property name="port" value="21"/>
   <property name="username" value="****"/>
   <property name="password" value="****"/>
    <property name="listParserKey" value="UNIX"/>
    <property name="defaultDateFormat"
                  value="MMM d yyyy"/>
    <property name="recentDateFormat"
                  value="MMM d HH:mm"/>
</non-managed-connection>
```

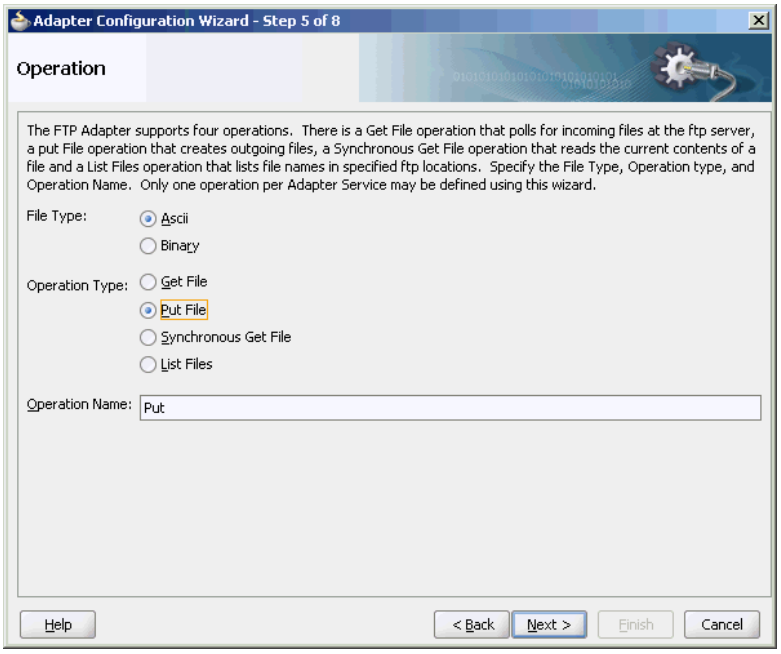
For more information on the `DefaultDateFormat` and the `RecentDateFormat` parameters, refer to Recursive Processing of Files Within Directories in Oracle File and FTP Adapters.

Oracle FTP Adapter Put File Concepts

In the outbound direction, the Oracle FTP Adapter works the same as the Write File operations of the Oracle File Adapter. The Oracle FTP Adapter receives messages from a BPEL process or a Mediator service and writes the messages in a file to a file system (in this case, remote). Because the messages must be written to a remote system, the Adapter Configuration Wizard prompts you to connect to the FTP server with the adapter instance JNDI name, as shown in Figure 4-43.

After logging in, you select the Put File (write) operation and the type of file to deliver. Figure 4-46 shows this selection.

Figure 4-46 Selecting the Put File Operation



From this point onwards, pages of the Adapter Configuration Wizard for the Put File operation are the same as those for the Write File operation of the Oracle File Adapter. Table 4-10 lists the pages that display and provide references to sections that describe their functionality.

Table 4-10 The Adapter Configuration Wizard Pages for Put File Operation

Page	See Section...
File Configuration (Figure 4-32)	Outbound File Directory Creation
Messages (Figure 4-37)	Native Data Translation

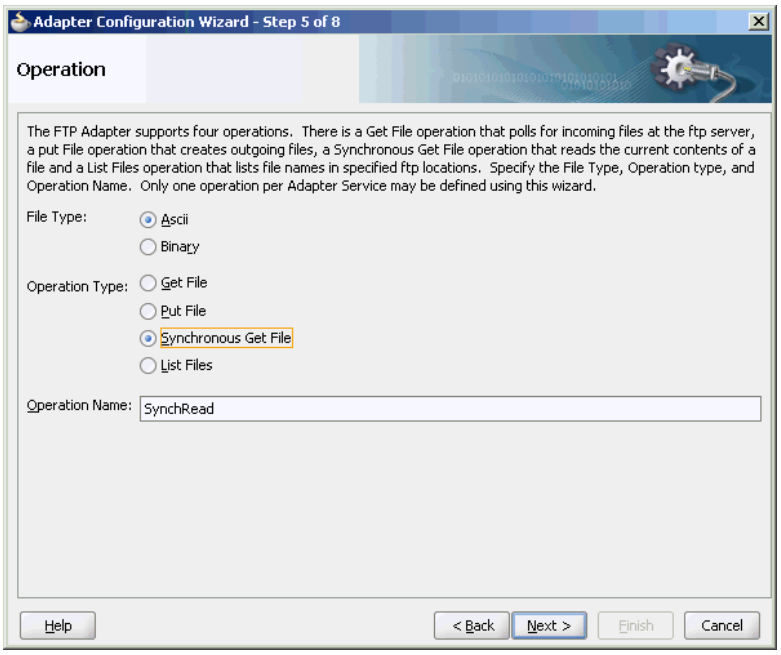
After the completion of the Adapter Configuration Wizard, configuration files are created in the Applications section of JDeveloper.

Oracle FTP Adapter Synchronous Get File Concepts

In the outbound direction, the Oracle FTP Adapter works the same way as the Synchronous Read File operations of the Oracle File Adapter in that it polls and gets files from a file system and reads the current contents of the file. The major difference

is that the Oracle FTP Adapter is used for remote file exchanges. Because of this polling, the Adapter Configuration Wizard asks for connection information to an FTP server to be used later. For reading a file synchronously, you select **Synchronous Get File** operation, as shown in Figure 4-47.

Figure 4-47 Selecting the Synchronous Get File Operation



Oracle FTP Adapter File Listing Concepts

The Oracle FTP Adapter file listing concepts are similar to the Oracle File Adapter file listing concepts discussed in File Listing Concepts. The Oracle FTP Adapter polls for files in a target directory and lists files from the target directory to specified FTP locations. The contents of the files are not read. This feature of the Oracle FTP Adapter lets you use an invoke activity to retrieve a list of files from a target directory. This list of files is returned as an XML document and contains information such as file name, directory name, file size, and last modified time.

Note:

The file creation time property, `creationTime`, is not supported for FTP because the standard Java APIs do not provide a mechanism to retrieve the creation time. The value of the `creationTime` property is always displayed as `0`.

The `creationTime` property is supported for SFTP only.

You use the Adapter Configuration Wizard to configure the Oracle FTP Adapter for use with a BPEL process or a Mediator service. This creates an outbound `WSDL` and `JCA` file pair.

For listing files, you must select the `List Files` operation from the Operation Type page of the Adapter Configuration Wizard. In the File Directories page of the Adapter Configuration Wizard, you must specify information about the directory to use for reading file names for the list operation. You can choose to list files recursively within

directories. The **File Filtering** page of the Adapter Configuration Wizard enables you to specify details of the files to retrieve or ignore.

The Oracle FTP Adapter acts as a listener and polls the specified directory on a local or remote file system and looks for files that match specified naming criteria.

File and FTP Adapter Extensions

There are four File/FTP Adapter Extensions:

- FTP Adapter extension to the login() operation in the default FTPClient implementation.

  FTP Adapter extension to support MLSD command.

- FTP Adapter extension to extend the Listing operation to send the MLSD command instead of the LIST command.

- FTP Adapter extension to the FTP Store operation to send additional proprietary FTP commands to an FTP server running on an MVS platform.

Each of these extensions is discussed in detail in the File and FTP Adapter use cases.

Configuring Oracle File and FTP Adapters

Various configuration tasks for Oracle File and FTP Adapters are discussed in the following sections:

- Configuring the Credentials for Accessing a Remote FTP Server
- Configuring Oracle File and FTP Adapters for High Availability
- Using Secure FTP with the Oracle File and FTP Adapters
- Using SFTP with Oracle File and FTP Adapters
- Configuring Oracle File and FTP Adapters for HTTP Proxy
- Configuring Oracle File and FTP Adapters for High Availability

---

Note:

You can use the `ftpAbsolutePathBegin` parameter to indicate to the adapter whether a directory name used in any FTP command issued by the FTP adapter is an absolute directory or relative directory. If the directory name starts with the `ftpAbsolutePathBegin`, it is an absolute directory, otherwise it is treated as a relative directory.

Configuring the Credentials for Accessing a Remote FTP Server

To access a remote FTP server, you must configure the following credentials:

- User name: the user name to use on the remote FTP server.
- Password: the password to use on the remote FTP server.
- Port: 21

- Host: the IP address of the remote FTP server.

You must configure these credentials by modifying the `weblogic-ra.xml` file using the Oracle WebLogic Server console.

To do so, in the Oracle WebLogic Server Admin Console:

1. Select **Deployments** from the Navigation pane on the left.
2. Select **FtpAdapter** from the table of Deployments shown on the right.
3. Select the **Configuration** subtab for the FtpAdapter and then **Outbound Connection Pools**.
4. Expand `javax.resource.cci.ConnectionFactory` and then select the instance that you are modifying. (For example, choose the `eis/Ftp/FtpAdapter` instance for the non-High Availability use case.)

Configuring Oracle File and FTP Adapters for High Availability

The requirements and procedure to configure the Oracle File and FTP Adapters for high availability for an active-active topology are discussed in the following sections:

- Prerequisites for High Availability
- High Availability in Inbound Operations
- High Availability in Outbound Operations

Prerequisites for High Availability

Before you configure the Oracle File or FTP Adapter for high availability, you must ensure that the following prerequisites are met:

- Clustered processes must use the same physical directory.
- Connection-factories must specify the same shared directory as the control directory, and their names must match. For example, if the deployment descriptor for one connection factory has `/shared/control_dir` as the value for `controlDir`, then the other deployment descriptor must also have the same value.
- Fault-policies and fault-bindings must be created for remote faults to ensure that the adapter acts correctly. For more information on fault-policies and fault-bindings, see Error Handling.
- The `MaxRaiseSize` property must be set in the inbound JCA file.

Note:

For large payloads, you must increase the transaction time out for the `SOADataSource` by adding the following:

```
<xa-set-transaction-timeout>true</xa-set-transaction-timeout>
<xa-transaction-timeout>1000</xa-transaction-timeout>
```

Note:

For Windows platforms, you must ensure that the input and output directories are made canonical. For example, you must use `C:\bpel\input` instead of `c:\bpel\input`. Note the use of capitalized drive letter `C:` instead of `c:`.

Note:

On all platforms, you must not end input or output directory names with the Java system property file.separator value. For example, `/tmp/file/in/` is invalid but `/tmp/file/in` is valid as in the former the file separator slash is at the end.

High Availability in Inbound Operations

The Oracle File and FTP Adapters must ensure that only one node processes a particular file in a distributed topology. You can use the database table as a coordinator to ensure that Oracle File and FTP Adapters are highly available for inbound operations.

Using Database Table as a Coordinator

You must use the following procedure to make an inbound Oracle File or FTP Adapter service highly available by using database table as a coordinator:

Note:

You must increase global transaction timeouts if you use database as a coordinator.

1. **Create Database Tables**

   You are not required to perform this step because the database schemas are pre-created as a part of soainfra.

2. **Modify Deployment Descriptor for Oracle File Adapter**

   Modify Oracle File Adapter deployment descriptor for the connection-instance corresponding to `eis/HAFileAdapter` from the Oracle WebLogic Server Administration Console:

   a. Log in to your Oracle WebLogic Server Administration Console. To access the console, navigate to `http://`*`servername`*`:`*`portnumber`*`/console`.

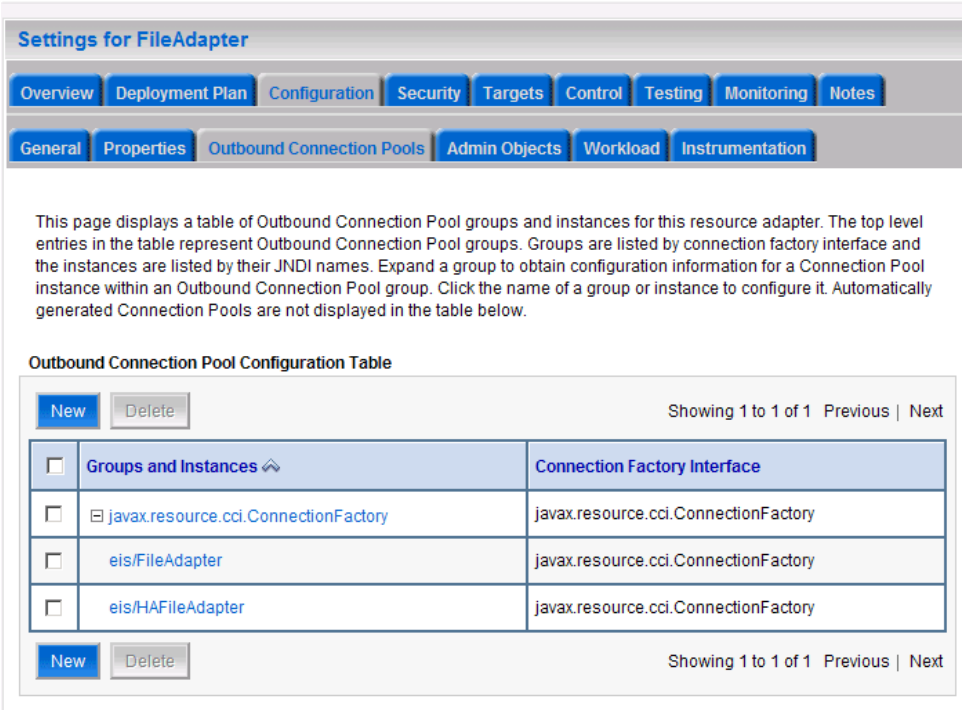
   b. Click **Deployments** in the left pane for Domain Structure.

   c. Click **FileAdapter** under S**ummary of Deployments** on the right pane.

   d. Click the **Configuration** tab.

   e. Click the **Outbound Connection Pools** tab, and expand **javax.resource.cci.ConnectionFactory** to see the configured connection factories, as shown in Figure 4-48:

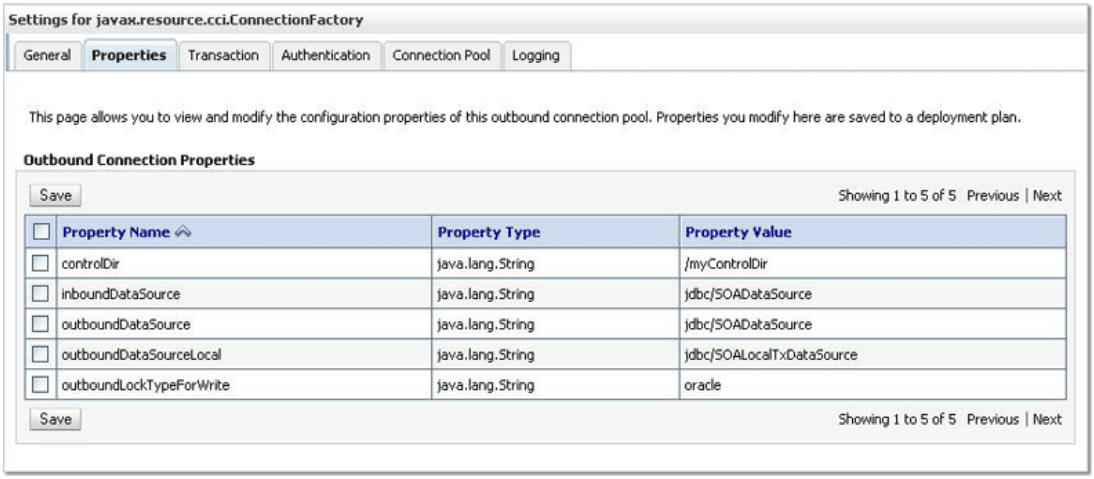
Figure 4-48 Oracle WebLogic Server Administration Console - Settings for FileAdapter Page



f. Click **eis/HAFileAdapter**. The **Outbound Connection Properties** for the connection factory corresponding to high availability is displayed.

g. Update the connection factory properties, as shown in Figure 4-49.

Figure 4-49 Oracle WebLogic Server Administration Console - Settings for javax.resource.cci.ConnectionFactory Page



The new parameters in connection factory for Oracle File and FTP Adapters are as follows:

`controlDir` - Set it to the directory structure where you want the control files to be stored. You must set it to a shared location if multiple WebLogic Server instances run in a cluster.

`inboundDataSource` - Set the value to `jdbc/SOADataSource`. This is the data source, where the schemas corresponding to high availability are pre-created. The pre-created schema file can be found under `$BEA_HOME/AS11gR1SOA/rcu/integration/soainfra/sql/adapter/createschema_adapter_oracle.sql`. To create the schemas elsewhere, use this script. You must set the inboundDataSource property accordingly if you choose a different schema.

h. Configure BPEL Process or Mediator Scenario to use the connection factory, as shown in the following example:

```
<adapter-config name="FlatStructureIn"
                    adapter="File Adapter"
xmlns="http://platform.integration.                oracle/blocks/adapter/fw/
metadata">
  <connection-factory location=
                     "eis/HAFileAdapter"
UIincludeWildcard="*.txt" adapterRef=""/>
  <endpoint-activation portType="Read_ptt"
                          operation="Read">
    <activation-spec
className="oracle.tip.adapter.
file.inbound.FileActivationSpec"../>
      <property../>
      <property../>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

Note:

The location attribute is set to `eis/HAFileAdapter` for the connection factory.

High Availability in Outbound Operations

The Oracle File and FTP Adapters must ensure that if multiple references write to the same directory, then these do not overwrite each other. The following locking capabilities you can use to make Oracle File and FTP Adapters highly available for outbound operations:

- Database mutex
- User-defined mutex

Using a Database Mutex

You must use the following procedure to make an outbound Oracle File or FTP Adapter service highly available by using database table as a coordinator:

Note:

You must increase global transaction timeouts if you use the database as a coordinator.

---

1. Create Database Tables

You are not required to perform this step as the database schemas are precreated as a part of soainfra.

2. **Modify Deployment Descriptor for Oracle File Adapter**

   Modify Oracle File Adapter deployment descriptor for the connection-instance corresponding to `eis/HAFileAdapter` from the Oracle WebLogic Server Administration Console:

   a. Log in to your Oracle WebLogic Server Administration Console. To access the console, navigate to `http://servername:portnumber/console`.

   b. Click **Deployments** in the left pane for Domain Structure.

   c. Click **FileAdapter** under Summary of Deployments on the right pane.

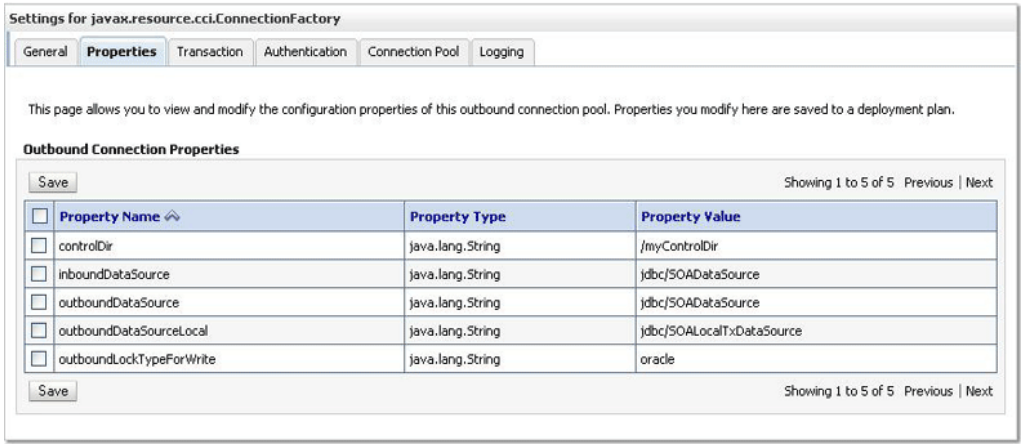
   d. Click the **Configuration** tab.

   e. Click the **Outbound Connection Pools** tab, and expand **javax.resource.cci.ConnectionFactory** to see the configured connection factories, as shown in Figure 4-48.

   f. Click **eis/HAFileAdapter**. The Outbound Connection Properties page is displayed with the connection factory corresponding to high availability.

   g. Update the connection factory properties, as shown in Figure 4-50.

   Figure 4-50 Oracle WebLogic Server Administration Console - Settings for javax.resource.cci.Connectionfactory Page



   The new parameters in connection factory for Oracle File and FTP Adapters are as follows:

   `controlDir` - Set it to the directory structure where you want the control files to be stored. You must set it to a shared location if multiple WebLogic Server instances run in a cluster.

   `inboundDataSource` - Set the value to `jdbc/SOADataSource`. This is the data source, where the schemas corresponding to high availability are precreated. The precreated schemas can be found under `$BEA_HOME/AS11gR1SOA/rcu/integration/soainfra/sql/adapter/createschema_adapter_oracle.sql`. To create the schemas elsewhere, use this script. You must set the inboundDataSource property accordingly if you choose a different schema.

`outboundDataSource` - Set the value to `jdbc/SOADataSource`. This is the data source where the schemas corresponding to high availability are precreated. The precreated schemas can be found under `$BEA_HOME/AS11gR1SOA/rcu/integration/soainfra/sql/adapter/createschema_adapter_oracle.sql`. To create the schemas elsewhere, use this script. You must set the `outboundDataSource` property if you choose to do so.

`outboundLockTypeForWrite` - Set the value to `oracle` if you are using Oracle Database. By default the Oracle File and FTP Adapters use an in-memory mutex to lock outbound write operations. You must choose from the following values for synchronizing write operations:

`memory` - The Oracle File and FTP Adapters use an in-memory mutex to synchronize access to the file system.

oracle - The adapter uses the Oracle Database sequence.

`db` - The adapter uses a precreated database table (`FILEADAPTER_MUTEX`) as the locking mechanism. You must use this option only if you are using a schema other than the Oracle Database schema.

`user-defined` - The adapter uses a user-defined mutex. To configure the user-defined mutex, you must implement the mutex interface `oracle.tip.adapter.file.Mutex` and then configure a new binding-property with the name `oracle.tip.adapter.file.mutex` and value as the fully qualified class name for the mutex for the outbound reference.

h. Configure BPEL Process or Mediator Scenario to use the connection factory, as shown in the following example:

```
<adapter-config name="FlatStructureOut" adapter="File Adapter"
xmlns="http://platform.integration.                    oracle/blocks/
adapter/fw/metadata">
  <connection-factory                  location="eis/HAFileAdapter"
adapterRef=""/>
  <endpoint-interaction portType="Write_ptt" operation="Write">
<interaction-spec
className="oracle.tip.adapter.file.outbound                          .Fil
eInteractionSpec">
      <property../>
      <property../>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

Note:

The location attribute is set to `eis/HAFileAdapter` for the connection factory.

You can change the connection-factory location dynamically using jca header properties in both BPEL as well as Mediator service engines. To dynamically set FTP JCA connection factory parameters, either remove the location (jndi) or give an invalid value (non existing jndi). This happens because, in non-managed connection factory when `location=<jndi>` is specified, the parameters defined in the outbound connections jndi get higher precedence. That means, the values that are specified in .jca will not take effect.

Using Secure FTP with the Oracle FTP Adapter

The Oracle FTP Adapter supports the use of the secure FTP feature on Windows, Solaris, and Linux. For Windows, this feature is certified on FileZilla FTP server with OpenSSL. This section provides an overview of secure FTP functionality and describes how to install and configure this feature.

This section includes the following topics:

- Secure FTP Overview
- Installing and Configuring FTP Over SSL on Solaris and Linux
- Installing and Configuring FTP Over SSL on Windows

Secure FTP Overview

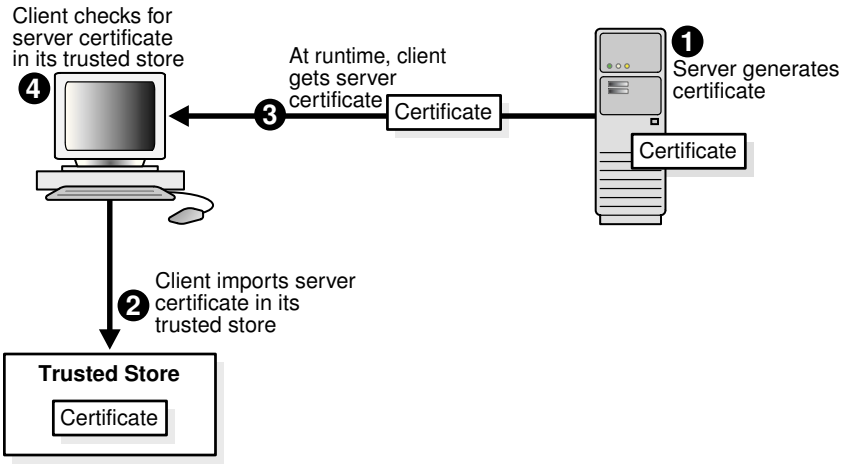
In environments in which sensitive data is transferred to remote servers (for example, sending credit card information to HTTP servers), the issue of security is very important. Security in these cases primarily refers to two requirements:

- Trust in the remote server with which you are exchanging data
- Protection from third parties trying to intercept the data

Secure socket layer (SSL) certificates and encryption focus on satisfying these two security requirements. When SSL is used for FTP, the resulting security mechanism is known as FTPS (or FTP over SSL).

To gain the trust of clients in SSL environments, servers obtain certificates (typically, X.509 certificates) from recognized certificate authorities. When you set up the FTP server, you use openSSL to create a certificate for the server. Every client trusts a few parties, to begin with. If the server is one of these trusted parties, or if the server's certificate was issued by one of these parties, then you have established trust, even indirectly. For example, if the server's certificate was issued by authority A, which has a certificate issued by authority B, and the client trusts B, that is good enough. For the setup shown in Figure 4-51, the server's certificate is directly imported into the client's certificate store as a trusted certificate.

Figure 4-51 Establishing Trust



You make the data being transferred immune to spying by encrypting it before sending it and decrypting it after receiving it. Symmetric encryption (using the same

key to encrypt and decrypt data) is much faster for large amounts of data than the public key and private key approach. Symmetric encryption is the approach used by FTPS. However, before the client and server can use the same key to encrypt and decrypt data, they must agree on a common key. This client typically does this by performing the following tasks:

- Generating a session key (to be used to encrypt and decrypt data)
- Encrypting this session key using the server's public key that is part of the server's certificate
- Sending the key to the server

The server decrypts this session key by using its private key and subsequently uses it to encrypt file data before sending it to the client.

Installing and Configuring FTP Over SSL on Solaris and Linux

The following subsections describe how to install and configure secure FTP for Solaris and Linux:

- Installing and Configuring OpenSSL
- Installing and Configuring vsftpd
- Setting Up the Oracle File and FTP Adapters

Installing and Configuring OpenSSL

OpenSSL is an open source implementation of the SSL protocol. OpenSSL implements basic cryptographic functions and provides utility functions. Install and configure OpenSSL on the Solaris or Linux host to be used as the FTP server.

1. Go to the following URL:

   ```
   http://www.openssl.org/source
   ```

2. Locate `openssl-0.9.7g.tar.gz` in the list of available files. For example:

   ```
   3132217 Apr 11 17:21:51 2005
                  openssl-0.9.7g.tar.gz (MD5) (PGP sign)
   ```

3. Download the following files:
   - `openssl-0.9.7g.tar.gz`
   - openssl-0.9.7g.tar.gz.md5 (under the `MD5` link)
   - openssl-0.9.7g.tar.gz.asc (under the `PGP sign` link
4. Unzip the following file using `gunzip`.

   ```
   gunzip openssl-0.9.7g.tar.gz
   ```

5. Untar the following file:

   ```
   tar xvf openssl-0.9.7g.tar
   ```

6. Change directories to the following location:

   ```
   cd openssl-0.9.7g
   ```

7. Run the following command:

```
./config --prefix=/usr --openssldir=/usr/local/openssl
```

8. Change to the Bourne shell (if you are not using it):

```
sh
```

9. Configure and export the PATH variable:

```
PATH=${PATH}:/usr/ccs/bin; export PATH
```

10. Run the following command:

```
make
```

11. Exit the Bourne shell:

```
exit
```

12. Run the following command:

```
make test
```

13. Log in as the super user:

```
msu
```

14. Enter the password when prompted.

15. Run the following command:

```
make install
```

Installing and Configuring vsftpd

The vsftpd server is a secure and fast FTP server for UNIX systems. Install and configure vsftpd on the Solaris or Linux host to be used as the FTP server.

1. Go to the following location:

```
ftp://vsftpd.beasts.org/users/cevans/
```

2. Download `vsftpd-2.0.5` (You must have the tar and signature file (`.asc` file)). For example:

```
[BINARY]     vsftpd-2.0.5.tar.gz. . . . . . . . . . .    [Mar 19 21:26]    149K
[FILE]       vsftpd-2.0.5.tar.gz.asc. . . . . . . . .    [Mar 19 21:26]    189B
```

3. Unzip the following file using `gunzip`.

```
gunzip vsftpd-2.0.5.tar.gz
```

4. Unzip the tar file:

```
tar xvf vsftpd-2.0.5.tar
```

5. Change directories to the following location:

```
cd vsftpd-2.0.5
```

6. Make the following change in the `builddefs.h` file:

```
#undef VSF_BUILD_SSL
```

to

```
#define VSF_BUILD_SSL
```

7. Log in as the super user:

```
msu
```

8. Enter the password when prompted.

9. Create a file named `vsftpd.conf` with the following settings in the `/etc` directory:

```
# Standalone mode
listen=YES
max_clients=200
max_per_ip=4
# Access rights
anonymous_enable=YES
#chroot_local_user=YES
#userlist_enable=YES
ftp_username=ftp
local_enable=YES
write_enable=YES
anon_upload_enable=YES
anon_mkdir_write_enable=YES
anon_other_write_enable=YES
chown_uploads=YES
chown_username=ftp
# Security
anon_world_readable_only=NO
allow_anon_ssl=YES
ssl_enable=YES
connect_from_port_20=YES
hide_ids=YES
pasv_min_port=50000
pasv_max_port=60000
# Features
ftpd_banner="Welcome to the FTP Service"
xferlog_enable=YES
ls_recurse_enable=NO
ascii_download_enable=NO
async_abor_enable=YES
# Performance
one_process_model=NO
idle_session_timeout=120
data_connection_timeout=300
accept_timeout=60
connect_timeout=60
anon_max_rate=50000
```

Note:

Copies of the `vsftpd.conf` file appear in several locations in the `vsftpd-2.0.5` directory structure. If you use one of those files to create the `vsftpd.conf` file in the `/etc` directory, then ensure that it only includes the parameters and settings described in Step 9.

---

10. Run the following commands:

```
mkdir /var/ftp
useradd -d /var/ftp ftp
chown root /var/ftp
chmod og-w /var/ftp
mkdir /usr/share/empty
mkdir /usr/share/ssl
mkdir /usr/share/ssl/certs
```

11. Run the following command:

```
openssl req -x509 -nodes -newkey rsa:1024 -keyout /usr/share/ssl/certs/vsftpd.pem
-out /usr/share/ssl/certs/vsftpd.pem
```

12. Run the `vsftpd` daemon from the `vsftpd-2.0.5` directory:

```
./vsftpd
```

Setting Up the Oracle FTP Adapter

Perform the following tasks to set up the Oracle FTP Adapter:

1. On your Solaris or Linux host, run the following commands:

```
mkdir /var/ftp/inDir
mkdir /var/ftp/outDir
chmod 777 /var/ftp/inDir /var/ftp/outDir
```

2. Specify the FTP connection parameters in the Oracle FTP Adapter deployment descriptor from the Oracle WebLogic Server Administration Console.

Where...	Is...
`useFtps`	Set to `True`. This setting is required to use FTP over SSL. The default is `False`.
`channelMask`	The type of channel: control channel or data channel. Possible values are `both`, `control`, `data`, or `none`. The default is `both`.
`securePort`	The port for FTP over SSL. The default is `990`.
`keyStoreProviderName`	The keystore provider class. The default is `oracle.security.pki.OraclePKIProvider`.
`keystoreType`	The keystore type. The default is `PKCS12`.
`keystoreAlgorithm`	The keystore algorithm. The default is `OracleX509`.
`enableCipherSuits`	List of comma separated cipher suites. The default is blank, in which case the default list of cipher suites are used. For most cases, you are not required to change this.
`pkiProvider`	The PKI provider name. The default is `OraclePKI`.
`jsseProvider`	The JSSE provider name. The default is `OracleJSSE`.

You have now installed and configured secure FTP and are ready to use this feature with the Oracle FTP Adapter.

Installing and Configuring FTP Over SSL on Windows

The FTPS feature is certified on FileZilla FTP server with OpenSSL. You must follow the procedure in the following subsections for installing and configuring OpenSSL for FileZilla on Windows:

- Installing OpenSSL
- Generating OpenSSL Server Key and Certificate
- Importing the Server Key and Certificate Into FileZilla Server
- Converting the Server Key From PEM to PKCS12 Format
- Configuring Oracle FTP Adapter Deployment Descriptor to Use the New Key

Installing OpenSSL

OpenSSL is an open source implementation of the SSL protocol. OpenSSL implements basic cryptographic functions and provides utility functions. Perform the following steps to install and configure OpenSSL on the Windows host to be used as the FTP server.

1. Go to the following URL:

   `http://www.slproweb.com/products/Win32OpenSSL.html`

2. Download and install `Visual C++ 2008 Redistributables`.
3. Download and install `Win32 OpenSSL v0.9.8k Light`.

Generating OpenSSL Server Key and Certificate

(Note that you can use FileZilla utility to generate a private key and a certificate by clicking the **Generate new certificate**... button in FileZilla and skip the steps in this section. Make sure the Common Name you enter is the fully qualified host name or IP of the machine on which the FileZilla FTP server is running. You still need to perform the steps in Converting the Server Key From PEM to PKCS12 Format However, you will use the FileZilla generated private key and certificate files.)

To create the server key and certificate files, you must perform the following steps:

1. Open the command prompt and browse to the `OpenSSL\bin` directory.
2. Run the following command:

   ```
   openssl req -new -x509 -keyout mykey.pem -out mycert.pem -days 365
   ```

   A sample command output is as follows:

   ```
   C:\OpenSSL\bin>openssl req -new -x509 -keyout mykey.pem       -out mycert.pem -
   days 365
   Loading 'screen' into random state - done
   Generating a 1024 bit RSA private key
   ..........++++++
   .......++++++
   writing new private key to 'mykey.pem'
   Enter PEM pass phrase:
   Verifying - Enter PEM pass phrase:
   -----
   ```

```
You are about to be asked to enter information that
       will be incorporated into your certificate request.
What you are about to enter is what is called a
                      Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:CA
Locality Name (eg, city) []:Belmont
Organization Name (eg, company)
                [Internet Widgits Pty Ltd]:Test
Organizational Unit Name (eg, section) []:Test
Common Name (eg, YOUR name) []:Test test
Email Address []:test@test.com
```

3. Enter a PEM pass phrase when prompted.
4. Re-enter PEM pass phrase entered in step 3 for verification.
5. Enter the requested details.

   The server key (`mykey.pem`) and certificate (`mycert.pem`) are generated in the `OpenSSL\bin` directory.

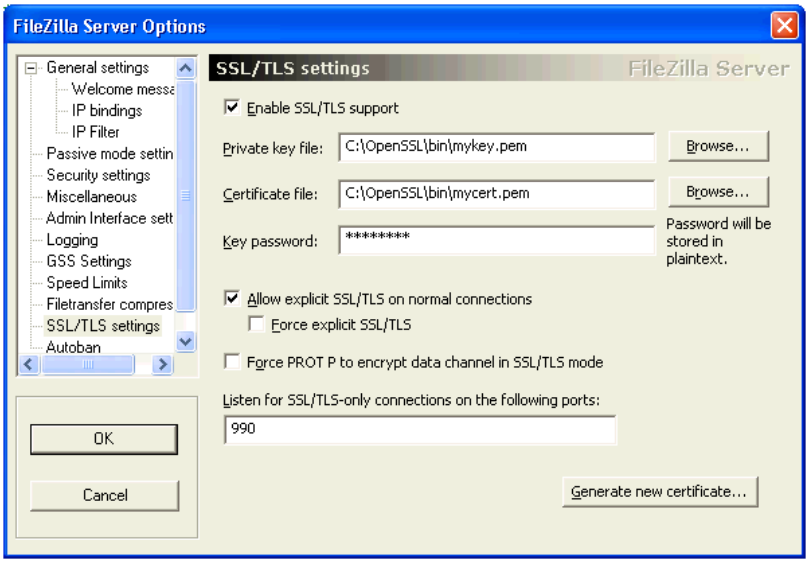
Importing the Server Key and Certificate Into FileZilla Server

To import the server key and certificate into FileZilla, you must perform the following steps:

1. Open a FileZilla Server interface from your Windows Start menu.
2. Click **Edit**, and then click **Settings**.

   The FileZilla **Server Options** dialog is displayed.
3. Click **SSL/TLS settings**.
4. Enter the server key and certificate details as shown in Figure 4-52.

Figure 4-52 The FileZilla Server Options Dialog



Note:

In the **Key password** field, you must use the PEM pass phrase generated in Step 3 of Generating OpenSSL Server Key and Certificate.

Converting the Server Key From PEM to PKCS12 Format

You must convert the server key and the server certificate from the PEM format to the PKCS#12 format as the Oracle FTP Adapter does not recognize the PEM format. To convert the server key and certificate to the PKCS#12 format, you must perform the following steps:

1. Open the command prompt and browse to the `OpenSSL\bin` directory.
2. Run the following command:

   ```
   openssl pkcs12 -export -out mykeyz.p12              -in mycert.pem  -inkey
   mykey.pem
   ```

   The command output is as follows:

   ```
   C:\OpenSSL\bin>openssl pkcs12           -export -out mykeyz.p12 -in mycert.pem  -
   inkey mykey.pem
   Loading 'screen' into random state - done
   Enter pass phrase for mykey.pem:
   Enter Export Password:
   Verifying - Enter Export Password:
   ```

3. Enter a PEM pass phrase when prompted. This is the pass phrase that you created while generating OpenSSL server key and certificate in Generating OpenSSL Server Key and Certificate.
4. Enter an export password for the PKCS#12 file.
5. Re-enter the export password for verification.
6. Enter the requested details.

The `mykeyz.p12` file is generated in the `OpenSSL\bin` directory.

7. Copy the `mykeyz.p12` file to the managed Oracle WebLogic Server instance running the Oracle FTP Adapter.

   For example,

   ```
   /scratch/$user/private/mykeyz.p12
   ```

Configuring Oracle FTP Adapter Deployment Descriptor to Use the New Key

You must perform the following steps to configure the Oracle FTP Adapter deployment descriptor:

1. Navigate to `http://servername:portnumber/console`.
2. Use the required credentials to open the home page of the Oracle WebLogic Server Administration Console.
3. Select **Deployments** in the Domain Structure pane.

   The Oracle WebLogic Server Administration Console - **Summary of Deployments** page is displayed.

4. Click **FtpAdapter**.

   The Oracle WebLogic Server Administration Console - **Settings for FtpAdapter** page is displayed.

5. Click the **Configuration** tab, and then click the **Outbound Connection Pools** tab.

   The **Outbound Connection Pool Configuration** table is displayed.

6. Select the JNDI name for the Ftp Adapter instance you configure. For example, `"eis/Ftp/FtpAdapter"`.
7. Configure the deployment descriptors listed in Table 4-11:

Table 4-11 JCA Properties for Oracle File and FTP Adapters

Property Name	Property Value
`useFtps`	Set the value to `true`.
`walletLocation`	Set it to the location of the PKCS#12 file in the managed Oracle WebLogic Server instance: `/scratch/$user/private/mykeyz.p12`.
`walletPassword`	Set the value to the export password generated in Step 4 of Converting the Server Key From PEM to PKCS12 Format.
`keyStoreProviderName`	Set the value to `sun.security.provider.Sun`
`keystoreType`	Set the value to `PKCS12`
`keystoreAlgorithm`	Set the value to `SunX509`
`pkiProvider`	Must be left blank.
`jsseProvider`	Must be left blank.

Using SFTP with Oracle FTP Adapter

SSH file transfer protocol (SFTP) is a network protocol that enables secure file transfer over a network. Oracle FTP Adapter supports the use of the SFTP feature on Windows and Linux. This section provides an overview of the SFTP functionality and describes how to install and configure this feature.

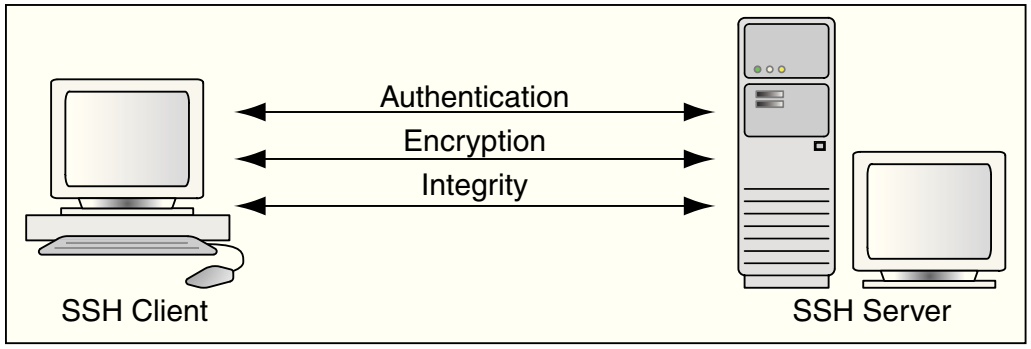
This section includes the following tasks:

- SFTP Overview
- Install and Configure OpenSSH for Windows
- Set Up for SFTP

SFTP Overview

FTP is the network protocol that enables clients to securely transfer files over the underlying SSH transport. SFTP is not similar to FTP over SSH or File Transfer Protocol (FTP). Figure 4-53 displays the communication process between an SSH client and an SSH server. SFTP is supported in Windows and Linux.

Figure 4-53 SFTP Communication



SFTP has the following features:

- Encryption
- Authentication
- Integrity
- Data Compression

Encryption

The SSH protocol uses public key cryptography for encryption. This section explains how data is encrypted:

1. The SSH subsystem uses symmetric key ciphers such as Data Encryption Standard (DES) or Blowfish to generate a session key. The SSH protocol currently uses the Diffie-Hellman Key Exchange Algorithm to derive the symmetric key for the session.
2. The data is encrypted using the session key.

3. The session key is encrypted by using the recipient's public key. Because the recipient has the private key, it can decrypt the message by using its preferred PKI algorithm such as Rivest-Shamir-Adleman (RSA) or Digital Signature Algorithm (DSA).

Authentication

The SSH protocol inherently supports password authentication by encrypting passwords or session keys as they are transferred over the network. In addition, the SSH protocol uses a mechanism known as 'known hosts' to prevent threats such as IP spoofing. When this mechanism is used, both the client and the server have to prove their identity to each other before any kind of communication exchange.

Integrity

The SSH protocol uses widely trusted bulk hashing algorithms such as Message Digest Algorithm 5 (MD5) or Secure Hash Algorithm (SHA-1) to prevent insertion attacks. Implementation of data integrity checksum by using the algorithms mentioned in Encryption prevents deliberate tampering of data during transmission.

Data Compression

The SSH protocol supports zlib, an open-source cross-platform algorithm for data compression. SSH uses `zlib` to compress in-flight data to reduce network bandwidth.

Install and Configure OpenSSH for Windows

OpenSSH for Windows is the free implementation of the SSH protocol on Windows. Perform the following steps to install and configure OpenSSH on Windows XP:

1. Log in as a user with Administrator privileges.
2. Download `setup.exe` from the following location:

   `http://www.cygwin.com`
3. Run `setup.exe`. The Cygwin Net Release Setup window is displayed.
4. Click **Next**. The **Choose Installation** type window is displayed.
5. Select **Install from Internet** as the download source and click **Next**. The Choose Installation Directory window is displayed.
6. Leave the root directory as `C:\cygwin`. Also, keep the default options for the **Install For** and the **Default Text File Type** fields.
7. Click **Next**. The **Select Local Package Directory** window is displayed.
8. Click **Browse** and select `C:\cygwin` as the local package directory.
9. Click **Next**. The **Select Connection Type** window is displayed.
10. Select a setting for Internet connection and click **Next**. The **Choose Download Site(s)** window is displayed.
11. Select a site from the **Available Download Sites** list and click **Next**. The **Select Packages** window is displayed.
12. Click **View** to see the complete list of packages available for installation.

13. Select **openssh** if it is not the default value.
14. Select the **Binaries** box for openssh.
15. Click **Next** to start the installation.
16. On Windows XP desktop, right -click **My Computer** and select **Properties**.
17. Click the **Advanced** tab and click **Environment Variables**.
18. Click **New** and enter `CYGWIN` in the **Variable Name** field and `ntsec` in the **Variable Value** field.
19. Add `C:\cygwin\bin` to the system path.
20. Open the cygwin window.
21. Type `ssh-host-config`.
22. You are prompted with the following questions:
    a. `Shall privilege separation be used? (yes/no)`

       Enter `yes`.

    b. `Shall this script create a local user 'sshd' on this machine?`

       Enter `yes`.

    c. `Do you want to install sshd as service?`

       `(Say "no" if it's already installed as service) (yes/no)`

       Enter `yes`.

    d. `Which value should the environment variable CYGWIN have when sshd starts? It's recommended to set at least "ntsec" to be able to change user context without password. Default is "binmode ntsec tty".`

       Enter `ntsec`.
23. Type `net start sshd` to start the sshd service.
24. Run the following command in the cygwin window to replicate the Windows local user accounts to cygwin:

```
mkpasswd --local > /etc/passwd
mkgroup --local > /etc/group
```

25. To test the setup, type `ssh localhost` in the cygwin window.

Set Up Oracle FTP Adapter for SFTP

To use the SFTP functionality, you must modify the deployment descriptor for Oracle FTP Adapter.

Table 4-12 lists the properties for which you must specify a value in the deployment descriptor. The values of these properties depend on the type of authentication and the location of OpenSSH.

Table 4-12 SFTP Properties

Property	Description
`useSftp`	Specify `true`. **Mandatory**: Yes **Default value**: `false`
`authenticationType`	Specify `PASSWORD` for password-based authentication or `PUBLICKEY` for public key authentication. For password-based authentication, the user name and password specified in the `weblogic-ra.xml` file are used. Ensure that there is a Windows user with the same name and password as specified in the `weblogic-ra.xml` file. In addition, the user should have administrative privileges. For public key authentication, the `privateKeyFile` parameter must be set to the location of the private key file. **Mandatory**: Yes
`preferredKey ExchangeAlgorithm`	Specify `diffie-hellman-group1-sha1` or `diffie-hellman-group-exchange-sha1`. This is an optional parameter where the user can select the default key exchange protocol for negotiating the session key for encrypting the message. **Mandatory**: No **Default value**: `diffie-hellman-group1-sha1`
`preferred CompressionAlgorithm`	Specify `none` or `zlib`. This parameter enables the user to choose whether in-flight data should be compressed or not. **Mandatory**: No
`preferred DataIntegrityAlgorithm`	Specify `hmac-md5` or `hmac-sha1`. This parameter enables the user to select the bulk-hashing algorithm for data integrity checks. **Mandatory**: No **Default value**: `hmac-md5`
`preferredPKIAlgorithm`	Specify `ssh-rsa` or `ssh-dsa`. This parameter enables the user to configure the asymmetric cipher for the communication. **Mandatory**: No **Default value**: `ssh-rsa`
`privateKeyFile`	Specify the path to the private key file. This is required if the `authenticationType` parameter is set to `PUBLICKEY`. **Mandatory**: No

Table 4-12 (Cont.) SFTP Properties

Property	Description
`preferredCipherSuite`	Specify a cipher from the following list: • twofish192-cbc • cast128-cbc • twofish256-cbc • aes128-cbc • twofish128-cbc • 3des-cbc • blowfish-cbc • aes256-cbc • aes192-cbc **Mandatory**: No **Default value**: blowfish-cbc
`transportProvider`	Specify `socket` or `HTTP`. Specify `socket` if the SSH server is inside a firewall. Specify `HTTP` if the SSH server is outside the firewall or a server is exposed through an HTTP server. If you select HTTP, then you must provide values for the following parameters: • `proxyHost` • `proxyPort` • `proxyUser` • `proxyPassword` • `useProxy` **Mandatory**: Yes

Configuring Oracle FTP Adapter for Password Authentication

To set up the Oracle FTP Adapter for password authentication, the deployment descriptor for Oracle FTP Adapter must specify the values of the properties listed in Table 4-12. Ensure that the `authenticationType` property is set to `password`.

Specify the following properties and values listed in Table 4-13:

Table 4-13 Sample SFTP Properties and Values

Property	Value
`useSftp`	`true`
`authenticationType`	`PASSWORD`
`preferredKey ExchangeAlgorithm`	`diffie-hellman-group1-sha1`
`preferred CompressionAlgorithm`	`none`
`preferred DataIntegrityAlgorithm`	`hmac-md5`
`preferredPKIAlgorithm`	`ssh-rsa`

Table 4-13 (Cont.) Sample SFTP Properties and Values

Property	Value
privateKeyFile	-
preferredCipherSuite	blowfish-cbc
transportProvider	socket

Configuring Oracle FTP Adapter for Public Key Authentication

For public key authentication, you must first configure OpenSSH and then set up the Oracle FTP Adapter.

The Oracle FTP Adapter setup depends if the OpenSSH is running inside a firewall or outside a firewall.

If OpenSSH is running inside the firewall, then see the following sections:

- Configuring OpenSSH for Public-Key Authentication
- Configuring for Public Key Authentication with OpenSSH Running Inside a Firewall

If OpenSSH is running outside the firewall, then see the following sections:

- Configuring OpenSSH for Public-Key Authentication
- Configuring for Public Key Authentication with OpenSSH Running Outside a Firewall

Configuring OpenSSH for Public-Key Authentication

Perform the following steps:

1. Go to the `C:\cygwin\etc` directory. If required, configure the `sshd_config` file to force public key authentication. For more information, see openssh help or manual.
2. Go to the `C:\cygwin\bin` directory.
3. Run the following command to generate the key pair:

   ```
   ssh-keygen -t rsa
   ```

4. Enter `/etc/id_rsa` when prompted for the file in which the key should be saved.
5. Enter the passphrase.
6. Enter the passphrase again.
7. Go to the `/etc` directory and verify that both the public key file (`id_rsa.pub`) and the private key file (`id_rsa`) are generated.
8. Run the following command to create a copy of the public key file:

   ```
   cp id_rsa.pub authorized_keys
   ```

9. Create a copy of the private key file in a secured location such as `C:\my-secured-folder\`. The Oracle FTP Adapter configuration refers to this private key file.

10. Restart the OpenSSH server by running the following commands:

```
net stop sshd
net start sshd
```

Configuring Oracle FTP Adapter for Public Key Authentication with OpenSSH Running Inside a Firewall

To set up the Oracle FTP Adapter for public key authentication, you must specify the values of the parameters listed in Table 4-12 in the deployment descriptor. Ensure that the `authenticationType` parameter is set to `publickey` and the `transportProvider` parameter is set to `socket`. The `privateKeyFile` parameters should contain the location of the private key file.

A sample list of public key authentication properties and their values is shown in Table 4-14.

Table 4-14 Sample SFTP Properties and Values

Property	Value
`useSftp`	`true`
`authenticationType`	`publickey`
`preferredKeyExchangeAlgorithm`	`diffie-hellman-group1-sha1`
`preferredCompressionAlgorithm`	`none`
`preferredDataIntegrityAlgorithm`	`hmac-md5`
`preferredPKIAlgorithm`	`ssh-rsa`
`privateKeyFile`	`C:\my-secured-folder\id_rsa`
`preferredCipherSuite`	`blowfish-cbc`
`transportProvider`	`socket`

Configuring Oracle FTP Adapter for Public Key Authentication with OpenSSH Running Outside a Firewall

Perform the following steps to set up the Oracle FTP Adapter for public key authentication when OpenSSH is running outside the firewall:

1. In the deployment descriptor for Oracle FTP Adapter, you must specify the values of the properties listed in Table 4-12 in the deployment descriptor for Oracle FTP Adapter. Ensure that the `authenticationType` property is set to `publickey` and the `transportProvider` property is set to `HTTP`. The `privateKeyFile` property contains the location of the private key file.

2. In the deployment descriptor for Oracle FTP Adapter, also specify the following proxy-related properties:

- `proxyHost`: The name of the proxy host.
- `proxyPort`: The port number of the proxy.
- `proxyUsername`: The user name for the proxy.
- `proxyPassword`: The password for the proxy.
- `useProxy`: Specify `true` to use proxy.

A sample list with public key authentication properties and proxy properties is shown in Table 4-15.

Table 4-15 Sample SFTP Properties and Values

Property	Value
proxyHost	proxy.host.com
proxyPort	80
proxyUsername	anonymous
proxyPassword	tiger@scott.com
useProxy	true
useSftp	true
authenticationType	publickey
preferredKey ExchangeAlgorithm	diffie-hellman-group1-sha1
preferred CompressionAlgorithm	none
preferred DataIntegrityAlgorithm	hmac-md5
preferredPKIAlgorithm	ssh-rsa
privateKeyFile	C:\my-secured-folder\id_rsa
preferredCipherSuite	blowfish-cbc
transportProvider	HTTP

Configuring Oracle FTP Adapter for HTTP Proxy

The Oracle FTP Adapter provides proxy support for HTTP proxy only. The HTTP proxy support is available in the following two modes, plain FTP mode and SFTP mode. This section explains how to configure the Oracle FTP Adapter for running in plain FTP mode and SFTP mode. It contains following sections:

- Configuring for Plain FTP Mode
- Configuring for SFTP Mode

Configuring for Plain FTP Mode

For running the Oracle FTP Adapter in plain FTP mode, you must specify the value of certain parameters in the Oracle FTP Adapter deployment descriptor. Table 4-16 lists the properties that you must modify.

Table 4-16 Plain FTP Mode Properties

Property	Description
`host`	The remote FTP server name.
`port`	The FTP control port number.
`username`	The FTP user name.
`password`	The FTP password.
`proxyHost`	The proxy host name.
`proxyPort`	The proxy port number.
`proxyUsername`	The proxy user name.
`proxyPassword`	The proxy password.
`proxyType`	The proxy type. Only HTTP proxy type is supported.
`proxyDefinitionFile`	The absolute path of the proxy definition file. This parameter is not mandatary. See Proxy Definition File for more information.
`useProxy`	Specify `true` to use proxy.

A sample list of Oracle FTP Adapter descriptor properties and their values is shown in Table 4-17.

Table 4-17 Sample Plain FTP Mode Properties and Values

Property	Value
`host`	`my.host.com`
`port`	`21`
`username`	`user`
`password`	`password`
`proxyHost`	`proxy.host.com`
`proxyPort`	`80`
`proxyUsername`	`anonymous`
`proxyPassword`	`tiger@scott.com`
`proxyType`	`http`

Table 4-17 (Cont.) Sample Plain FTP Mode Properties and Values

Property	Value
proxyDefinitionFile	c:\proxydefinitions.xml
useProxy	true

Proxy Definition File

You can specify all proxy-specific information in a proxy definition file and configure the adapter to use this file with the `proxyDefinitionFile` property of the Oracle FTP Adapter deployment descriptor file. A proxy definition file is written in XML format and is based on XML schema. The XML schema for the proxy definition file is shown in example below. Your proxy definition file must be based on this XML schema.

Example - Proxy Definition File XML Schema

```
<?xml version = \"1.0\" encoding = \"UTF-8\"?>
<schema targetNamespace = "http://ns.oracle.com/ip
                /af/ftp/proxy" xmlns =
"http://www.w3.org/2001/XMLSchema"
xmlns:proxy="http://ns.oracle.com/ip/af/ftp/proxy">

         <element name="ProxyDefinitions" type="proxy:
                     ProxyDefinitionsType"/>
              <complexType name="ProxyDefinitionsType">
                      <sequence>
                           <element name="Proxy"
                                  type="proxy:ProxyDefinition"
                            minOccurs="0"
                                maxOccurs="unbounded"/>
                      </sequence>
              </complexType>

              <complexType name="ProxyDefinition">
                      <sequence>
                           <element name="Step"
                                       type="proxy:StepType"
                            minOccurs="1"
                                maxOccurs="unbounded"/>
                      </sequence>
                      <attribute name="key" type="ID"
                                        use="required"/>
                      <attribute name="description"
                                        type="string"
                       use="required"/>
                      <attribute name="type"
                           type="proxy:Protocol"
                        use="optional"/>
              </complexType>

              <complexType name="StepType">
                  <simpleContent>
                     <extension base="string">
                         <attribute name="command"
                                type="string"
                                    use="required"/>
```

```
                        <attribute name="args" type="string"
                                   use="required"/>
                    </extension>
                  </simpleContent>
               </complexType>

                <simpleType name="Protocol">
                     <restriction base="string">
                           <enumeration value="ftp" />
                           <enumeration value="http" />
                     </restriction>
                </simpleType>
</schema>
```

A sample proxy definition file, based on the XML schema in **Example - Proxy Definition File XML Schema**, would look as shown in the example below.

Example - Proxy Definition File

```
<?xml version = '1.0' standalone = 'yes'?>
<proxy:ProxyDefinitions xmlns:proxy=
             "http://ns.oracle.com/ip/af/ftp/proxy">
<Proxy key="http" description="http" type="http">
<Step command="USER" args="remote_username" />
<Step command="PASS" args="remote_password" />
</Proxy>
</proxy:ProxyDefinitions>
```

When you use the file in **Example - Proxy Definition File**, the Oracle FTP Adapter sends the following sequence of commands to log in:

1. USER remote_username
2. PASS remote_password

You can also direct the proxy definition file to pick values from the deployment descriptor for Oracle FTP Adapter. You can use the following expressions for this:

- `$proxy.user`: This corresponds to the value of the `proxyUsername` parameter in the Oracle FTP Adapter deployment descriptor.
- `$proxy.pass`: This corresponds to the value of the `proxyPassword` parameter in the Oracle FTP Adapter deployment descriptor.
- `$remote.user`: This corresponds to the value of the `username` parameter in the Oracle FTP Adapter deployment descriptor.
- `$remote.pass`: This corresponds to the value of the `password` parameter in the Oracle FTP Adapter deployment descriptor.
- `$remote.host`: This corresponds to the value of the `host` parameter in the Oracle FTP Adapter deployment descriptor.
- `$remote.port`: This corresponds to the value of the `port` parameter in the Oracle FTP Adapter deployment descriptor.

A sample proxy definition file based on the XML schema in **Example - Proxy Definition File** and taking values from the `weblogic-ra.xml` file is shown in the example below.

Example - Proxy Definition File Taking Values from the Deployment Descriptor

```
<?xml version = '1.0' standalone = 'yes'?>
<proxy:ProxyDefinitions xmlns:proxy=
           "http://ns.oracle.com/ip/af/ftp/proxy">
<Proxy key="http" description="http" type="http">
<Step command="USER" args="$remote.user" />
<Step command="PASS" args="$remote.pass" />
</Proxy>
</proxy:ProxyDefinitions>
```

Configuring for SFTP Mode

For running the Oracle FTP Adapter in SFTP mode, you must specify the value of certain properties in the Oracle FTP Adapter deployment descriptor. Table 4-18 lists the properties that you must modify.

Table 4-18 SFTP Mode Properties

Property	Description
`host`	The remote FTP server name.
`port`	The FTP control port number.
`username`	The SFTP user name.
`password`	The SFTP password.
`proxyHost`	The proxy server host name.
`proxyPort`	The proxy port number.
`proxyUsername`	The proxy user name.
`proxyPassword`	The proxy password.
`useSftp`	Specify `true` for SFTP mode. This value is required to use the SFTP feature.
`authenticationType`	Specify either `PASSWORD` or `PUBLICKEY.PASSWORD` See Set Up for SFTP
`transportProvider`	Specify `http` as value. Only HTTP transport provider is supported.

A sample list of deployment descriptor properties is shown in Table 4-19.

Table 4-19 Sample SFTP Mode Properties and Values

Property	Value
`host`	`my.host.com`
`port`	`22`
`username`	`user`
`password`	`password`
`proxyHost`	`proxy.host.com`
`proxyPort`	`80`

Table 4-19 (Cont.) Sample SFTP Mode Properties and Values

Property	Value
proxyUsername	anonymous
proxyPassword	password
useSFTP	true
authenticationType	password
transportProvider	http

Configuring File and FTP Adapters for High Availability

You can configure the File and FTP Adapters for high availability within the active-active topology for both SOA and OSB. You can configure high availability for both inbound and outbound operations.

Inbound Operations

Oracle File and FTP Adapters ensure that only one node processes a specific file in a distributed topology. To enable HA, the File or FTP adapter can either use database table or coherence cache as a coordinator to ensure that Oracle File and FTP Adapters are highly available for inbound operation.

To enable High Availability using a database table, you must indicate `eis/HAFIleAdapter` in your .jca file. This indicates use of an Oracle table for high availability or one of the other variants such as `eis/HAFileAdapterMQSSQL` or `eis/HAFileAdapterDB2` for SQL Server and DB2 tables respectively.

In addition to the database-based coordinator, the File and FTP adapters support Coherence for HA capabilities. To do so, you can indicate `eis/CoherenceHAFileAdapter` or `eis/Ftp/CoherenceHAFtpAdapter` for File and FTP adapters respectively.

In the following .jca file for a File Adapter, the connection factory has been specified as `CoherenceHAFileAdapter:`

Example - .jca File for File Adapter with CoherenceHAFileAdapter Specified as Connection Factory

```
<adapter-config name="FileHACoherenceIn"
                  adapter="File Adapter"
    wsdlLocation="FileHACoherenceIn.wsdl"
    xmlns="http://platform.integration.
           oracle/blocks/adapter/fw/metadata">
  <connection-factory location=
                 "eis/CoherenceHAFileAdapter"
   UIincludeWildcard="*.zip" adapterRef=""/>
  <endpoint-activation portType="Read_ptt"
                  operation="Read">
    <activation-spec className=
                    "oracle.tip.adapter.file.
                                  inbound.
                    FileActivationSpec">
      <[..]>
      <[..]>
    </activation-spec>
```

```
  </endpoint-activation>

</adapter-config>
```

Outbound Operations

The Oracle File and FTP Adapters ensure that if multiple references write to the same directory, they do not overwrite each other.

The adapter uses a database table for serializing access to concurrent writes to the same file in the folder. The procedure is similar to inbound; you must select `eis/HAFileAdapter` which uses an Oracle table or one of the other variants such as `eis/HAFileAdapterMQSSQL`, or `eis/HAFileAdapterDB2` for SQL Server and DB2 respectively.

As with inbound File and FTP Adapter high availability operations, the File and FTP adapters performing outbound support Coherence for High Availability capabilities. From a design experience, you must indicate `eis/CoherenceHAFileAdapter` or `eis/Ftp/CoherenceHAFtpAdapter` in your .jca file for File and FTP adapters respectively.

Additional Considerations

The File/FTP Adapter supports High Availability for both inbound as well as outbound using Coherence as a locking mechanism.

Inbound Operations

For inbound processing, the File/FTP Adapter must lock the file before an attempt to process the file (for example, to perform file translation and publish to the fabric). If the lock is already held for the same file by another node, the adapter ignores that file and continues with the next file.

To ensure that one particular node does not monopolize the distribution of files by acquiring locks on all the files, you must configure `MaxRaiseSize` to some finite value for inbound processing. See the example below.

Example - .jca File with MaxRaiseSize Configured to a Finite Value

```
<adapter-config name="FileHACoherenceIn" adapter="File Adapter"
wsdlLocation="FileHACoherenceIn.wsdl" xmlns="http://platform.integration.oracle/
blocks/
            adapter/fw/metadata">

  <connection-factory location="eis/CoherenceHAFileAdapter"
UIincludeWildcard="*.zip" adapterRef=""/>
  <endpoint-activation portType="Read_ptt" operation="Read">
    <activation-spec className="oracle.tip.adapter.file.inbound.FileActivationSpec">
      <property name="MaxRaiseSize" value="100"/>
      <[..]>
      <[..]>
    </activation-spec>
  </endpoint-activation>

</adapter-config>
```

Additionally, if you have configured `DeleteFile = "false"`(for example, for `Readonly polling`) or when using de-batching, you must configure a valid shared location so that all the nodes in the cluster see exactly the same path. This is required since in both these cases, the File/FTP Adapter stores some control information in the

shared control directory and if one node goes down during processing, the second node sees exactly the same control information.

To indicate your own Coherence Cache, use one of the following parameters:

- `CoherenceCacheConfig` - the Relative path of the Coherence Cache Configuration
- `InboundCoherenceCacheName` - the Coherence NamedCache used for the inbound File/FTP Adapter
- `OutboundCoherenceCacheName` - Coherence NamedCache used for the outbound File/FTP Adapter.

If you want to create your own cache configuration, you must bundle the cache configuration in a jar file and add it to the server classpath (one quick way is to copy the jar file containing the cache configuration into the `$DOMAIN_HOME/lib` directory and then refer to the configuration in the connection factory). The default values are provided.

Example - Indicating Your Own Coherence Cache

```
<wls:properties>
            <wls:property>
                 <wls:name>CoherenceCacheConfig</wls:name>
                 <wls:value>config/
                     fileadapter-cache-config.xml</wls:value>
            </wls:property>
            <wls:property>
                 <wls:name>InboundCoherenceCacheName</wls:name>
                 <wls:value>FileAdapter-inbound</wls:value>
            </wls:property>
            <wls:property>
                 <wls:name>OutboundCoherenceCacheName</wls:name>
                 <wls:value>FileAdapter-outbound</wls:value>
            </wls:property>
            <wls:property../>
            <wls:property../>
        <wls:properties>
```

Outbound Operations

If you are appending to same file in the cluster, the adapter must obtain an explicit lock on the Named Cache for the filename begin appended to.

Additionally, if you are using batching, you would require the control directory to store batched content before the batching criteria evaluates to force a write to the outbound folder.

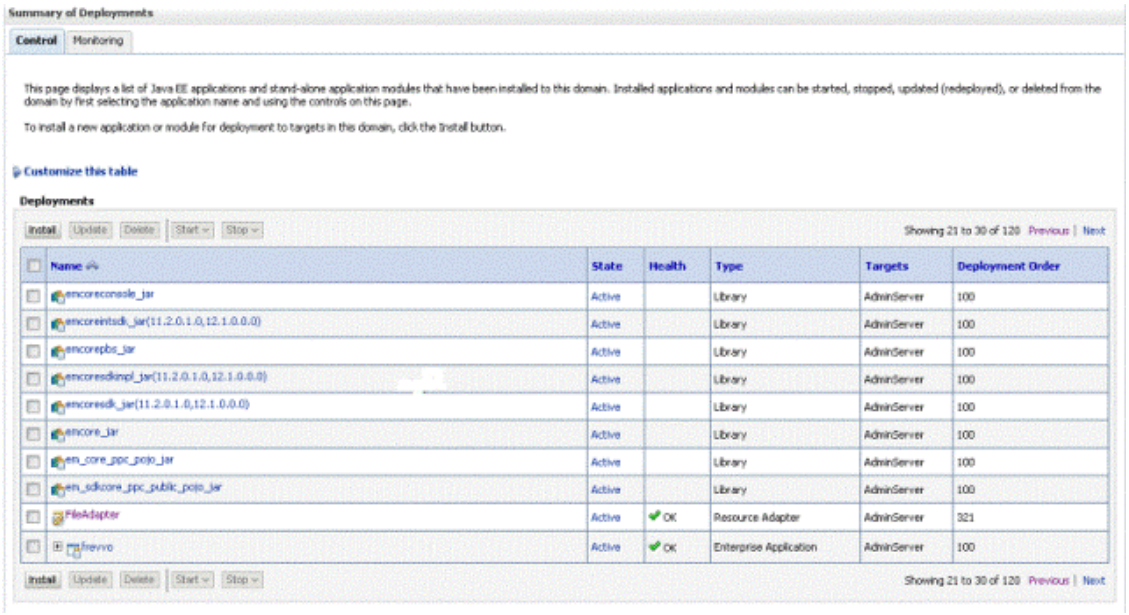
In order to fulfill HA on the outbound, the adapter would require a control directory on a shared file system (which is similar to inbound functioning).

Configuring XA in High-Availability Scenarios

If you want to provide XA transaction capabilities in a high-availability scenario.

1. Login to the WebLogic Server console, select **Deployments > File Adapter**.

Figure 4-54 WebLogic Console Showing Deployments-File Adapter

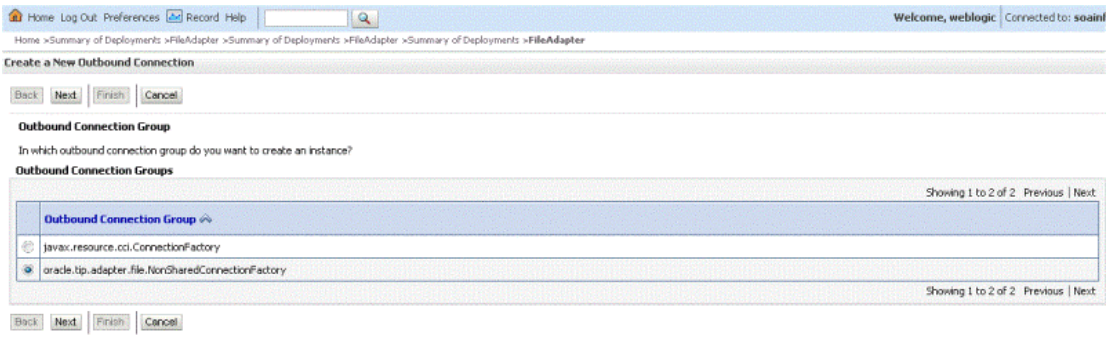


2. Select **Configurations > Outbound Connection Pool**.

Figure 4-55 WebLogic Console Showing Outbound Connection Pool

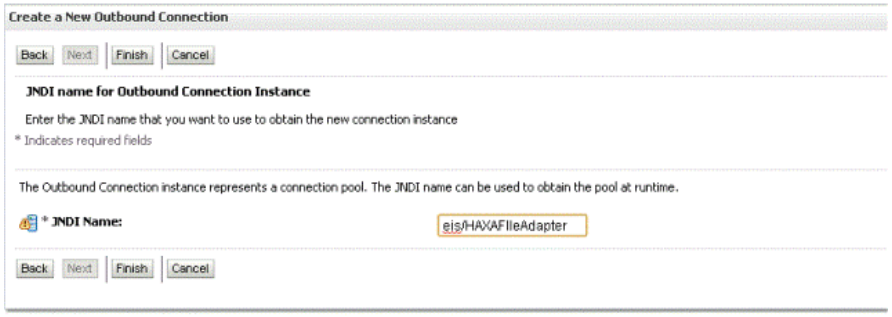
3. Select New. Select `oracle.tip.adapter.file.NonSharedConnectionFactory`.

Figure 4-56 WebLogic Console, File Adapter Deployments with oracle.tip.adapter.file.NonSharedConnectionFactory Selected



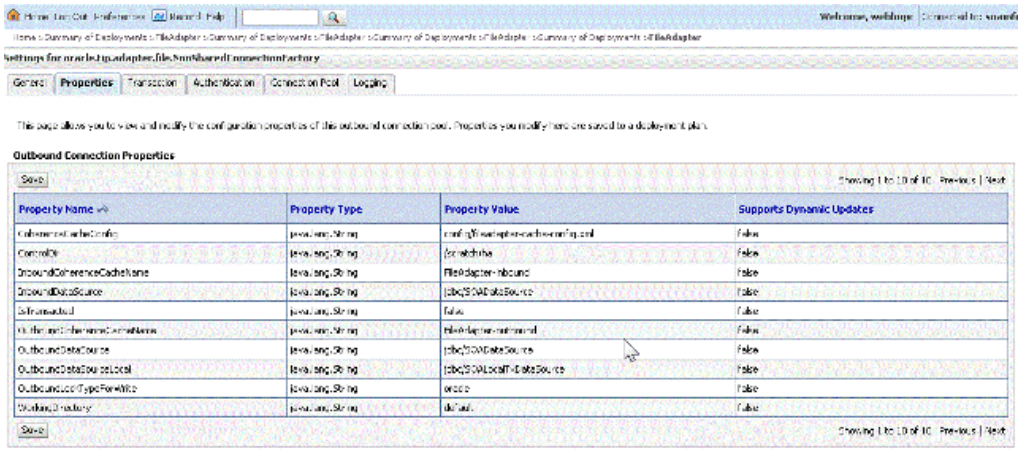
4. Enter the JNDI name that you want to provide for the new connection instance.

Figure 4-57 WebLogic Console Showing the JNDI Name for the New Instance Provided



5. Select Finish and Save. Re-open the JNDI instance and enter the following Outbound Connection properties, provided in Figure 4-58.

Figure 4-58 WebLogic Console Showing Instance with Outbound Connection Properties Shown



Also, refer to Table 4-20

Table 4-20 Outbound Connection Properties for the JNDI Instance

Property	Property Type	Property Value	Supports Dynamic Indexing
`ConferenceCacheConfig`	java.lang.string	`config/fileadpter-cache-config.xml`	False
`ControlDir`	java.lang.string	`/scratch/ha`	False
`InboundCoherenceCacheName`	java.lang.string	`FileAdapter-inbound`	False
`InboundDataSource`	java.lang.string	`jdbc/SOADataSource`	False
`IsTransacted`	java.lang.string	`false`	False

Table 4-20 (Cont.) Outbound Connection Properties for the JNDI Instance

Property	Property Type	Property Value	Supports Dynamic Indexing
`OutboundCoherenceCacheName`	java.lang.string	`FileAdapter-outbound`	False
`OutboundDataSource`	java.lang.string	`jdbc/SOADataSource`	False
`OutboundDataSourceLocal`	java.lang.string	`jdbc/SOALocalTxDataSource`	False
`OutboundLockTypeForWrite`	java.lang.string	`oracle`	False
`WorkingDirectory`	java.lang.string	`default`	False

6. Select **Transaction tab -->XA Transaction**. Select **Save**.

Figure 4-59 Selecting the Transaction Tab in the WebLogic Console

7. If you are using OSB, make sure you do these additional tasks:

   - Make sure you supply `jdbc/SOALocalTXDataSource` datasource as it is not available by default.
   - `QoS (Once-and-only once)` for Routing options and Pipeline transaction is required for outbound XA
   - `Check Pipeline transaction` is not required for inbound Local Transaction.

Oracle File and FTP Adapters Use Cases

This section includes the following Oracle File and FTP Adapters use cases:

- Oracle File and FTP Adapters XML Debatching
- Flat Structure for Oracle BPEL PM
- Flat Structure for Mediator
- Oracle File Adapter Scalable DOM
- Oracle File Adapter Chunked Read
- Oracle File Adapter Read File As Attachments

- Oracle File Adapter File Listing
- Oracle File Adapter Complex Structure
- Oracle File Adapter Debatching
- Oracle File Adapter Dynamic Synchronous Read
- Copying_ Moving_ and Deleting Files
- Creating a Synchronous BPEL Composite using File Adapter
- Changing the Sequencing Strategy for FILE/FTP Adapter
- Controlling the Order in which Files Are Processed
- Extending FTP Adapter

Oracle File Adapter XML Debatching

This is an Oracle File Adapter feature that debatches large XML documents into smaller individual XML fragments.

In this use case, the Debatching XML process uses the Oracle File Adapter to debatch an XML file containing a batch of employees occurring in the XML file as repeating nodes. The Adapter then processes the nodes and writes separate output files to every individual node.

This use case includes the following sections:

- Prerequisites
- Splitting Large Input XML Document that Contains Repeating Elements
- Designing the SOA Composite
- Creating the Inbound Service
- Creating the Outbound File Adapter Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console)

Prerequisites

To perform debatching, you require the following files from the `artifacts.zip` file contained in the `Adapters-102FileAdapterXMLDebatching` sample:

- `artifacts/input/emps.xml`
- `artifacts/schemas/employees.xsd`

You can obtain the `Adapters-102FileAdapterXMLDebatching` sample by accessing the Oracle SOA Sample Code site.

Splitting Input XML Document that Contains Repeating Element

This section describes the process for splitting an input XML document with repeating elements into smaller documents. This is helpful if you want to use XML debatching with the File or FTP Adapter.

`nxsd:elementDepth` is a schema level annotation that facilitates XML debatching in cases when the repeating element `maxOccurs='unbounded` of the XML structure is not an immediate descendant of the root element. The value of the `nxsd:elementDepth` should comply to the XML structure defined in the XSD.

Example of XML Document with Repeating Elements

```
<root>
<ElementList>
    <element1>
        <element11>11</element11>
        <element12>12</element12>
    </element1>
    <element1>
        <element11>21</element11>
        <element12>22</element12>
    </element1>
    <element1>
        <element11>31</element11>
        <element12>32</element12>
    </element1>
</ElementList>
</root>
```

Procedure for splitting XML Document with Repeating Elements

> **Note:** Apply the correct value during design time; the correct value is the number of levels starting below the root element. Incorrect values cause unpredictable results and invalid XML documents. For example, in the following schema, to get the desired output, `nxsd:elementDepth` must be set to 2 as the repeating element <element1> is located 2 levels beneath the root element <root>.

To produce the required output, modify the XML Schema by adding `xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd" nxsd:elementDepth="2"`

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.examplefileIn.org"
xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd" nxsd:elementDepth="2"
targetNamespace="http://www.examplefileIn.org"
elementFormDefault="qualified">
<xsd:element name="root">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="ElementList">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="element1" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
```

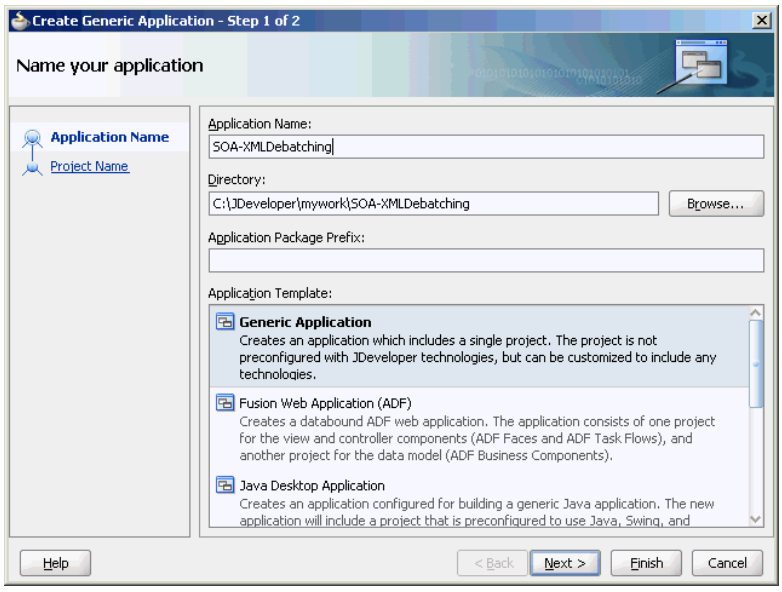
```
<xsd:element name="element11" type="xsd:string"/>
<xsd:element name="element12" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

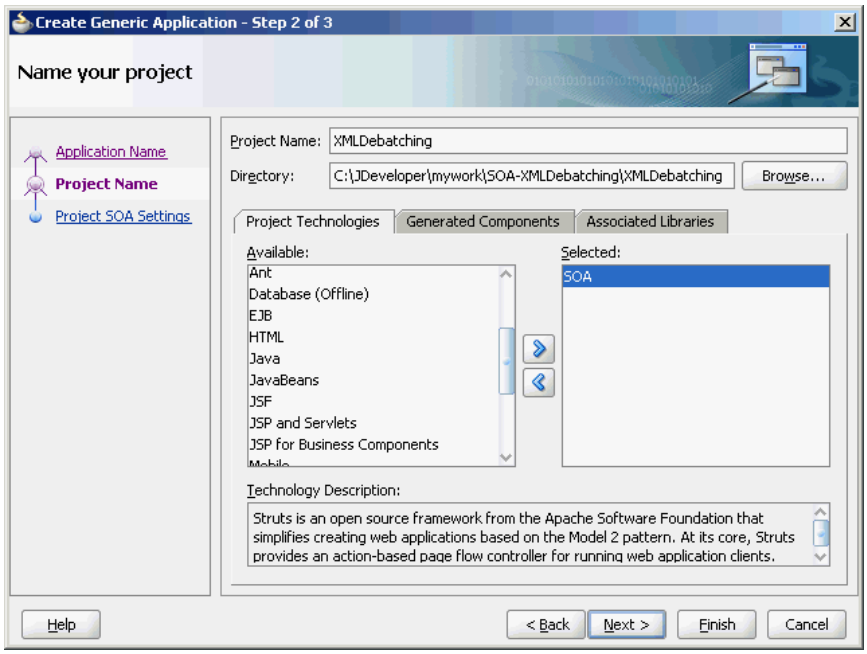
1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `SOA-XMLDebatching` in the **Application Name** field, as shown in Figure 4-60, and click **Next**. The **Create Generic Application - Name your project** page is displayed.

Figure 4-60 The Generic Create Application - Name your application Page



3. Enter `XMLDebatching` in the **Project Name** field.
4. In the **Available** list under the Project Technologies tab, double-click **SOA** to move it to the Selected list, as shown in Figure 4-61.

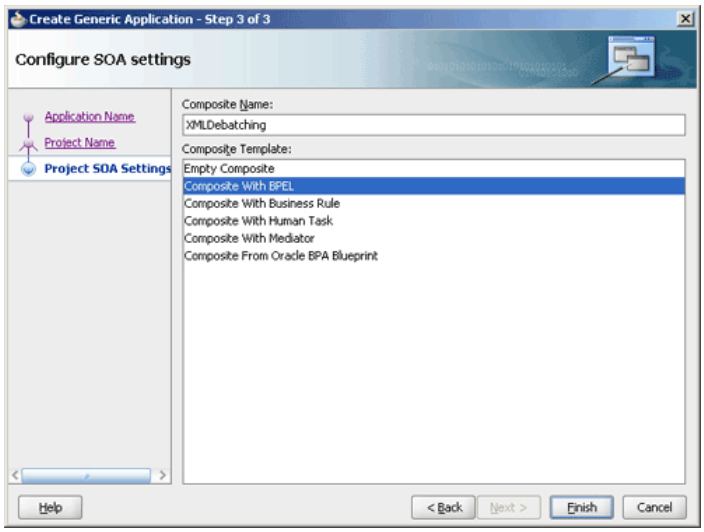
Figure 4-61 The Create Generic Application - Name your project Page



5. Click **Next**. The Configure SOA settings dialog appears.

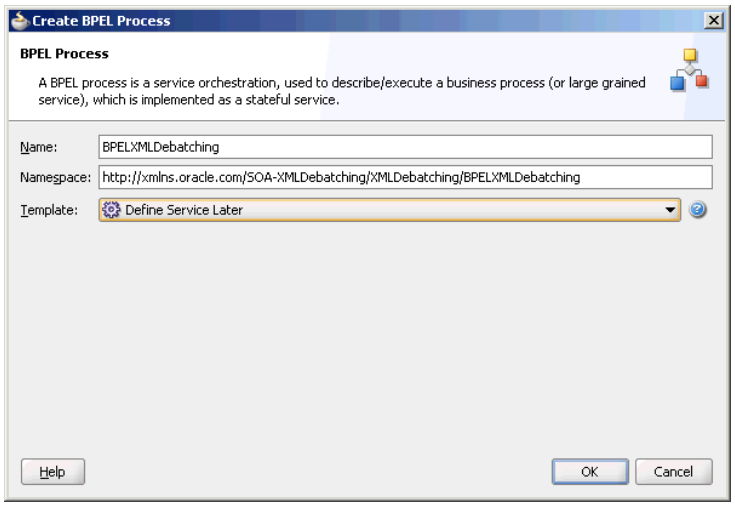
6. Select **Composite With BPEL** in the Composite Template box, as shown in Figure 4-62, and click **Finish**. The Create BPEL Process - BPEL Process page is displayed.

Figure 4-62 The Create Generic Application - Configure SOA settings Page



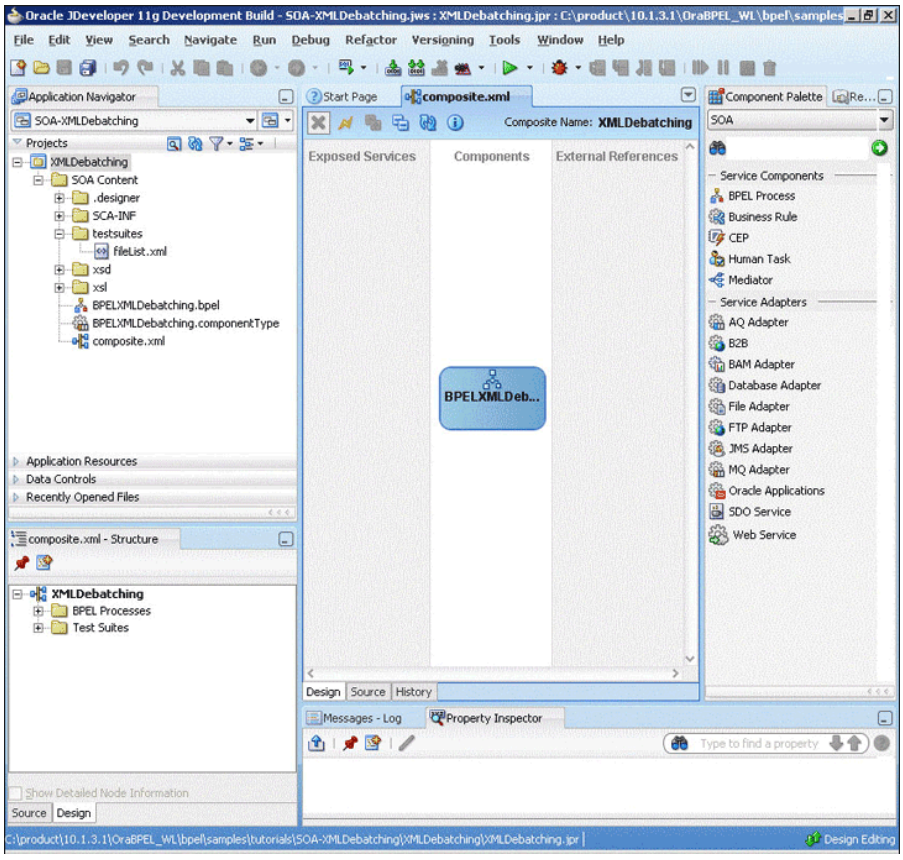
7. Enter `BPELXMLDebatching` in the **Name** field, select **Define Service Later** from the Template box, as shown in Figure 4-63.

Figure 4-63 The Create BPEL Process - BPEL Process Page



8. Click **OK**. The SOA-XMLDebatching application and the XMLDebatching project appear in the design area, as shown in Figure 4-64.

Figure 4-64 The JDeveloper - Composite.xml



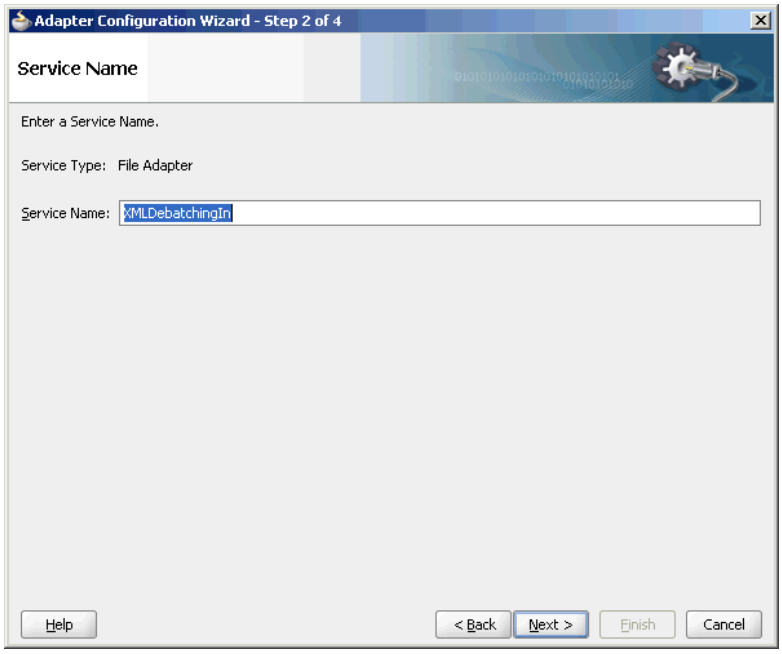
9. Copy the `employees.xsd` file to the xsd directory in your project (see Prerequisites for the location of this file).

Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory:

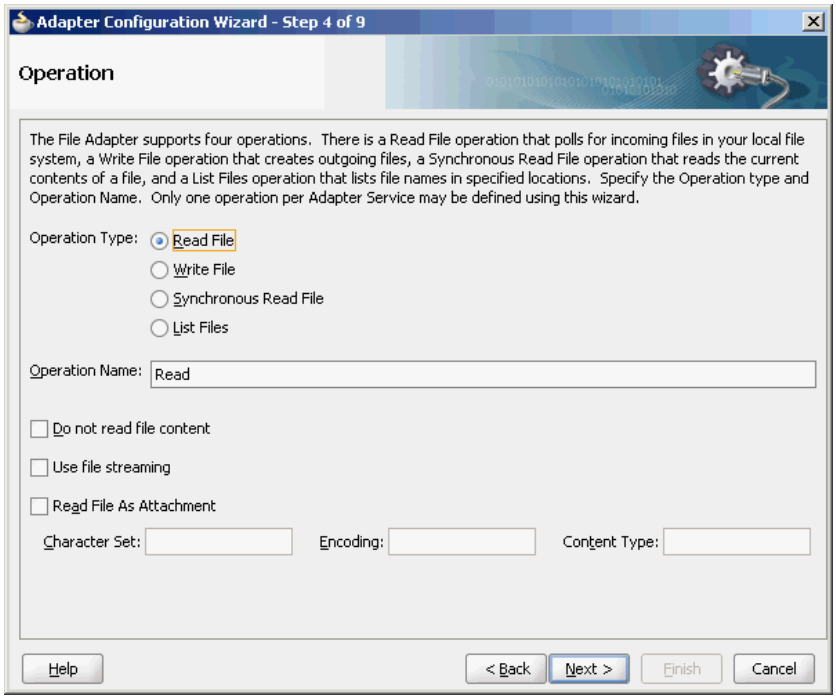
1. Drag and drop the Oracle File Adapter from the Components window to the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `XMLDebatchingIn` in the **Service Name** field and, as shown in Figure 4-65.

Figure 4-65 The Adapter Configuration Wizard - Service Name Page



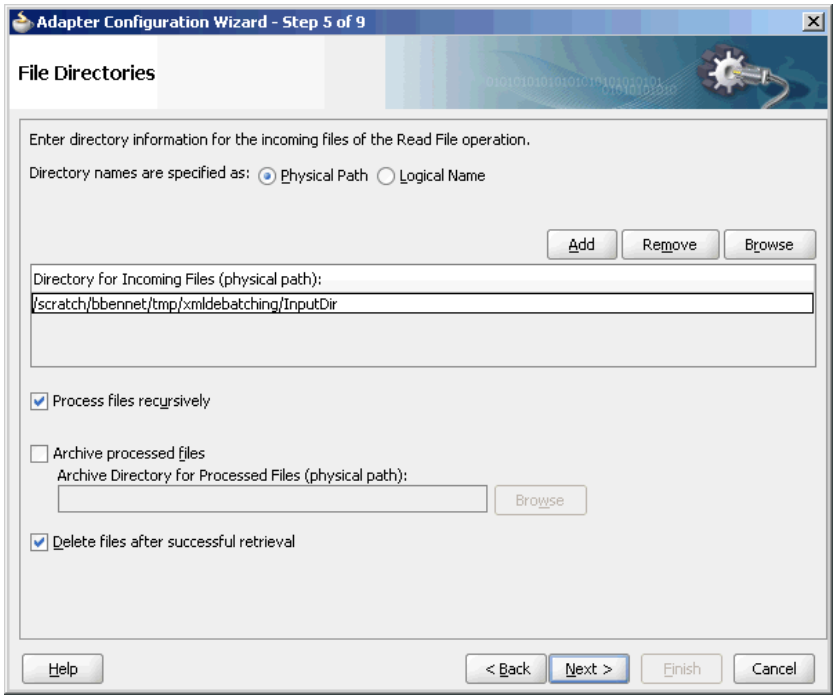
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Read File**, as shown in Figure 4-66, and click **Next**. The File Directories page is displayed.

Figure 4-66 The Adapter Configuration Wizard Operation Page



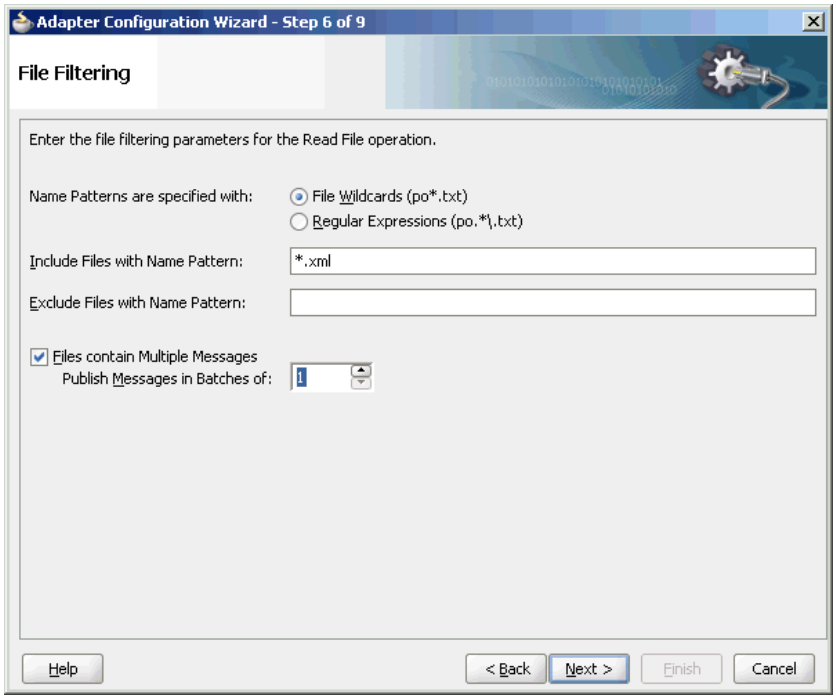
7. Enter the physical path for the input directory, as shown in Figure 4-67. The File Filtering page is displayed.

Figure 4-67 The Adapter Configuration Wizard - File Directories Page



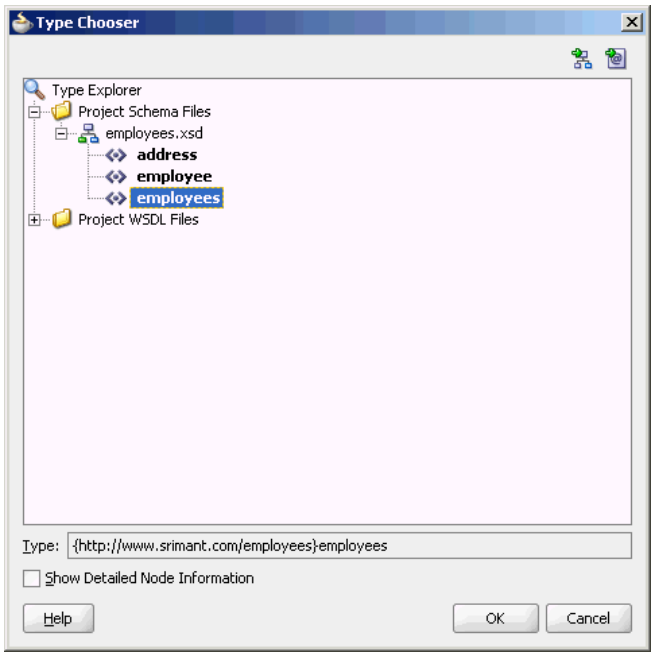
8. Enter `*.xml` in the **Include Files With Name Pattern** field, select **Files Contain Multiple Messages** check box, specify `1` as the value for Publish Messages in Batches Of box, as shown in Figure 4-68.

Figure 4-68 The Adapter Configuration Wizard File Filtering Page



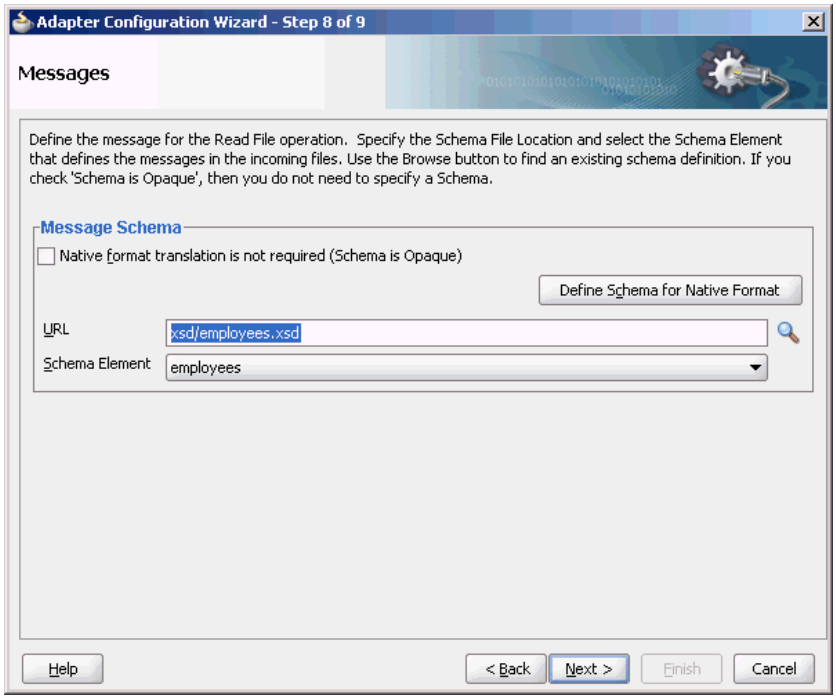
9. Click **Next**. The File Polling page is displayed.
10. Click **Next**. The Messages page is displayed.
11. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
12. Click **Project Schema Files**, **employees.xsd**, and **employees**, as shown in Figure 4-69.

Figure 4-69 The Type Chooser Dialog



13. Click **OK**. The URL field in the Messages page is populated with the **employees.xsd** file, as shown in Figure 4-70.

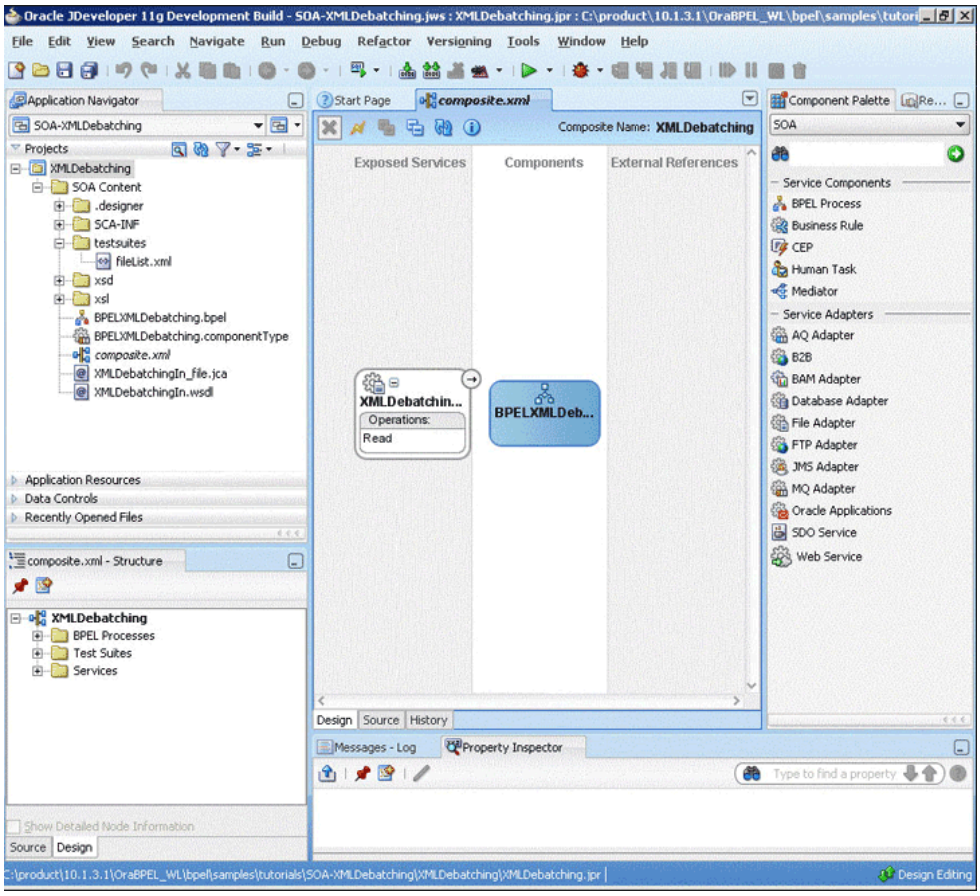
Figure 4-70 The Adapter Configuration Wizard File Messages Page



14. Click **Next**. The **Finish** page is displayed.

15. Click **Finish**. The inbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-71.

Figure 4-71 The JDeveloper - Composite.xml

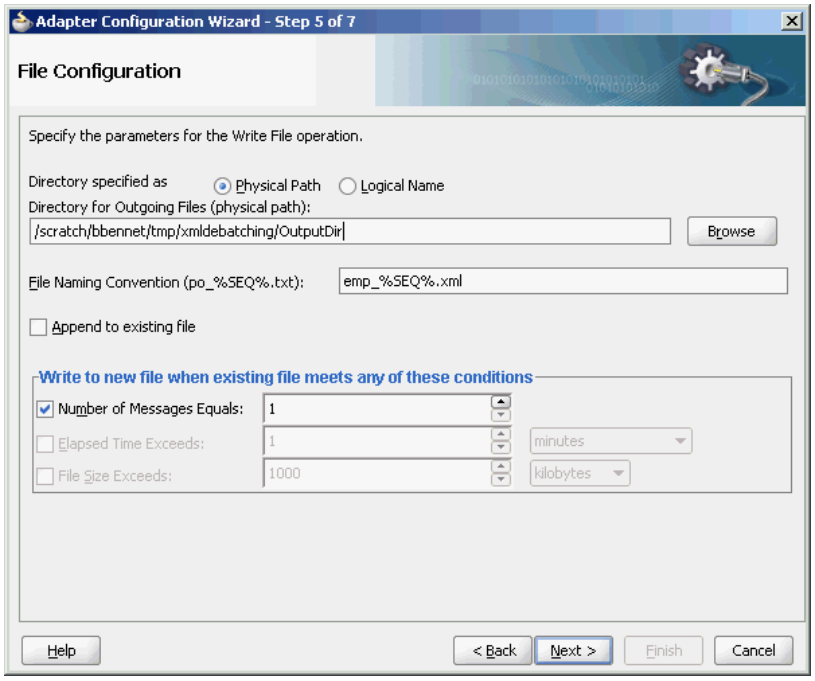


Creating the Outbound File Adapter Service

Perform the following steps to create an outbound file adapter service to write the file from a local directory to the FTP server:

1. Drag and drop the File Adapter from the Components window to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `XMLOut` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, and click **Next**. The File Configuration page is displayed.
7. Enter the physical path for the output directory and enter `emp_%SEQ%.xml` in the **File Naming Convention (po_%SEQ%.txt)** field, as shown in Figure 4-72.
8. Select **Number of Messages Equals** option, if not selected. The default value is `1`.

Figure 4-72 The Adapter Configuration Wizard - File Configuration Page

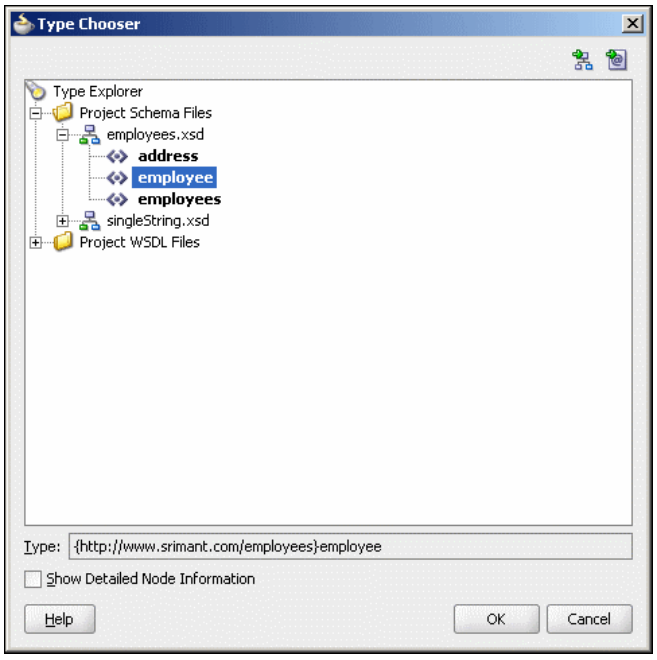


9. Click **Next**. The Messages page is displayed.

10. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.

11. Click **Project Schema Files**, **employees.xsd**, and **employee**, as shown in Figure 4-73.

Figure 4-73 The Type Chooser Dialog

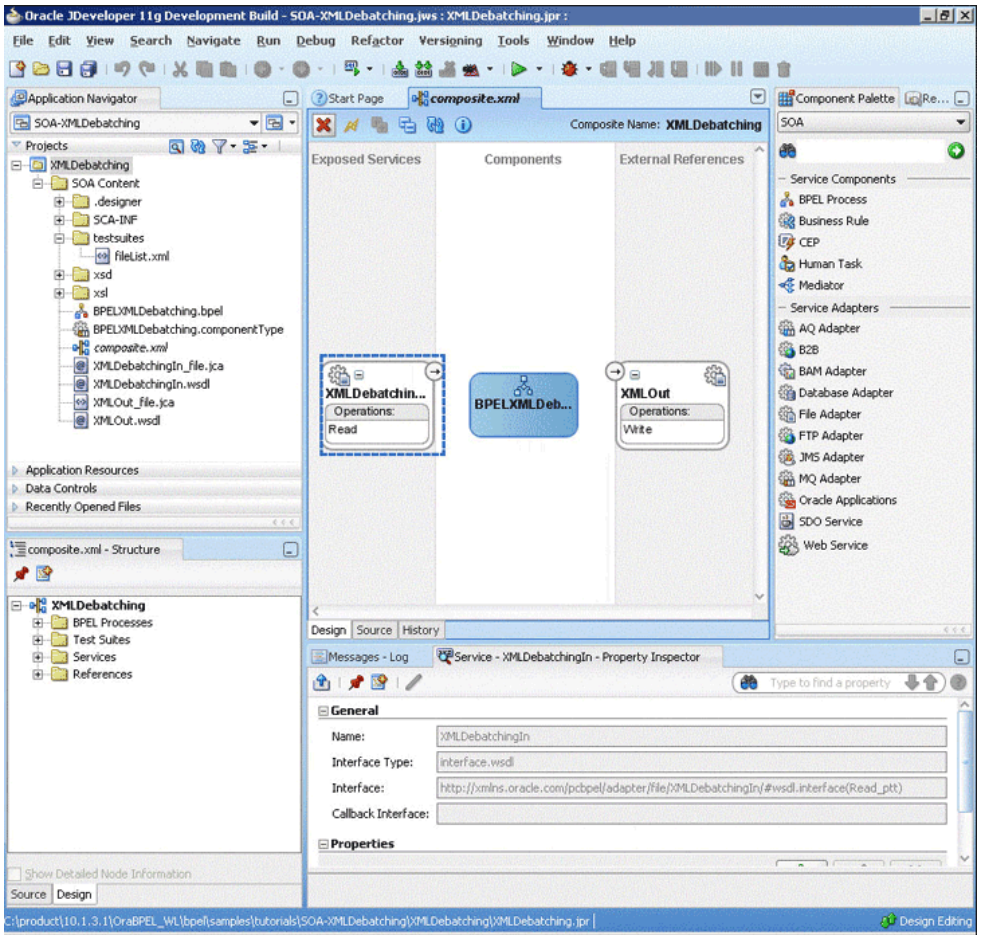


12. Click **OK**. The URL field in the Messages page is populated with the employees.xsd file, as shown in Figure 4-70.

13. Click **Next**. The **Finish** page is displayed.

14. Click **Finish**. The outbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-74.

Figure 4-74 The JDeveloper - Composite.xml



Wiring Services and Activities

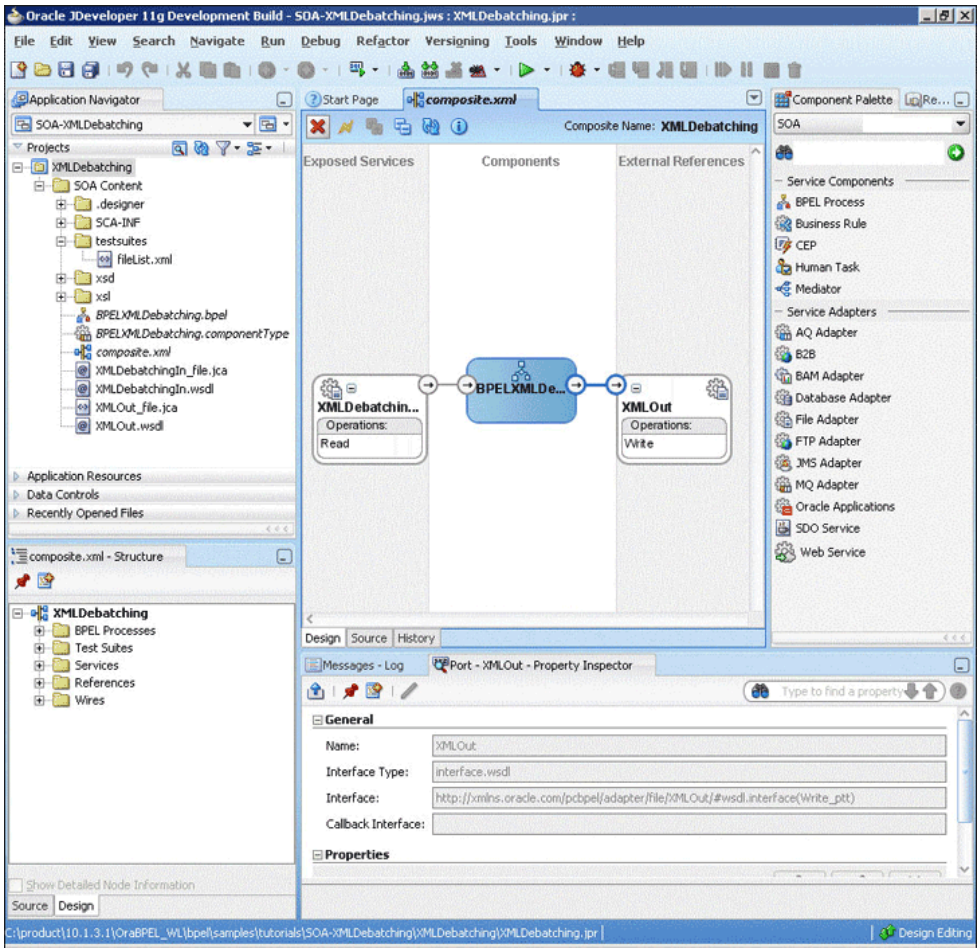
You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the **XMLDebatchingIn** in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.

2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the XMLOut in the External References area.

   The JDeveloper `composite.xml` appears, as shown in Figure 4-75.

Figure 4-75 The JDeveloper - Composite.xml

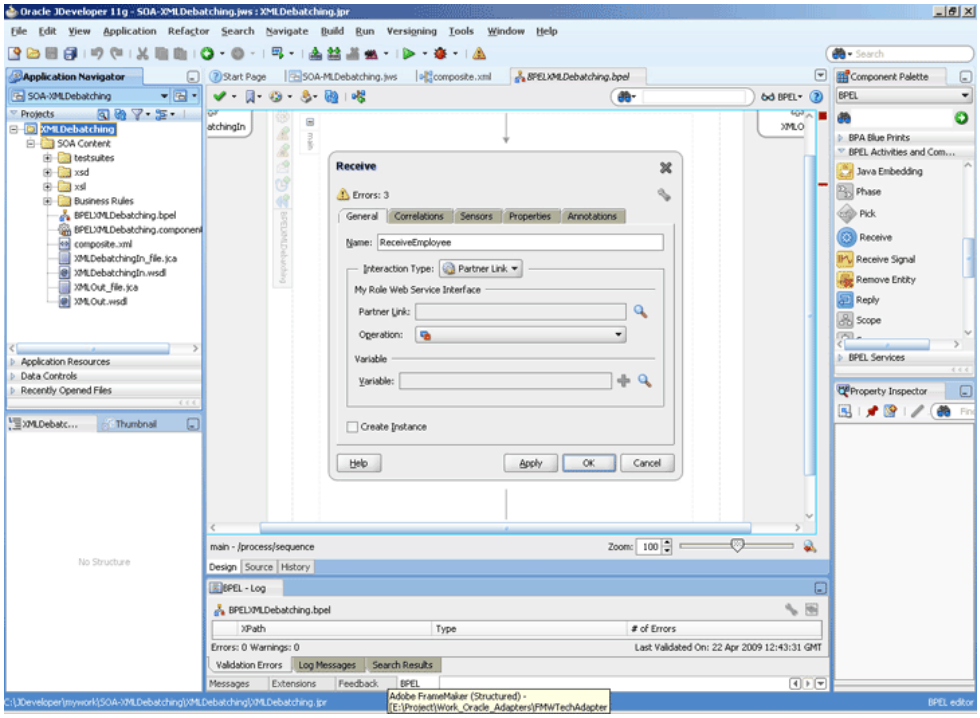


3. Click **File**, **Save All**.

Add a Receive Activity

1. Double-click **BPELXMLDebatching**. The `BPELXMLDebatching.bpel` page is displayed.
2. Drag and drop a **Receive** activity from the Components window to the design area.
3. Double-click the **Receive** activity. The **Receive** dialog is displayed.
4. Enter `ReceiveEmployee` in the **Name** field, as shown in Figure 4-76.

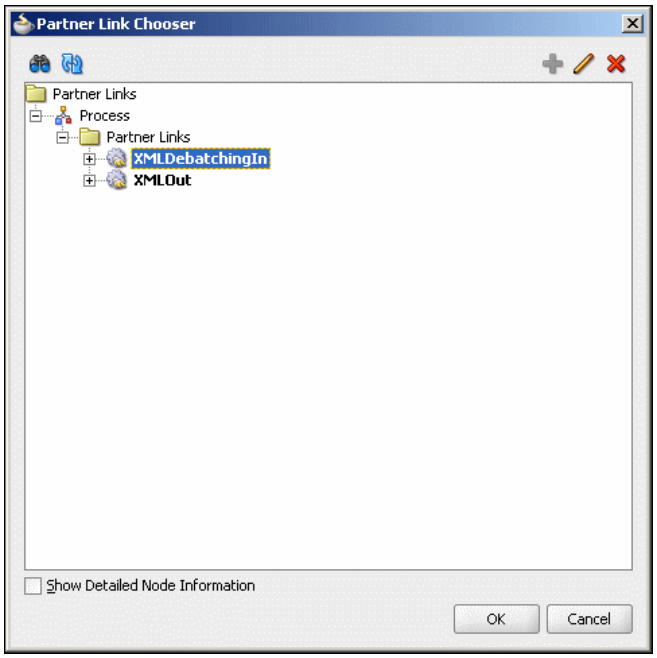
Figure 4-76 The JDeveloper - BPELXMLDebatching.bpel



5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.

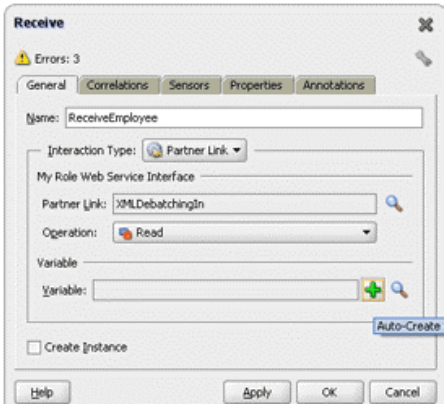
6. Select **XMLDebatchingIn**, as shown in Figure 4-77, and click **OK**.

Figure 4-77 The Partner Link Chooser Dialog



7. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog, as shown in Figure 4-78. The Create Variable dialog is displayed.

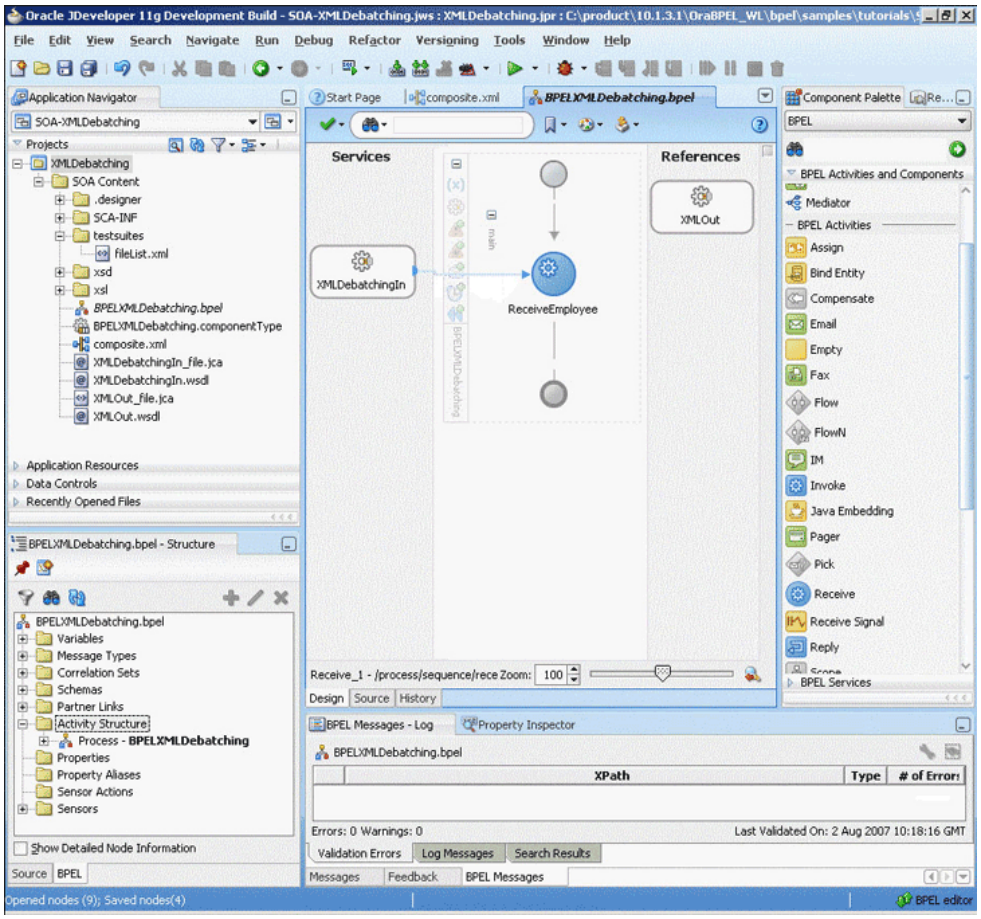
Figure 4-78 The Receive Dialog



8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.

9. Check **Create Instance**, and click **OK**. The JDeveloper BPELXMLDebatching.bpel page appears, as shown in Figure 4-79.

Figure 4-79 The JDeveloper - BPELXMLDebatching.bpel

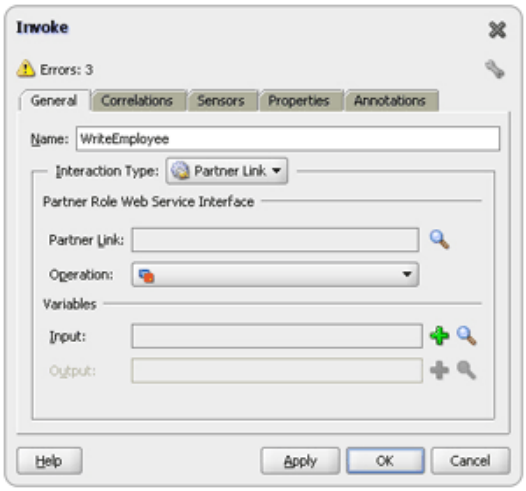


Add an Invoke Activity

1. Drag and drop an **Invoke** activity from the Components window to the design area.

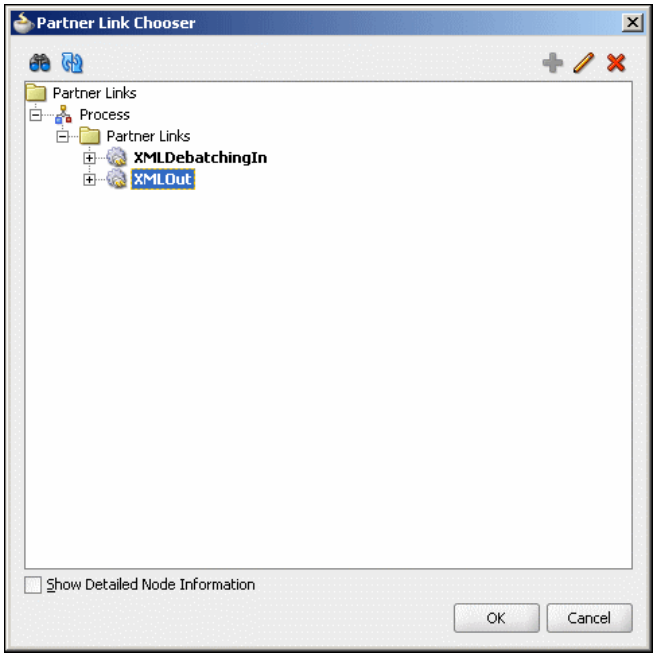
2. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.
3. Enter `WriteEmployee` in the **Name** field, as shown in Figure 4-80.

Figure 4-80 The Invoke Dialog



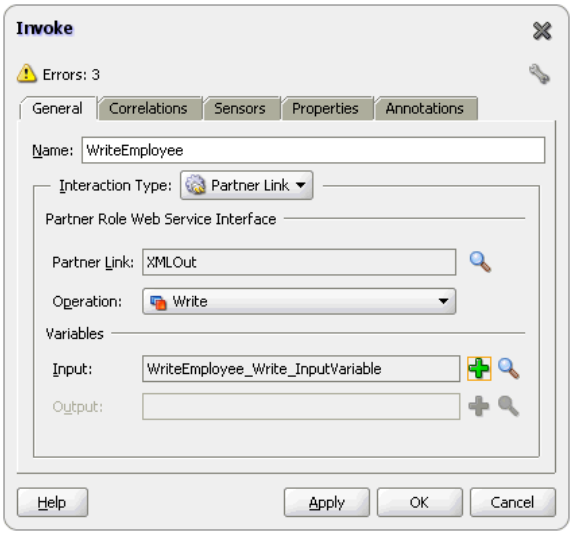
4. Click **Browse Partner Links** at the end of the **Partner Link** field. The Partner Link Chooser dialog is displayed.
5. Select **XMLOut**, as shown in Figure 4-81, and click **OK**.

Figure 4-81 The Partner Link Chooser Dialog



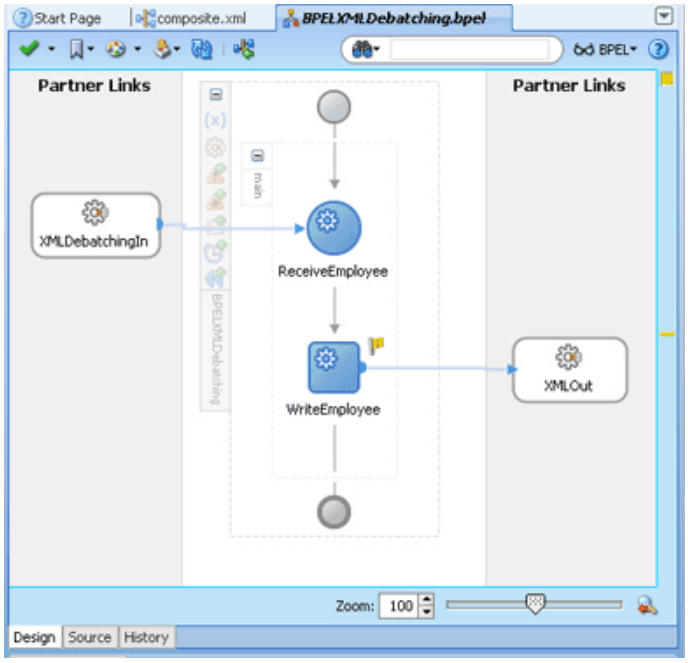
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The **Variable** field is populated with the default variable name. The **Invoke** dialog is displayed, as shown in Figure 4-82.

Figure 4-82 The Invoke Dialog



8. Click **OK**. The JDeveloper BPELXMLDebatching.bpel page appears, as shown in Figure 4-83.

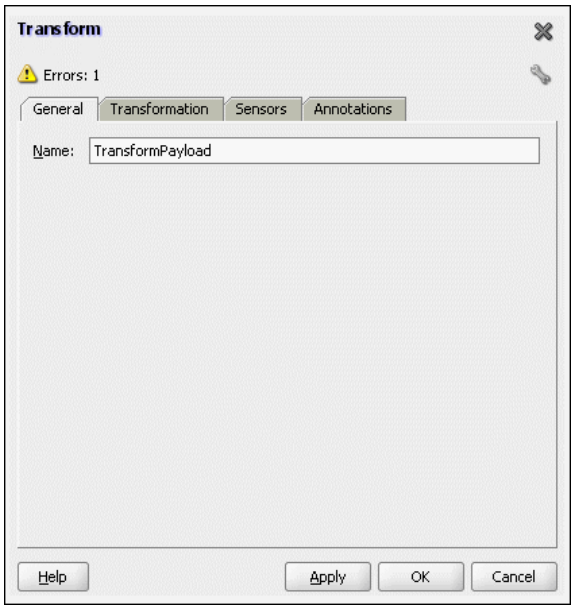
Figure 4-83 The JDeveloper - BPELXMLDebatching.bpel



Add a Transform Activity

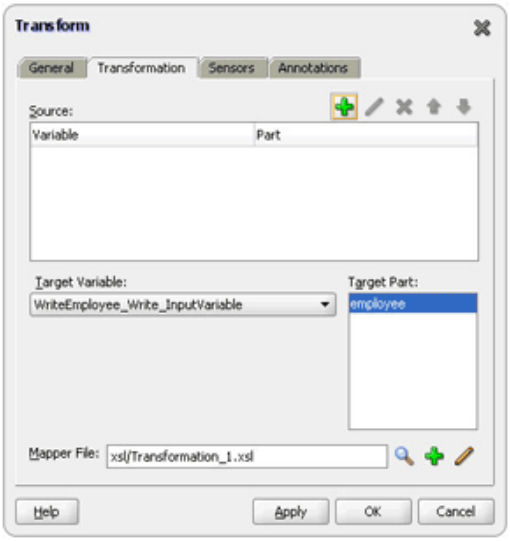
1. Drag and drop a **Transform** activity from the Components window in between the Receive and Invoke activities in the design area.
2. Double-click the **Transform** activity. The Transform dialog is displayed.
3. Enter `TransformPayload` in the Name field, as shown in Figure 4-84.

Figure 4-84 The Transform Dialog



4. Click the **Transformation** tab. The **Transform** dialog is displayed, as shown in Figure 4-85.

Figure 4-85 The Transform Dialog - Transformation Tab

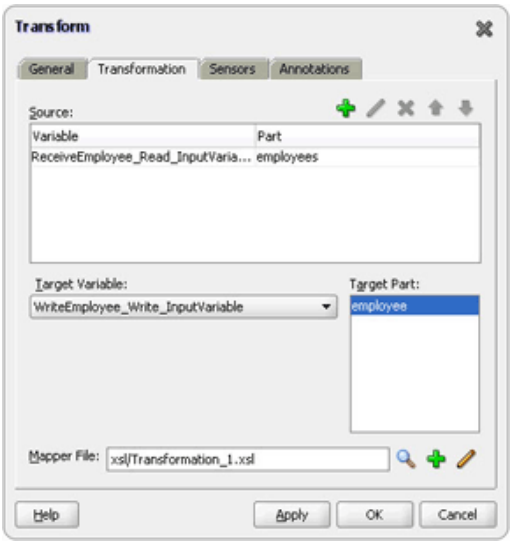


5. Click the **Create...** icon. The **Source Variable** dialog is displayed.

6. Select **ReceiveEmployee_Read_InputVariable** in the Source Variable box, and select **employees** in the Source Part box, and then click **OK**. The Transform dialog is displayed with the Source and Part selected.

7. Select **WriteEmployee_Write_InputVariable** in the Target Variable list, select **employee** in the Target Part, as shown in Figure 4-86.

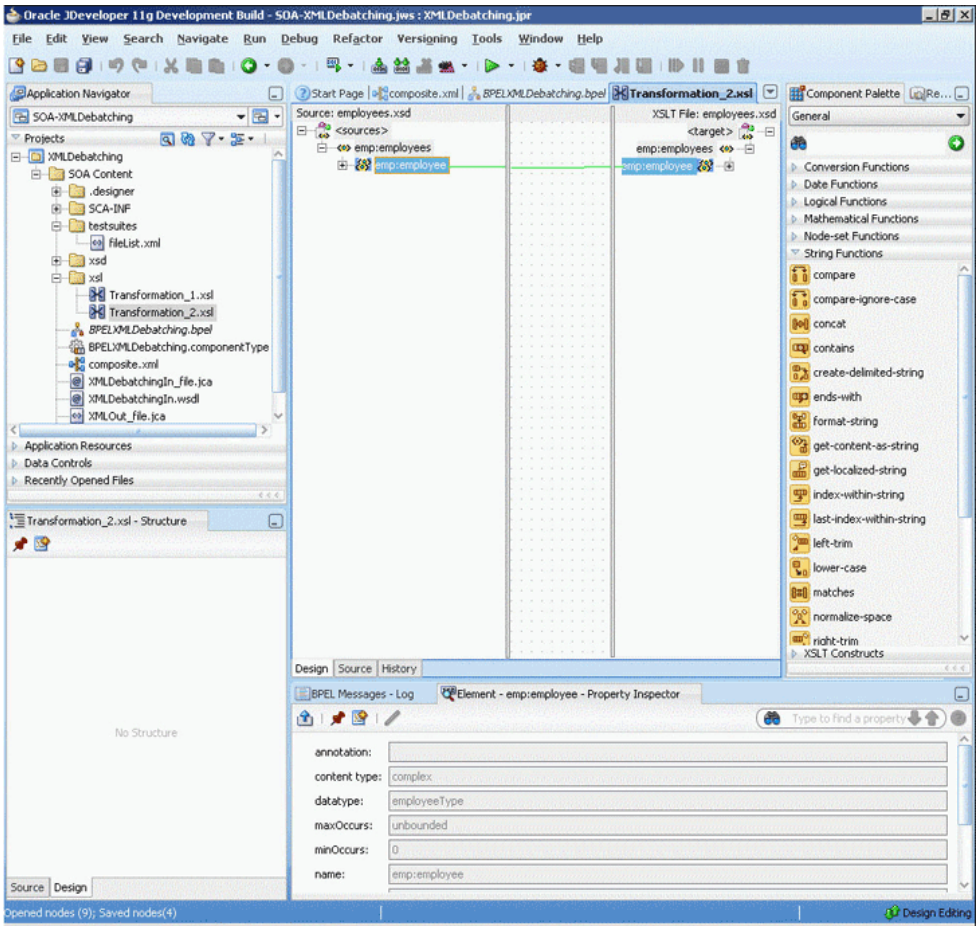
Figure 4-86 The Transform Dialog



8. Click the **Create Mapping** icon. The XSL Editor page is displayed.

9. Drag employees from sources to employee in the target, as shown in Figure 4-87. The Auto Map Preferences dialog is displayed.

Figure 4-87 The JDeveloper - Transformation_2.xsl

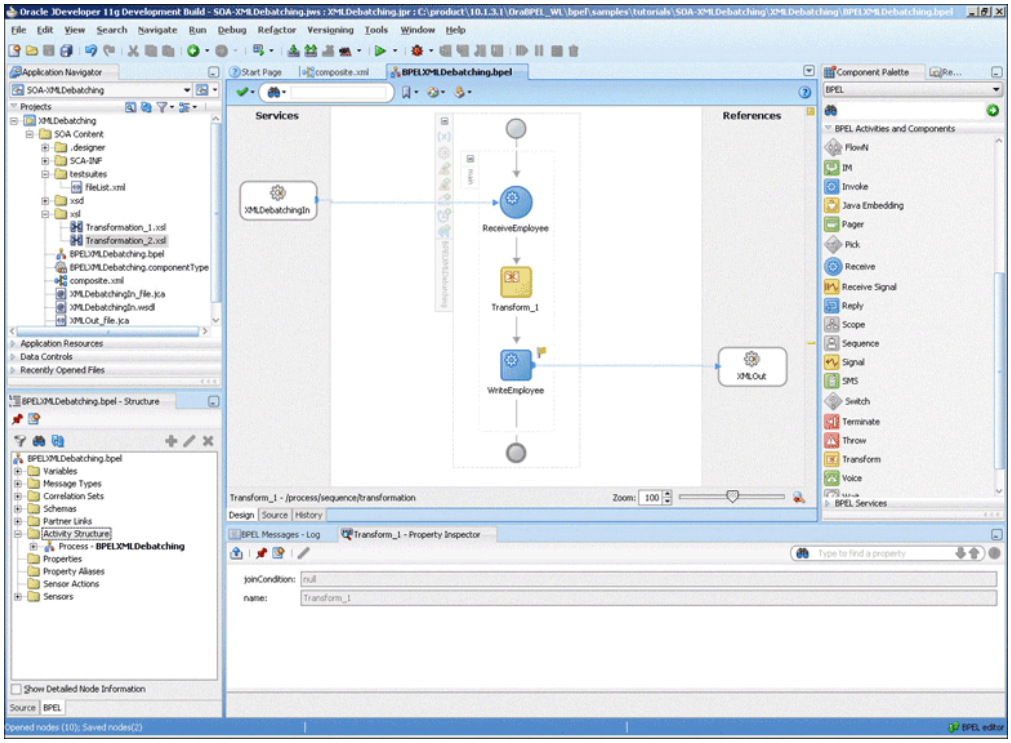


10. Click **OK**.

11. Click **File**, **Save All**.

12. Close the XSL Editor page. The **BPELXMLDebatching.bpel** page is displayed, as shown in Figure 4-88.

Figure 4-88 The JDeveloper - XML Debatching Complete



Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.

2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from

Monitoring Using Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console)

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.

2. Copy the **emps.xml** file to the input directory and ensure it gets processed. Check the output directory to ensure that an output file has been created.

3. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the Recent Instances area.

4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The Flow Trace page is displayed.
6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow.
9. Click an activity to display the activity details.

Flat Structure for Oracle BPEL PM

This use case demonstrates how a flat structure business process uses the Oracle File Adapter to process address book entries from a Comma Separated Value (CSV) file. This is then transformed and written to another file in a Fixed Length format.

This use case includes the following sections:

Prerequisites

To perform the flat structure business process, you require the following files from the `artifacts.zip` file contained in the `Adapters-101FileAdapterFlatStructure` sample:

- `artifacts/input/address-csv.txt`
- `artifacts/schemas/address-csv.xsd`
- `artifacts/schemas/address-fixedLength.xsd`
- `artifacts/xsl/addr1Toaddr2.xsl`

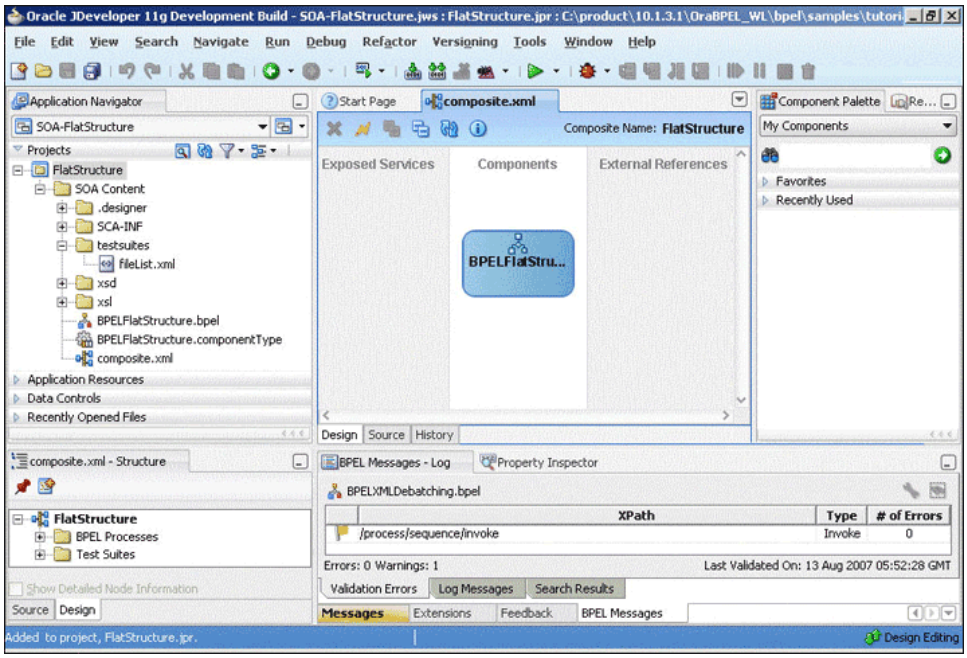
You can obtain the `Adapters-101FileAdapterFlatStructure` sample by accessing the Oracle SOA Sample Code site.

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `SOA-FlatStructure` in the **Application Name** field, and click **OK**. The Create Generic Application - Name your project page is displayed.
3. Enter `FlatStructure` in the **Project Name**.
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process - BPEL Process page is displayed.
7. Enter `BPELFlatStructure` in the **Name** field, select **Define Service Later** from the Template box.
8. Click **OK**. The SOA-FlatStructure application and the FlatStructure project appear in the design area, as shown in Figure 4-89.

Figure 4-89 The JDeveloper - Composite.xml



9. Copy the **address-csv.xsd** and **address-fixedLength.xsd** files to the schema directory in your project (see Prerequisites for the location of this file).
10. Copy **addr1Toaddr2.xsl** to the xsl directory of your project (see Prerequisites for the location of this file).

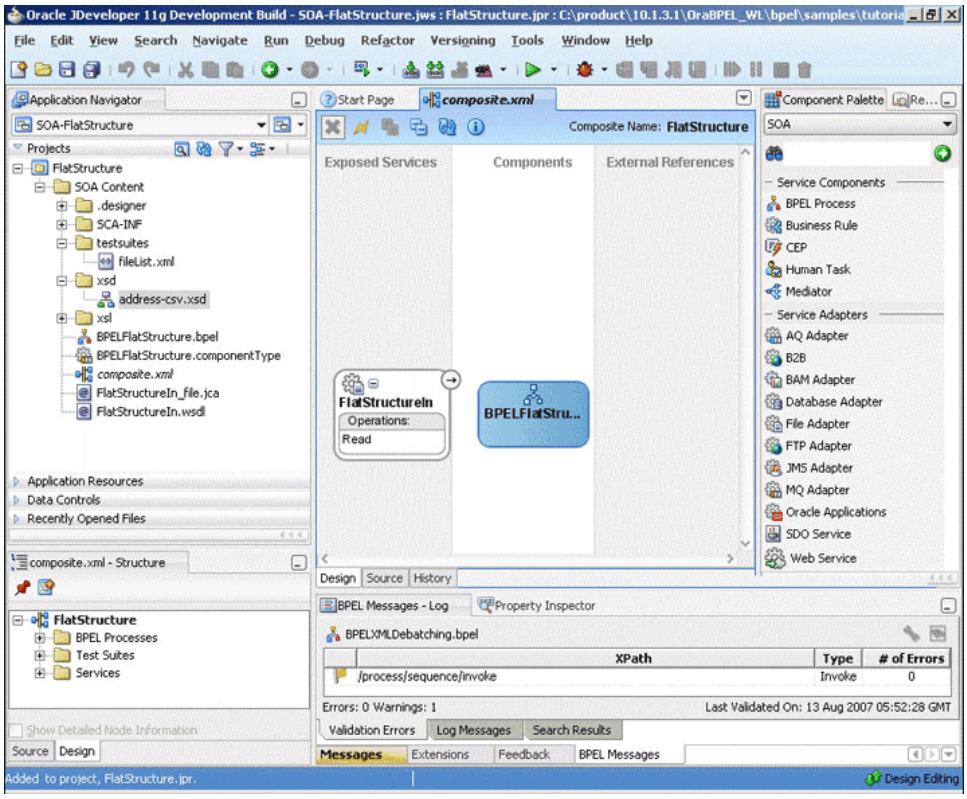
Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory:

1. Drag and drop File Adapter from the Components window to the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.

2. Click **Next**. The Service Name page is displayed.
3. Enter `FlatStructureIn` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Click **Next**. The **Operation** page is displayed.
6. Select **Read File**, and click **Next**. The File Directories page is displayed.
7. Enter the physical path for the input directory. Check **Archive Processed Files**.
8. Enter the physical path for the archive directory for processed files.
9. Click **Next**. The **File Filtering** page is displayed.
10. Enter `*.txt` in the **Include Files With Name Pattern** field, click **Next**. The File Polling page is displayed.
11. Click **Next**. The **Messages** page is displayed.
12. Click **Browse For Schema File** that appears at the end of the URL field. The Type **Chooser** dialog is displayed.
13. Click **Project Schema Files**, **address-csv.xsd**, and **Root-Element**.
14. Click **OK**. The URL field in the **Messages** page is populated with the `address-csv.xsd` file.
15. Click **Next**. The **Finish** page is displayed.
16. Click **Finish**. The inbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-90.

Figure 4-90 The JDeveloper - Composite.xml



Creating the Outbound Oracle File Adapter Service

Perform the following steps to create an outbound Oracle File Adapter service to write the file from a local directory to the FTP server:

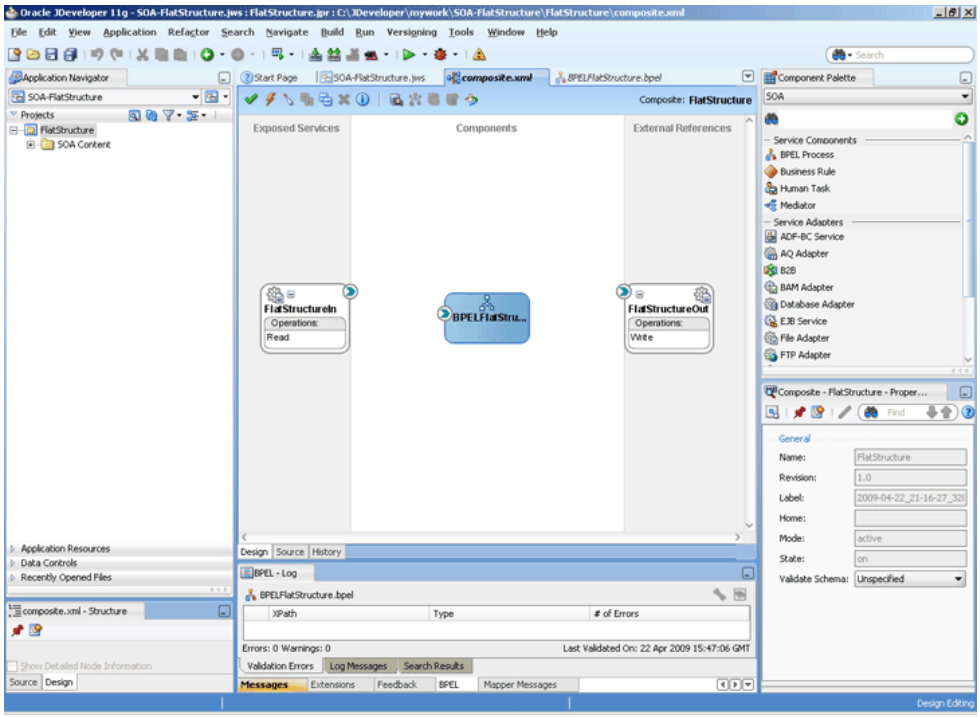
1. Drag and drop File Adapter from the Components window to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `FlatStructureOut` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, and click **Next**. The **File Configuration** page is displayed.
7. Enter the physical path for the output directory and enter `address_%SEQ%.data` in the **File Naming Convention(po_%SEQ%.txt)** field.
8. Click **Next**. The **Messages** page is displayed.
9. Click **Browse For Schema File** that appears at the end of the URL field. The **Type Chooser** dialog is displayed.
10. Click **Project Schema Files**, **address-fixedLength.xsd**, and **Root-Element**.

11. Click **OK**. The URL field in the **Messages** page is populated with the address-fixedLength.xsd file.

12. Click **Next**. The **Finish** page is displayed.

13. Click **Finish**. The outbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-91.

Figure 4-91 The JDeveloper - Composite.xml



Wiring Services and Activities

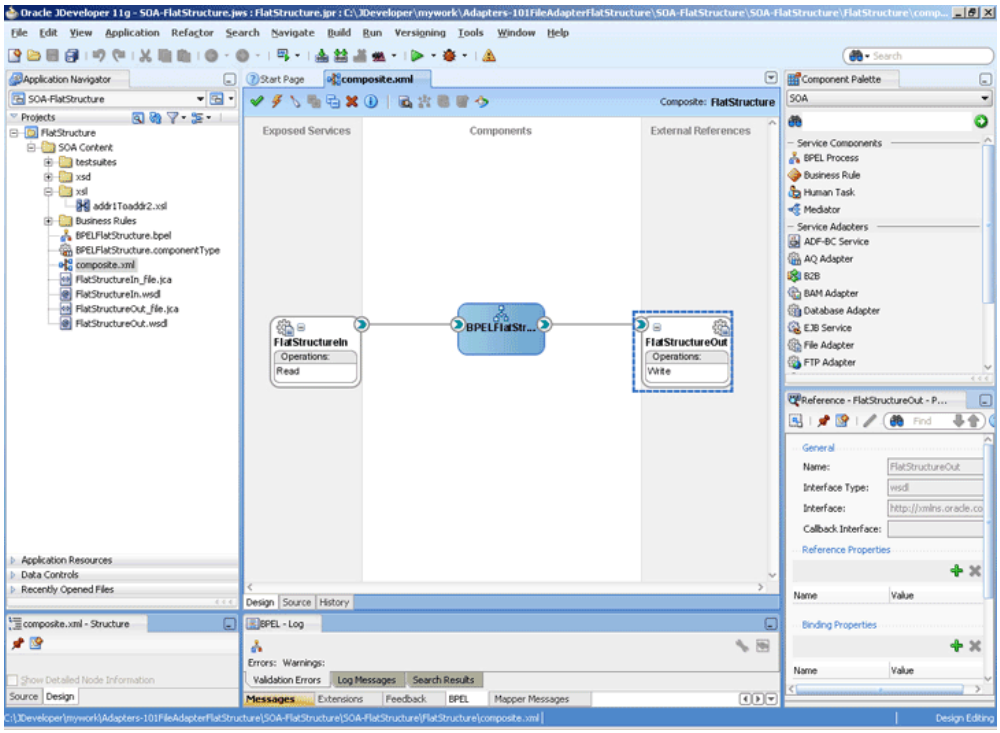
You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the **FlatStructureIn** in the **Exposed Services** area to the drop zone that appears as a green triangle in the BPEL process in the **Components** area.

2. Drag the small triangle in the BPEL process in the **Components** area to the drop zone that appears as a green triangle in the FlatStructureOut in the **External References** area.

   The JDeveloper `composite.xml` appears, as shown in Figure 4-92.

Figure 4-92 The JDeveloper - composite.xml

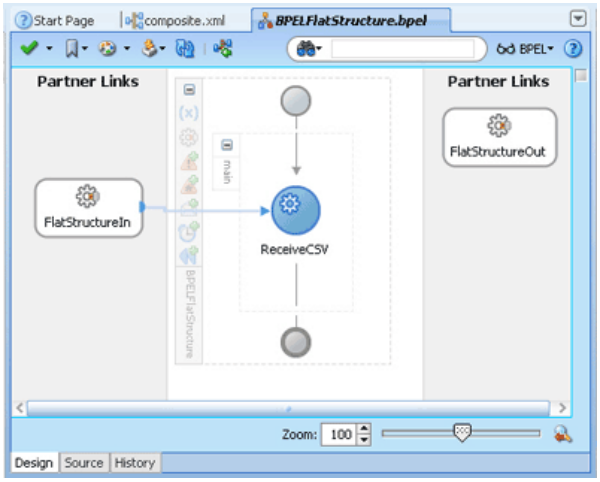


3. Click **File**, **Save All**.

Add a Receive Activity

1. Double-click **BPELFlatStructure**. The BPELFlatStructure.bpel page is displayed.
2. Drag and drop a **Receive** activity from the Components window to the design area.
3. Double-click the **Receive** activity. The Receive dialog is displayed.
4. Enter `ReceiveCSV` in the **Name** field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
6. Select **FlatStructureIn**, and click **OK**.
7. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog. The Create Variable dialog is displayed.
8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper BPELFlatStructure.bpel page appears, as shown in Figure 4-93.

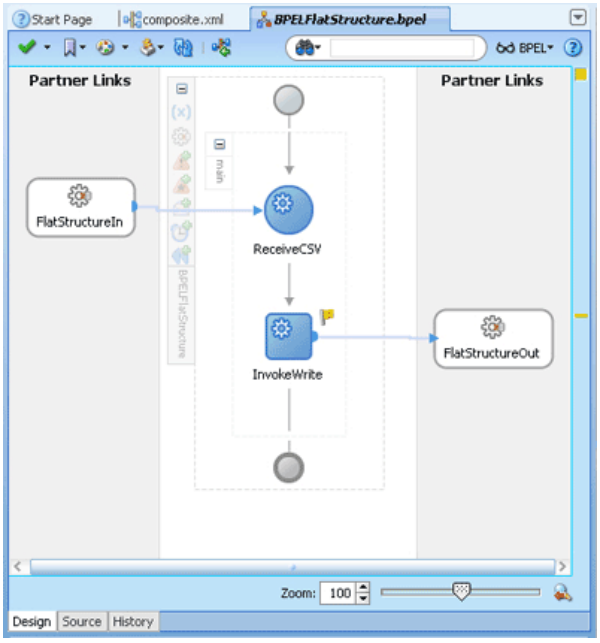
Figure 4-93 The JDeveloper - BPELFlatStructure.bpel



Add an Invoke Activity

1. Drag and drop an **Invoke** activity from the Components window to the design area.
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `InvokeWrite` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **FlatStructureOut**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
8. Click **OK**. The JDeveloper BPELFlatStructure.bpel page appears, as shown in Figure 4-94.

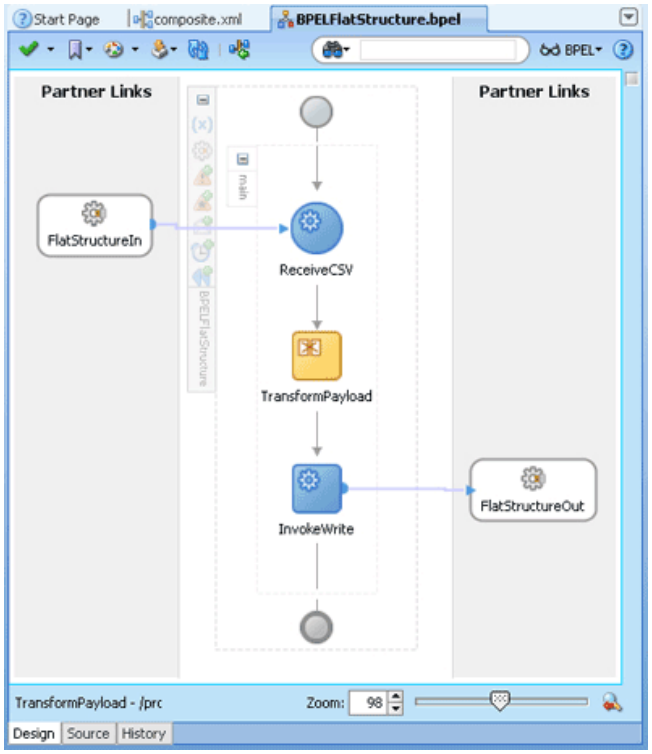
Figure 4-94 The JDeveloper - BPELFlatStructure.bpel



Add a Transform Activity

1. Drag and drop a **Transform** activity from the Components window in between the Receive and Invoke activities in the design area.
2. Double-click the **Transform** activity. The Transform dialog is displayed.
3. Enter `TransformPayload` in the **Name** field.
4. Click the **Transformation** tab. The **Transform** dialog is displayed.
5. Click the **Create...** icon. The **Source Variable** dialog is displayed.
6. Select **ReceiveCSV_Read_InputVariable** in the Source Variable box, and select **body** in the Source Part box, and then click **OK**. The Transform dialog is displayed with the Source and Part selected.
7. Select **InvokeWrite_Write_InputVariable** in the Target Variable list, select **body** in the Target Part.
8. Click the **Browse** button at the end of the Mapper File field and select **addr1Toaddr2.xsl** file from the xsl directory in your project.
9. Click **OK**.
10. Click **File**, **Save All**. The **BPELFlatStructure.bpel** page is displayed, as shown in Figure 4-95.

Figure 4-95 The JDeveloper - BPELFlatStructure.bpel



Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from .

Monitoring Using Oracle Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the `address-csv.txt` file to the input directory and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The **Dashboard** is displayed.

   Note your Instance ID in the Recent Instances area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The Flow Trace page is displayed.

6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow. Additionally, click an activity (such as invoke, receive) to view the details of an activity.
9. Click **ReceiveCSV** to display the activity details.

Flat Structure for Mediator

In this use case, Mediator receives the customer data from a file system as a text file, through an inbound Oracle File Adapter service named `ReadFile`. The `ReadFile` adapter service sends the message to a routing service named `ReadFile_RS`. The `ReadFile_RS` sends the message to the outbound adapter service `WriteFTP`. The `WriteFTP` service delivers the message to its associated external application.

This use case includes the following sections:

Prerequisites

This example assumes that you are familiar with basic Mediator constructs, such as services, routing service, and JDeveloper environment for creating and deploying Mediator services.

To perform the flat structure for Mediator business process, you require the following files from the `artifacts.zip` file contained in the `Adapters-101FileAdapterFlatStructure` sample:

- `artifacts/schemas/address-csv.xsd`

You can see the `Adapters-101FileAdapterFlatStructure` sample by accessing the Oracle SOA Sample Code site.

Creating a Mediator Application and Project

To create an application and a project for the use case, follow these steps:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.

2. Enter `FileFTP_RW` in the **Application Name** field and click **Next**. The Create Generic Application - Name your project page is displayed.
3. Enter `FileRead_FTPWrite` in the **Project Name** field.
4. In the Available list in the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Create Generic Application - Configure SOA settings page is displayed.
6. Select **Composite With Mediator** in the Composite Template box.
7. Click **Finish**. The Create Mediator - Mediator Component page is displayed.
8. Enter `FileRead_RS` in the Name field.
9. Select **Define Interface Later** in the Template list, and then click **OK**. The `FileFTP_RW` application and the `FileRead_FTPWrite` project appear in the design area.

Importing the Schema Definition (.XSD) Files

Perform the following steps to import the XSD files that define the structure of the messages:

1. Create a `Schema` directory and copy the `address-csv.xsd` file to this directory (see Prerequisites for the location of this file).
2. In the **Application Navigator**, select **FileRead_FTPWrite**.
3. From the **File** menu, select **Import**. The Import dialog is displayed.
4. From the **Select What You Want to Import** list, select **Web Source,** and then click **OK.** The **Web Source** dialog is displayed.
5. To the right of the **Copy From** field, click **Browse.** The Choose Directory dialog is displayed.
6. Navigate to the **Schema** directory and click **Select**. The Web Source dialog with the directory is displayed.
7. Click **OK**.

Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory

1. Drag a File Adapter service from the **Components** to the design area. The **Adapter Configuration Wizard Welcome page** is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `ReadFile` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Click **Next**. The Operation page is displayed.

6. Select **Read File** and click **Next**. The **File Directories** page is displayed.
7. Select **Physical Path** option, and click **Browse** and select a polling directory.
8. Click **Next**. The File Filtering page is displayed.
9. Enter `*.txt` in the **Include Files with Name Pattern** field and click **Next**. The File Polling page is displayed.
10. Click **Next**. The Messages page is displayed.
11. Click the **Browse For Schema File** button at the end of the URL field. The Type Chooser dialog is displayed.
12. Select **Project Schema Files**, **address-csv.xsd**, and then **Root-Element**.
13. Click **OK**.
14. Click **Next** in the Messages page. The **Finish** page is displayed.
15. Click **Finish**. A ReadFile adapter service is created.

Creating the Outbound Oracle FTP Adapter Service

Perform the following steps to create an outbound Oracle FTP Adapter service to write the file to an FTP server:

1. Drag an FTP Adapter service from **Components** to the design area. The Adapter Configuration Wizard Welcome page is displayed.
2. Click Next. The **Service Name** page is displayed.
3. Enter `WriteFTP` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The FTP Server Connection page is displayed.
6. Specify the JNDI Name of the FTP Server in the **FTP Server JNDI Name** field and click **Next**. The Operation page is displayed.
7. Select **Ascii** option as File Type.
8. Select **Put File** option as the Operation Type and click **Next**. The **File Configuration page** is displayed.
9. Specify the directory to which file must be written in the **Directory for Outgoing Files (physical path)** field.
10. Specify the naming convention for the output file name in the **File Naming Convention** field. For example, `po_%SEQ%.txt`.
11. Click **Next**. The **Messages page** is displayed.
12. Click the **Browse For Schema File** button at the end of the URL field. The **Type Chooser** dialog is displayed.
13. Select **Project Schema Files**, **address-csv.xsd**, and then **Root-Element**.
14. Click **OK**.

15. Click **Next** in the **Messages** page. The **Finish** page is displayed.

16. Click **Finish**. A WriteFTP adapter service is created.

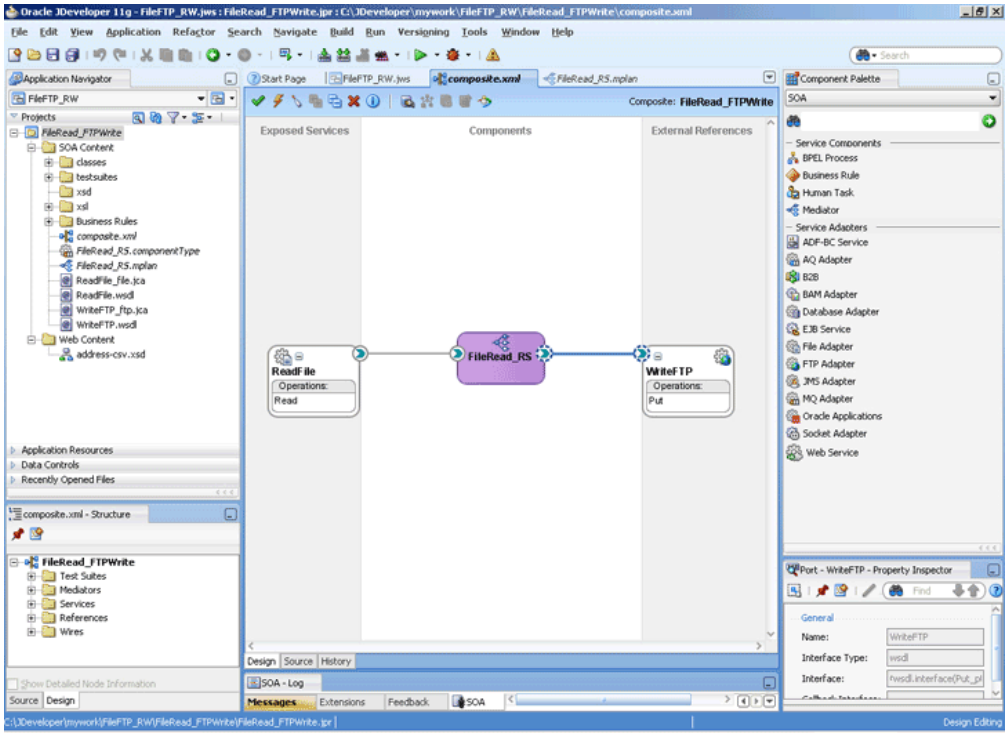
Wiring Services

You have to assemble or wire the three components that you have created: Inbound Oracle File Adapter service, Mediator component, Outbound Oracle FTP Adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the ReadFile in the **Exposed Services** area to the drop zone that appears as a green triangle in the Mediator component in the Components area.

2. Drag the small triangle in the Mediator component in the **Components** area to the drop zone that appears as a green triangle in the WriteFTP in the **External References** area. The JDeveloper `composite.xml` appears, as shown in Figure 4-96.

Figure 4-96 The JDeveloper - Composite.xml

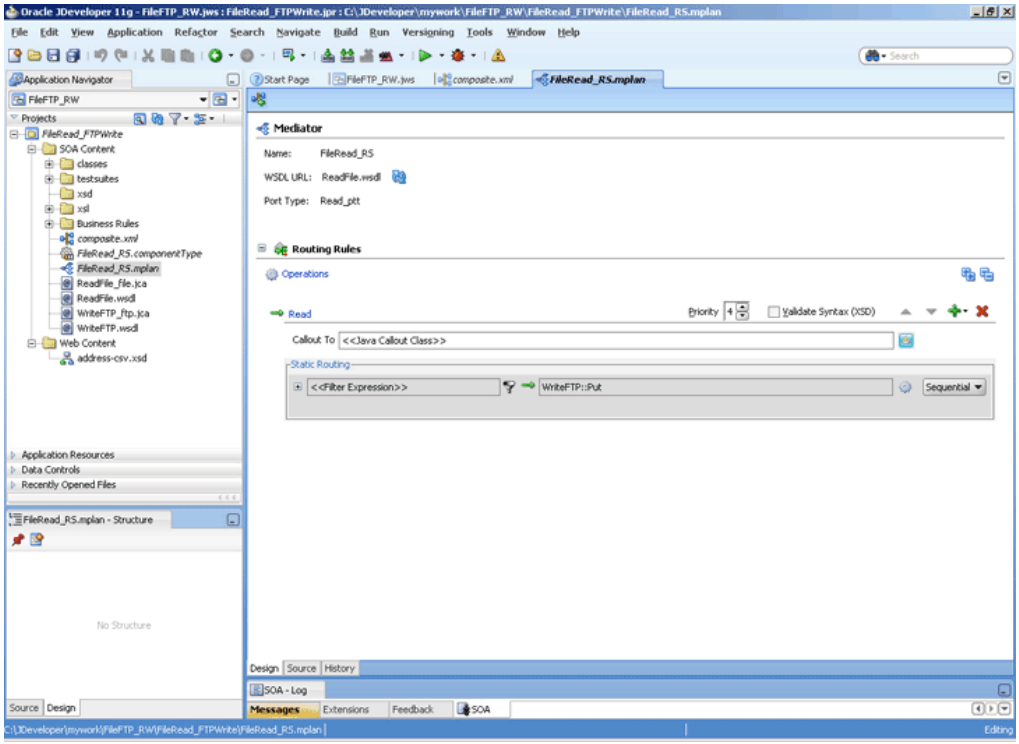


Creating the Routing Rule

Perform the following steps to create a routing service:

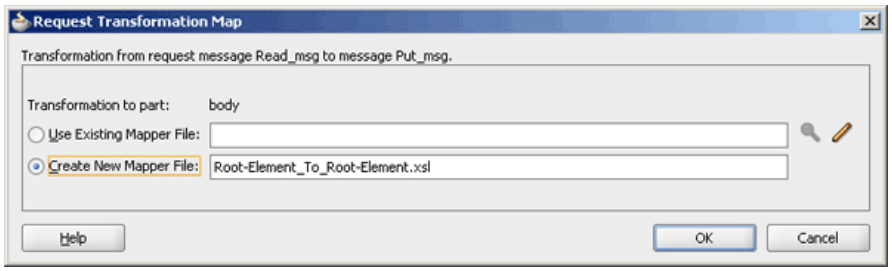
1. Double-click the **ReadFile_RS** routing service. The Read operation is listed in the Operations pane, as shown in Figure 4-97.

Figure 4-97 The JDeveloper - ReadFile_RS Routing Service Page



2. Click the + sign to the left of **<<Filter Expression>>**to expand the routing rule.
3. Click the icon that appears at the end of the Transform Using field. The **Request Transformation Map dialog** is displayed, as shown in Figure 4-98.

Figure 4-98 The Request Transformation Map Dialog



4. Select **Create New Mapper File** and click **OK**.

   A **Root-Element_To_Root-Element.xsl** tab is added to JDeveloper. This tab enables you to graphically create a document transformation file to convert the structure of the file data to a canonical data structure.
5. Drag and drop the **imp1:Address** source element to the **imp1:Address** target element. The Auto Map Preferences dialog is displayed.
6. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names.**
7. Click **OK**.
8. From the **File** menu, click **Save**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from .

Runtime Task

At runtime, copy a text file to the polling directory. Once the Oracle File Adapter picks the file, it writes the file to the directory that you specified at design time.

Oracle File Adapter Scalable DOM

This use case demonstrates how a scalable DOM process uses the streaming feature to copy/move huge files from one directory to another.

The streaming option is not supported with DB2 hydration store.

This use case includes the following sections:

- Prerequisites
- Designing the SOA Composite
- Creating the Inbound Service
- Creating the Outbound Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using Fusion Middleware Control Console

Prerequisites

To perform the streaming large payload process, you require the following files from the `artifacts.zip` file contained in the `Adapters-103FileAdapterScalableDOM` sample:

- `artifacts/schemas/address-csv.xsd`
- `artifacts/input/address-csv-large.txt`

You can obtain the `Adapters-103FileAdapterScalableDOM` sample by accessing the Oracle SOA Sample Code site.

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.

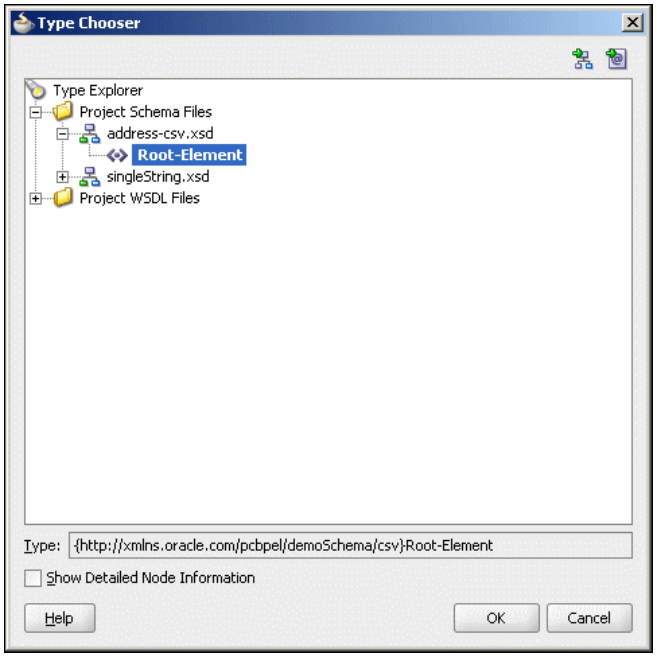
2. Enter `SOA-ScalableDOM` in the **Application Name** field, and click **Next**. The Create Generic Application - Name your project page is displayed.
3. Enter `ScalableDOM` in the **Project Name** field.
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The SOA-ScalableDOM application and ScalableDOM project appears in the Application Navigator and the Create BPEL Process - BPEL Process page is displayed.
7. Enter `BPELScalableDOM` in the **Name** field, select **Define Service Later** from the Template box.
8. Click **OK**. The SOA-ScalableDOM application and the ScalableDOM project appears in the design area.
9. Copy the **address-csv.xsd** file to the xsd directory in your project (see Prerequisites for the location of this file).

Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory:

1. Drag and drop File Adapter from the **Components** window to the Exposed Services swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `ScalableDOMIn` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Read File**, and check **Use File Streaming**, and click **Next**. The File Directories page is displayed.
7. Enter the physical path for the input directory. The File Filtering page is displayed.
8. Enter `*.txt` in the **Include Files With Name Pattern** field, click **Next**. The File Polling page is displayed.
9. Click **Next**. The Messages page is displayed.
10. Click **Browse For Schema File** that appears after the URL field. The Type Chooser dialog is displayed.
11. Click **Project Schema Files**, **address-csv.xsd**, and **Root-Element**, as shown in Figure 4-99.

Figure 4-99 The Type Chooser Dialog

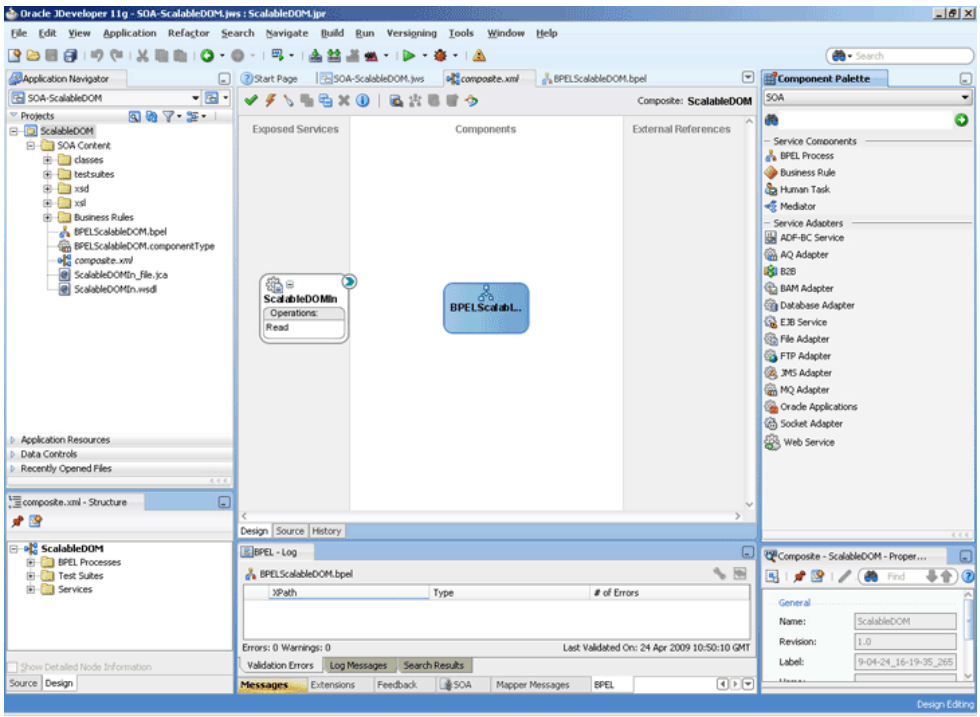


12. Click **OK**. The URL field in the Messages page is populated with the `address-csv.xsd` file.

13. Click **Next**. The **Finish** page is displayed.

14. Click **Finish**. The inbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-100.

Figure 4-100 The JDeveloper - Composite.xml

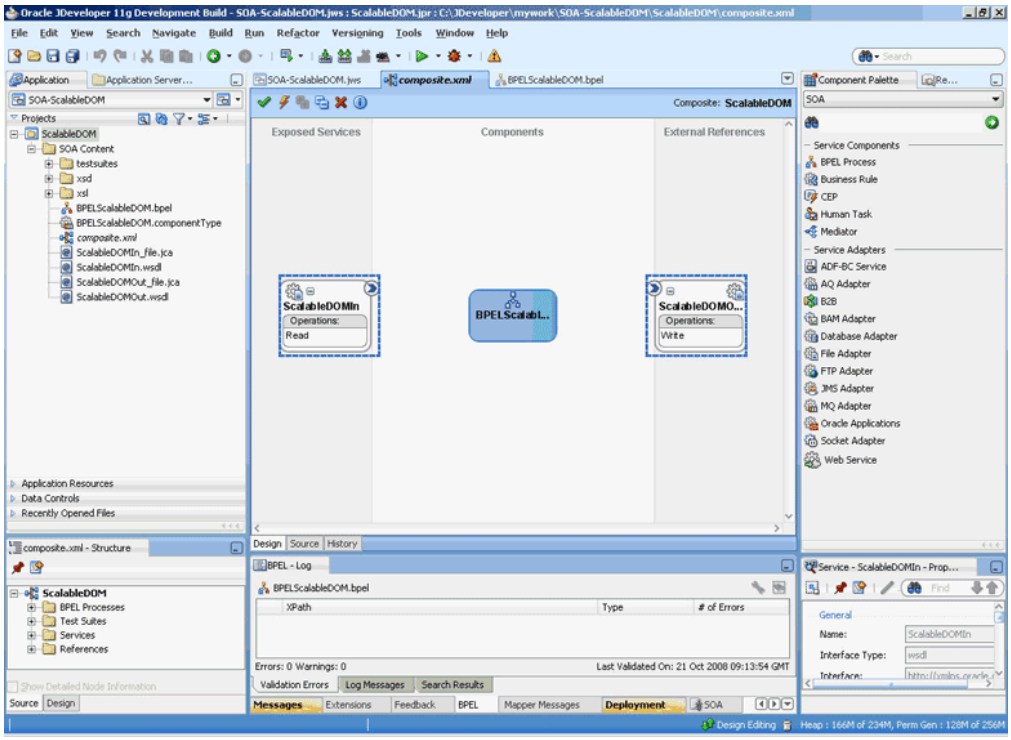


Creating the Outbound Oracle File Adapter Service

Perform the following steps to create an outbound Oracle File Adapter service to write the file from a local directory to the FTP server:

1. Drag and drop File Adapter from the Components window to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `ScalableDOMOut` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, and click **Next**. The **File Configuration** page is displayed.
7. Enter the physical path for the output directory and enter `address-csv_%SEQ%.xml` in the **File Naming Convention (po_%SEQ%.txt)** field.
8. Click **Next**. The **Messages** page is displayed.
9. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
10. Click **Project Schema Files**, **address-csv.xsd**, and **Root-Element**.
11. Click **OK**. The URL field in the **Messages** page is populated with the `address-csv.xsd` file.
12. Click **Next**. The **Finish** page is displayed.
13. Click **Finish**. The outbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-101.

Figure 4-101 The JDeveloper - Composite.xml



Wiring Services and Activities

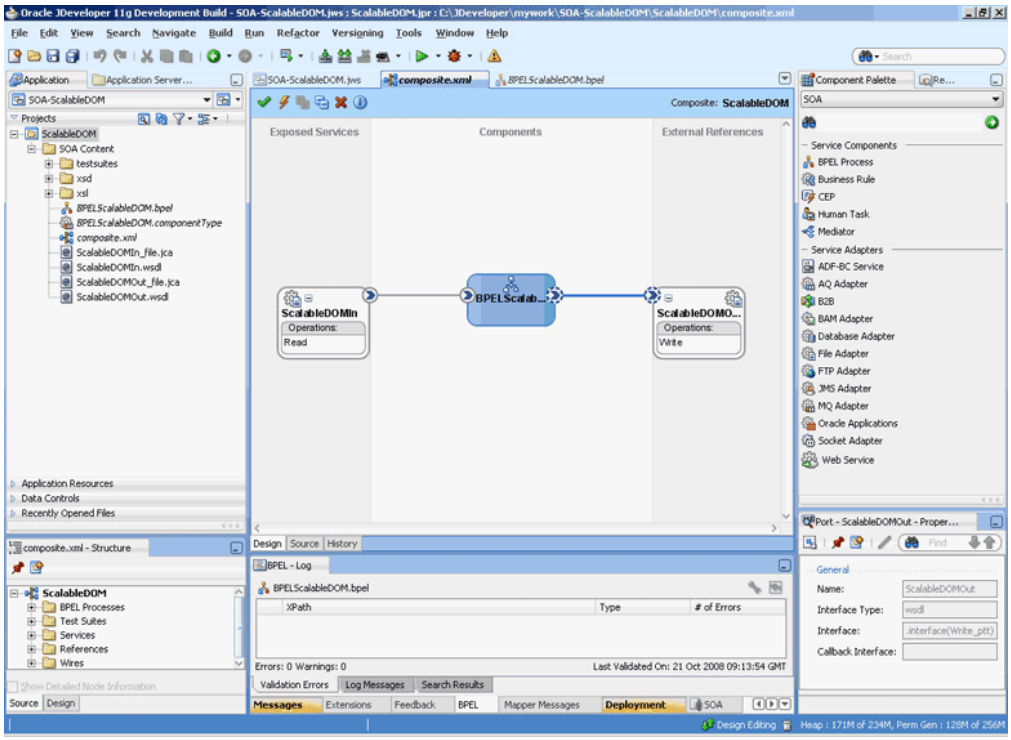
You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the ScalableDOMIn in the **Exposed Services** area to the drop zone that appears as a green triangle in the BPEL process in the **Components** area.

2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the ScalableDOMOut in the **External References** area.

   The JDeveloper `composite.xml` appears, as shown in Figure 4-102.

Figure 4-102 The JDeveloper - Composite.xml

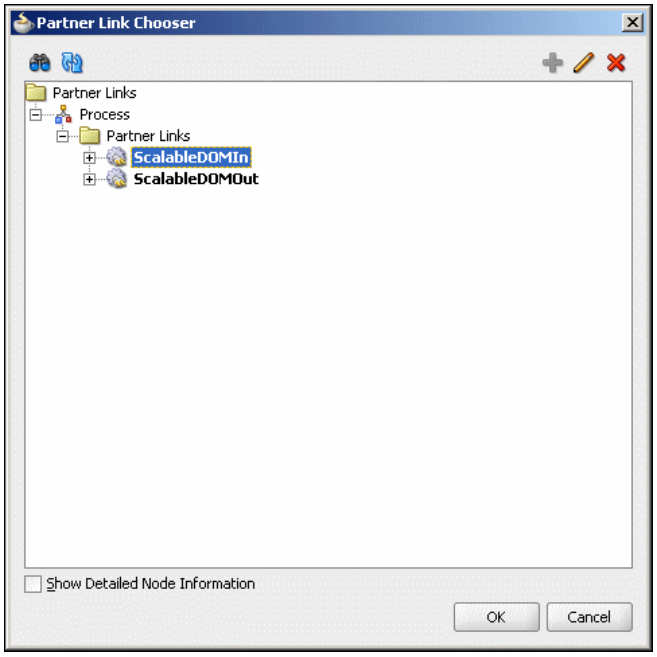


3. Click **File**, **Save All**.

Add a Receive Activity

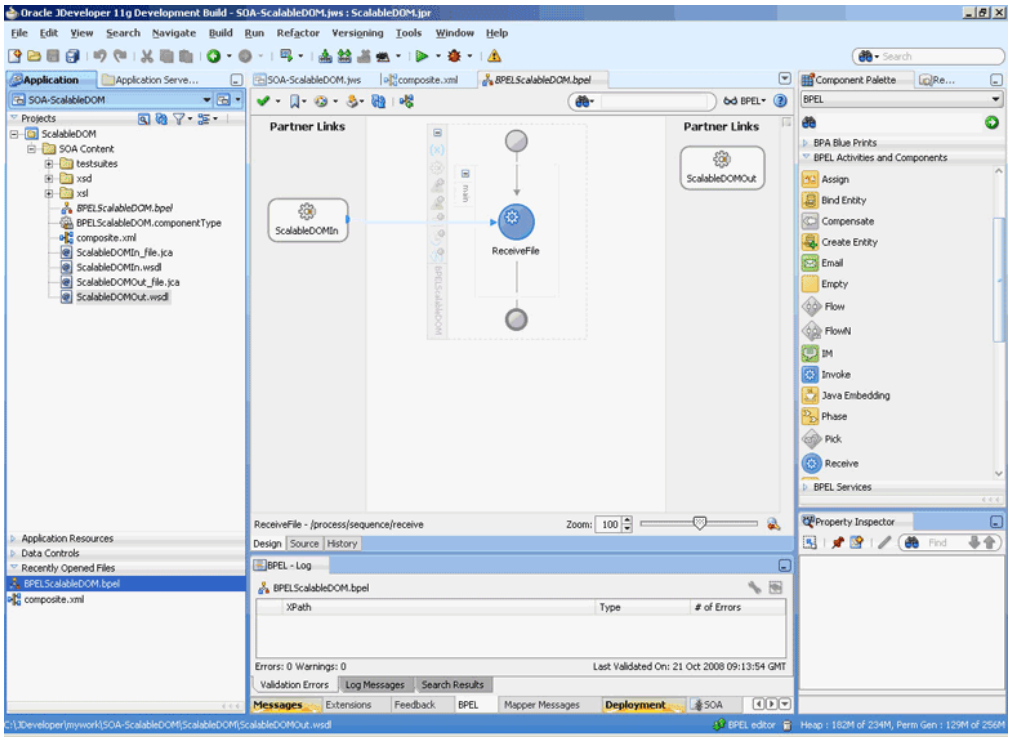
1. Double-click **BPELScalableDOM**. The BPELScalableDOM.bpel page is displayed.
2. Drag and drop a **Receive** activity from the Components window to the design area.
3. Double-click the **Receive** activity. The **Receive** dialog is displayed.
4. Enter `ReceiveFile` in the **Name** field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.
6. Select **ScalableDOMIn**, as shown in Figure 4-103, and click **OK**.

Figure 4-103 The Partner Link Chooser Dialog



7. Click the **Auto-Create Variable** icon to the right of the Variable field in the **Receive** dialog. The **Create Variable** dialog is displayed.
8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper `composite.xml` page appears, as shown in Figure 4-104.

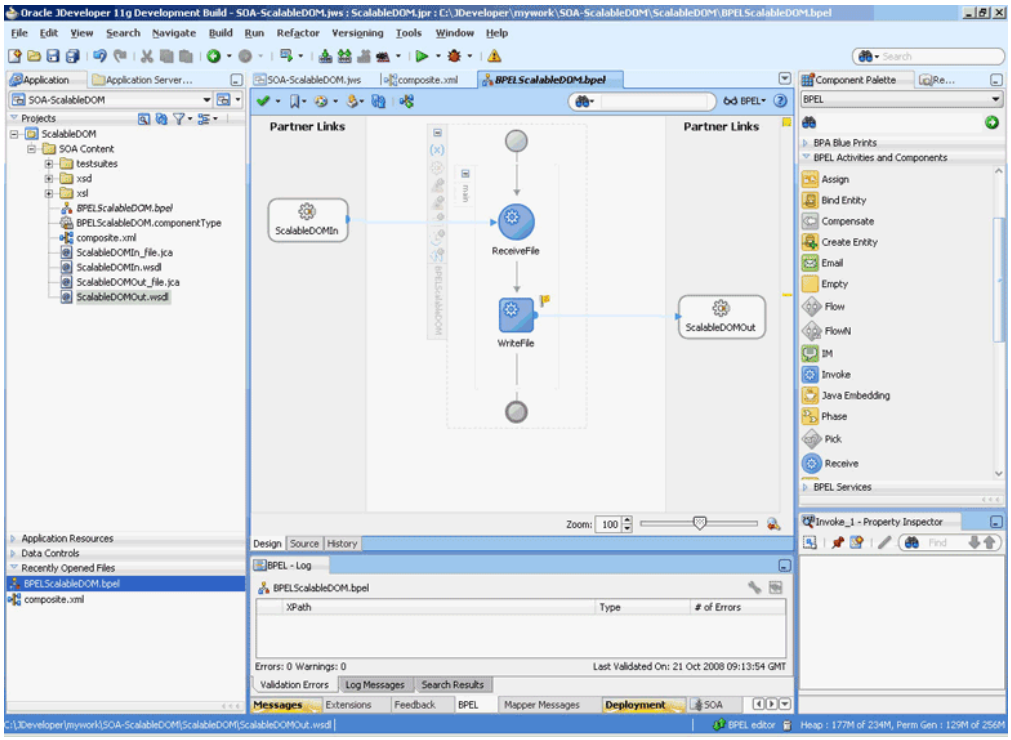
Figure 4-104 The JDeveloper - BPELScalableDOM.bpel



Add an Invoke Activity

1. Drag and drop an **Invoke** activity from the Components window to the design area.
2. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.
3. Enter `WriteFile` in the Name field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.
5. Select **ScalableDOMOut**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The **Create Variable** dialog is displayed.
7. Select the default variable name and click **OK**. The **Input variable** field is populated with the default variable name. The **Invoke** dialog is displayed.
8. Click **OK**. The JDeveloper BPELScalableDOM.bpel page appears, as shown in Figure 4-105.

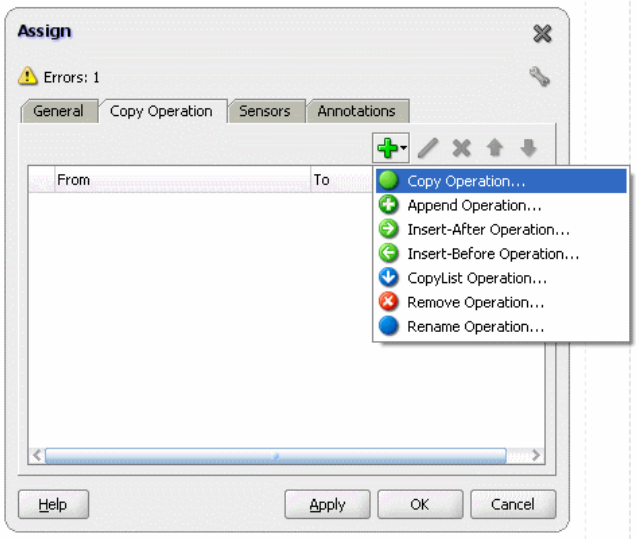
Figure 4-105 The JDeveloper - BPELScalableDOM.bpel Page



Add an Assign Activity

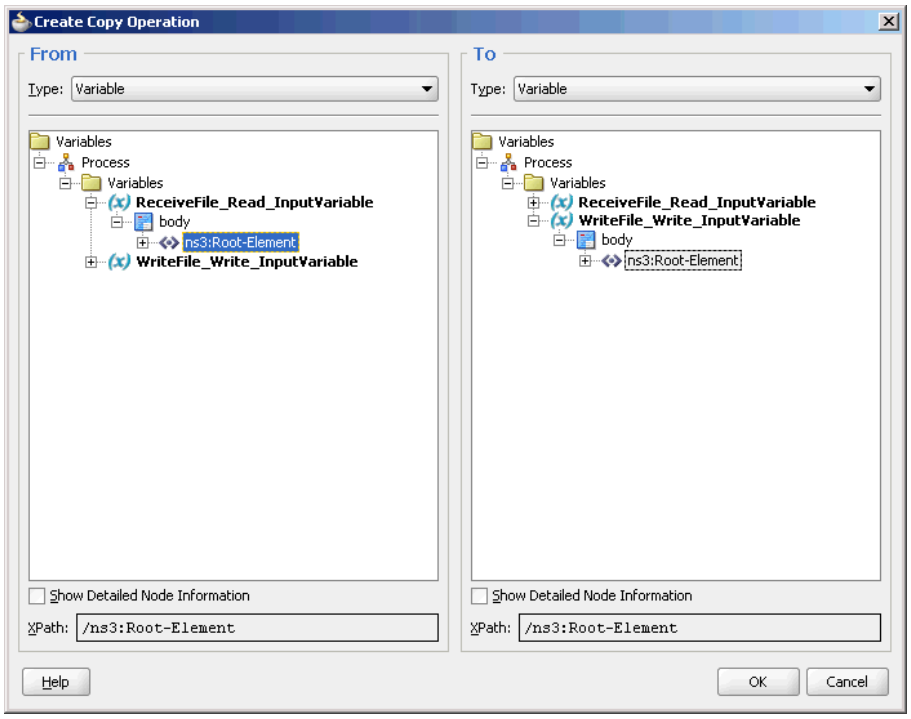
1. Drag and drop an **Assign** activity from the **Components** window in between the **Receive** and **Invoke** activities in the design area.
2. Double-click the **Assign** activity. The **Assign** dialog is displayed.
3. Enter `AssignPayload` in the **Name** field.
4. Click the **Copy Operation** tab. The **Assign** dialog is displayed, as shown in Figure 4-106.

Figure 4-106 The Assign Dialog - Copy Operation Tab



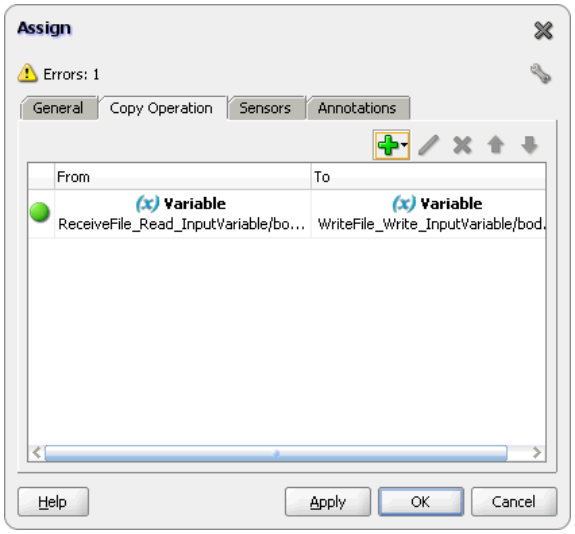
5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
6. Expand the variables in the **From** and **To** panes, as shown in Figure 4-107.

Figure 4-107 The Create Copy Operation Dialog



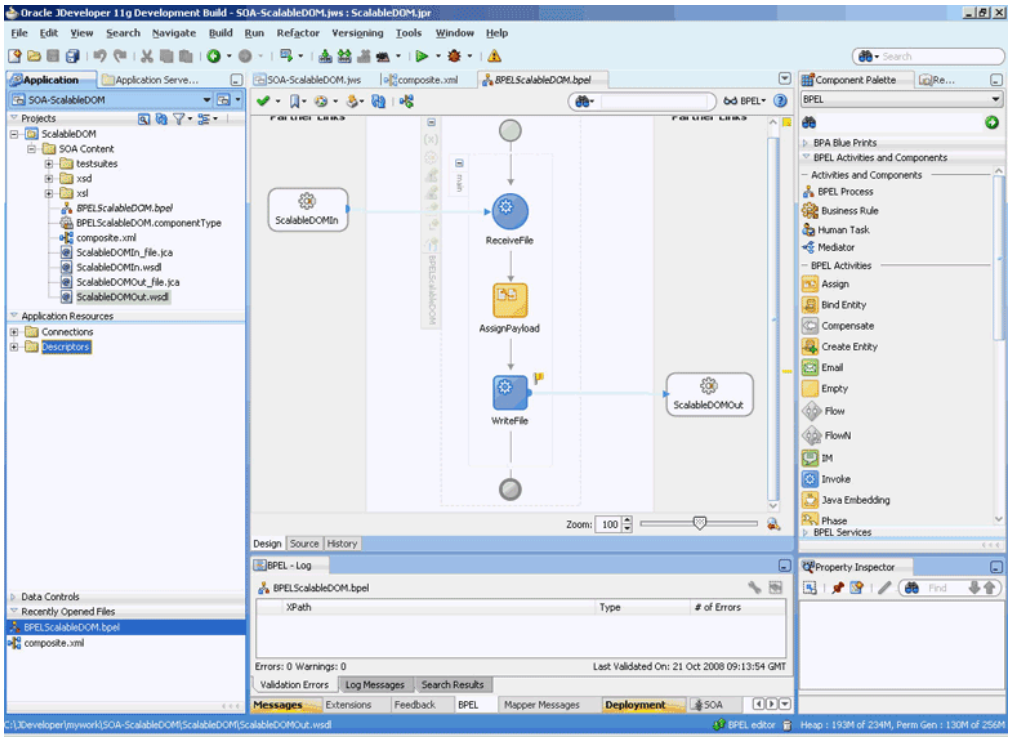
7. Click **OK**. The **Assign** dialog is displayed, as shown in Figure 4-108.

Figure 4-108 The Assign Dialog



8. Click **OK**, the **JDeveloper BPELScalableDOM.bpel** page is displayed, as shown in Figure 4-109.

Figure 4-109 The JDeveloper - BPELScalableDOM.bpel



9. Click **File**, **Save All**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.

2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from .

Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.

2. Copy the `address-csv-large.txt` file to the input directory and ensure it gets processed. Check the output directory to ensure that an output file has been created.

3. Click the SOA composite that you deployed. The **Dashboard** is displayed.

   Note your Instance ID in the **Recent Instances** area.

4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.

5. Click the Instance ID that you noted in Step 3. The **Flow Trace** page is displayed.
6. Click your BPEL process instance. The **Audit Trail** of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow. Additionally, click an activity to view the details of an activity.

Oracle File Adapter Chunked Read

Chunked Read is an Oracle File Adapter feature that uses an invoke activity within a while loop to process the target file. This feature enables you to process arbitrarily large files.

This use case includes the following sections:

- Prerequisites
- Designing the SOA Composite
- Creating the Inbound Service
- Creating the Outbound Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using Fusion Middleware Control Console

Prerequisites

To perform the Oracle File Adapter ChunkRead operation, you require the following files from the `artifacts.zip` file contained in the `Adapters-106FileAdapterChunkedRead` sample:

- `artifacts/schemas/address-csv.xsd`
- `artifacts/schemas/address-fixedLength.xsd`
- `artifacts/xsl/addr1Toaddr2.xsl`
- `artifacts/input/address-csv.txt`

You can obtain the `Adapters-106FileAdapterChunkedRead` sample by accessing the Oracle SOA Sample Code site.

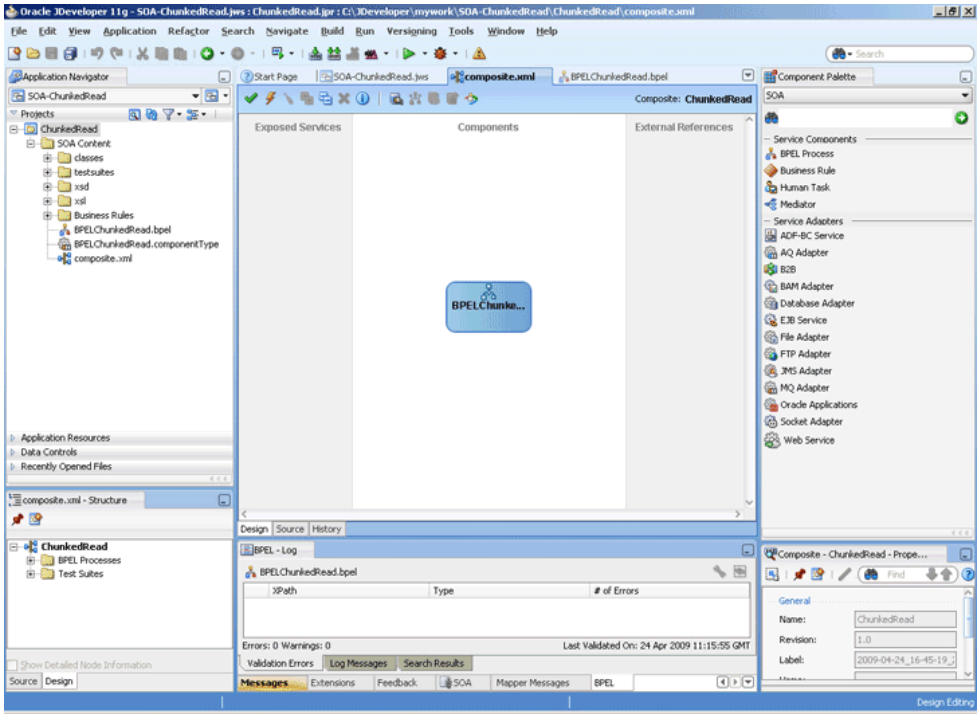
Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite application. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The **Create Generic Application - Name your application** page is displayed.
2. Enter `SOA-ChunkedRead` in the **Application Name** field, and click **Next**. The Create Generic Application - Name your project page is displayed.
3. Enter `ChunkedRead` in the **Project Name** field.

4. In the **Available** list under the **Project Technologies** tab, double-click **SOA** to move it to the **Selected** list.
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the **Composite Template** box, and click **Finish**. The Create BPEL Process - BPEL Process page is displayed.
7. Enter `BPELChunkedRead` in the **Name** field, select **Define Service Later** from the Template box.
8. Click **OK**. The SOA-ChunkedRead application and the ChunkedRead project appears in the design area, as shown in Figure 4-110.

Figure 4-110 The JDeveloper - Composite.xml



9. Copy the **address-csv.xsd** and **address-fixedLength.xsd** files to the xsd directory in your project (see Prerequisites for the location of these files).
10. Copy **addr1Toaddr2.xsl** to the xsl directory of your project (see Prerequisites for the location of these files).

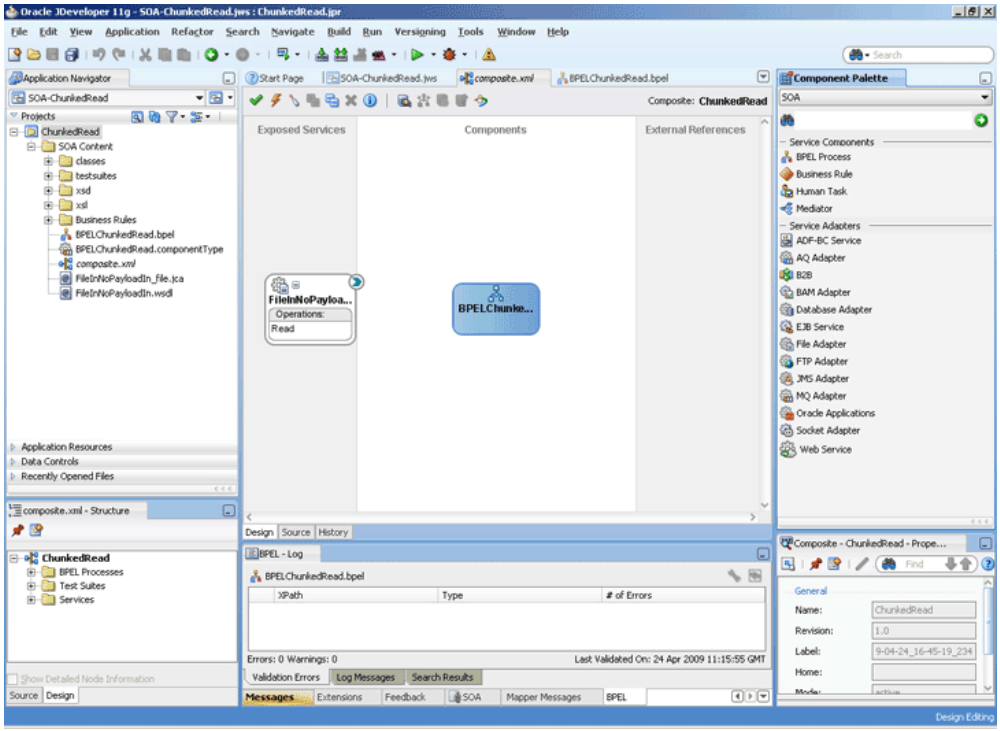
Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory:

1. Drag and drop File Adapter from the **Components** window to the **Exposed Services** swim lane. The Adapter Configuration Wizard **Welcome** page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `FileInNoPayloadIn` in the **Service Name** field.

4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The **Operation** page is displayed.
6. Select **Read File**, check **Do Not Read File Content box**, and then click **Next**. The **File Directories** page is displayed.
7. Enter the physical path for the input directory. Check **Process Files Recursively**.
8. Click **Next**. The **File Filtering** page is displayed.
9. Enter `*.txt` in the **Include Files With Name Pattern** field, click **Next**. The **File Polling** page is displayed.
10. Click **Next**. The **Finish** page is displayed.
11. Click **Finish**. The inbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4-111.

Figure 4-111 The JDeveloper - Composite.xml



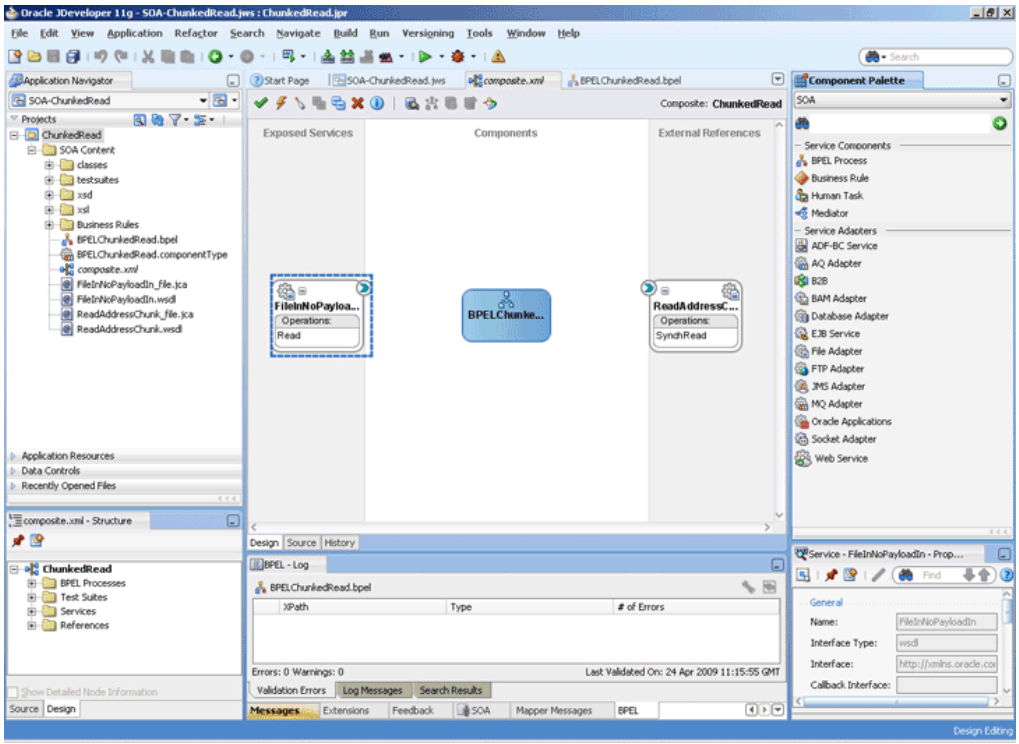
Creating the Outbound Oracle File Adapter Service

Perform the following steps to create an outbound Oracle File Adapter service to write the file from a local directory to the FTP server:

1. Drag and drop File Adapter from the **Components** window to the External References swim lane. The Adapter Configuration Wizard **Welcome** page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `ReadAddressChunk` in the **Service Name** field.

4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The **Operation page** is displayed.
6. Select **Synchronous Read File**, enter `ChunkedRead` in the **Operation Name** field, and then click **Next**. The **File Directories** page is displayed.
7. Enter the physical path for the output directory and select **Delete Files After Successful Retrieval**.
8. Click **Next**. The **File Name** page is displayed.
9. Enter `dummy.txt` in the **File Name** field.
10. Click **Next**. The **Messages** page is displayed.
11. Click **Browse For Schema File** that appears at the end of the URL field. The **Type Chooser** dialog is displayed.
12. Click **Project Schema Files**, **address-csv.xsd**, and **Root-Element**.
13. Click **OK**. The URL field in the **Messages** page is populated with the `address-csv.xsd` file.
14. Click **Next**. The **Finish** page is displayed.
15. Click **Finish**. The outbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-112.

Figure 4-112 The JDeveloper - Composite.xml



16. Manually edit the metadata to incorporate the chunked read feature.

Open `ReadAddressChunk_file.jca` file and modify the metadata as shown the example below.

Example - Modifying Metadata to Incorporate the Chunked Read Feature

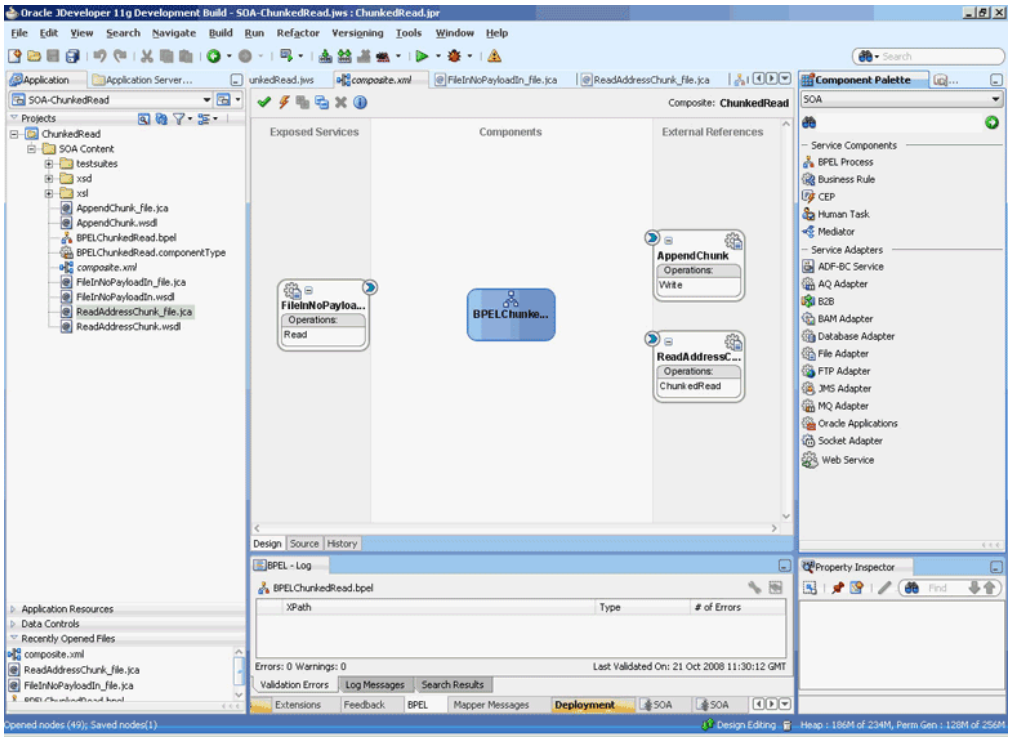
```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="ReadAddressChunk"
                adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks
              /adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter"
                           adapterRef=""/>
  <endpoint-interaction portType="ChunkedRead_ptt"
                    operation="ChunkedRead">
    <interaction-spec
className="oracle.tip.adapter.file.
                       outbound.ChunkedInteractionSpec">
      <property name="PhysicalDirectory"
                          value="/tmp/chunked/in"/>
      <property name="FileName" value="dummy.txt"/>
      <property name="ChunkSize" value="1"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

17. Click **File**, **Save All**.

Add Another Outbound Oracle File Adapter Service

1. Drag and drop File Adapter from the **Components** window to the **External References** swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `AppendChunk` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, enter `Write` in the **Operation Name** field, and then click **Next**. The File Configuration page is displayed.
7. Enter the physical path for the output directory, enter `dummy.txt` in the **File Naming Convention (po_%SEQ%.txt)** and select **Append to Existing File**.
8. Click **Next**. The Messages page is displayed.
9. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
10. Click **Project Schema Files**, **address-fixedLength.xsd**, and **Root-Element**.
11. Click **OK**. The URL field in the Messages page is populated with the `address-fixedLength.xsd` file.
12. Click **Next**. The **Finish** page is displayed.
13. Click **Finish**. The outbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-113.

Figure 4-113 The JDeveloper - Composite.xml



Wiring Services and Activities

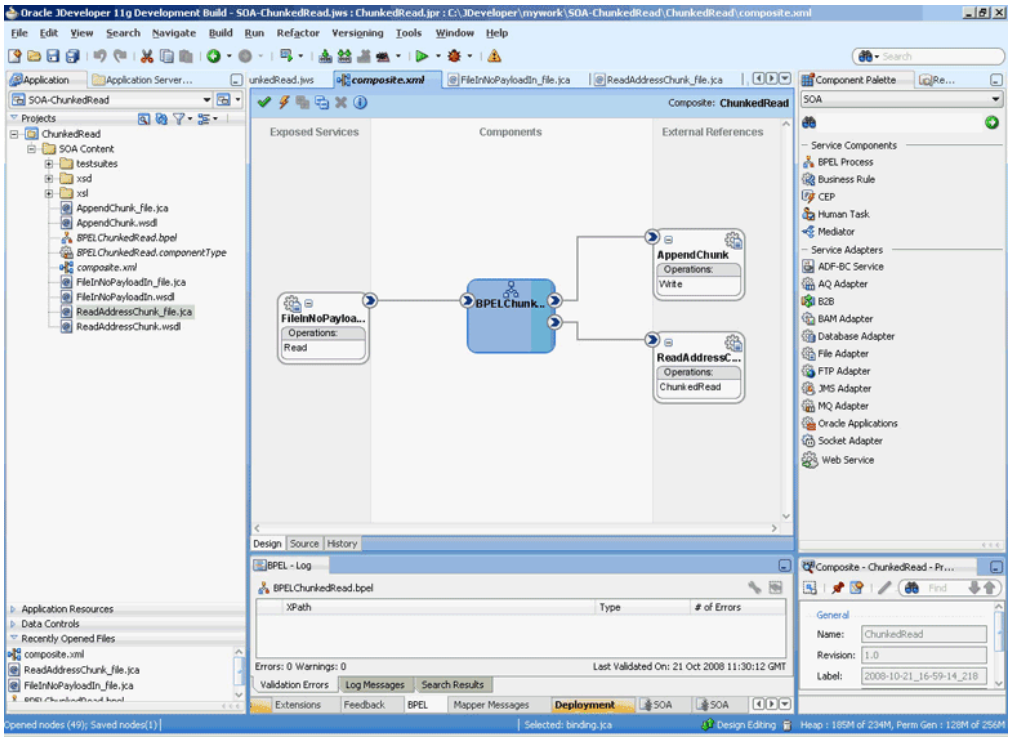
You must assemble or wire the three components that you have created: Inbound adapter service, BPEL process, two Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the FileInNoPayloadIn in the **Exposed Services** area to the drop zone that appears as a green triangle in the BPEL process in the **Components** area.

2. Drag the small triangle in the BPEL process in the **Components** area to the drop zone that appears as a green triangle in the ReadAddressChunk in the **External References** area and also to the green triangle in the AppendChunk in the **External References** area.

   The JDeveloper `composite.xml` appears, as shown in Figure 4-114.

Figure 4-114 The JDeveloper - Composite.xml



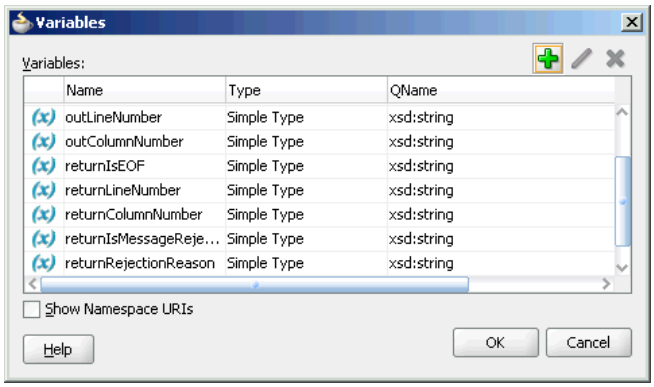
3. Click **File**, **Save All**.

Add a Receive Activity

1. Double-click **BPELChunkedRead**. The BPELChunkedRead.bpel page is displayed.
2. Click the **Variables...** icon represented by (x). The **Variables** dialog is displayed.
3. Click the **Create...** icon. The **Create Variable** dialog is displayed.
4. Create the following variables, as shown in Figure 4-115, for later use:

```
 <variable name="dir" type="xsd:string"/>
    <variable name="file" type="xsd:string"/>
    <variable name="outIsEOF" type="xsd:string"/>
    <variable name="outLineNumber" type="xsd:string"/>
    <variable name="outColumnNumber" type="xsd:string"/>
    <variable name="returnIsEOF" type="xsd:string"/>
    <variable name="returnLineNumber" type="xsd:string"/>
    <variable name="returnColumnNumber" type="xsd:string"/>
    <variable name="returnIsMessageRejected"
type="xsd:string"/>
    <variable name="returnRejectionReason" type="xsd:string"/>
    <variable name="returnNoDataFound" type="xsd:string"/>
```

Figure 4-115 The Variables Dialog

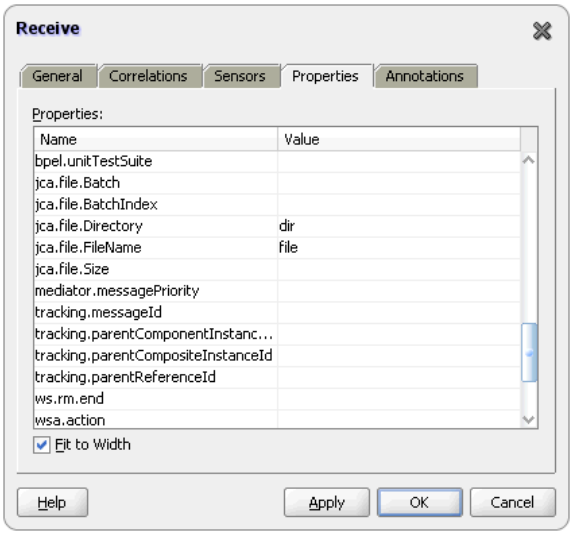


Note:

All variables are Simple Types of type xsd:string.

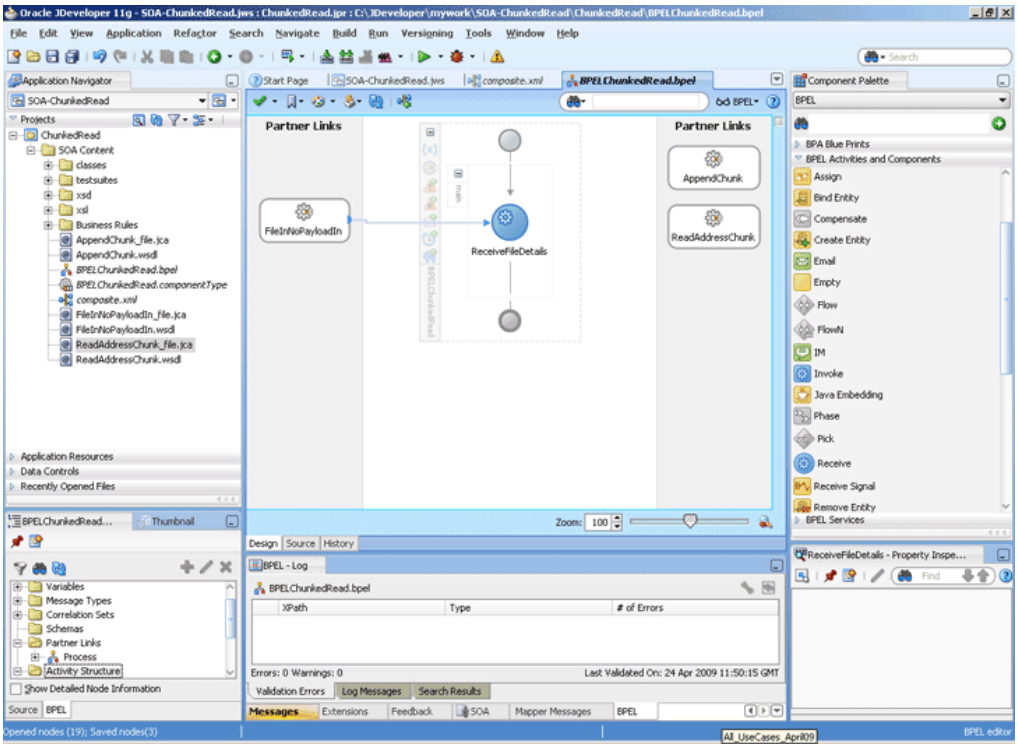
5. Drag and drop a **Receive** activity from the **Components** window to the design area.
6. Double-click the **Receive** activity. The **Receive** dialog is displayed.
7. Enter `ReceiveFileDetails` in the **Name** field.
8. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.
9. Select **FileInNoPayloadIn**, and click **OK**.
10. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog. The **Create Variable** dialog is displayed.
11. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
12. Check **Create Instance**.
13. Click the **Properties** tab. The properties and the corresponding value column is displayed.
14. Select jca.file.Directory property. Double-click in the corresponding value column. The **Adapter Property value** dialog is displayed.
15. Click the **Browse Variables** icon. The Variable XPath Builder dialog is displayed.
16. Expand **Variables**, select **dir**, and then click **OK**. The value of the `jca.file.Directory` is set to `dir`.
17. Repeat the same for `jca.file.FileName` property and set the value to **file**. The **Receive** dialog is displayed, as shown in Figure 4-116.

Figure 4-116 The Receive Dialog - Adapters Tab



18. Click **OK**. The JDeveloper BPELChunkedRead.bpel page appears, as shown in Figure 4-117.

Figure 4-117 The JDeveloper - BPELChunkedRead.bpel



Add an Assign Activity

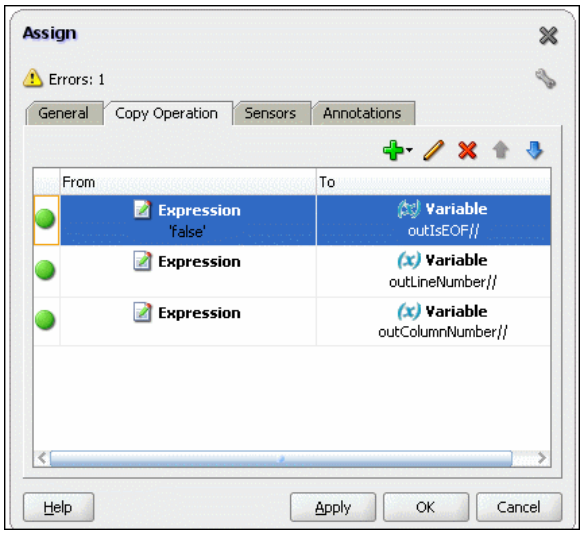
1. Drag and drop an **Assign** activity from the **Components** window after the Receive activity in the design area.
2. Double-click the **Assign** activity. The **Assign dialog** is displayed.
3. Enter `AssignChunkedRead` in the **Name** field.

4. Click the **Copy Operation** tab. The **Assign dialog** is displayed, as shown in Figure 4-106.

5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.

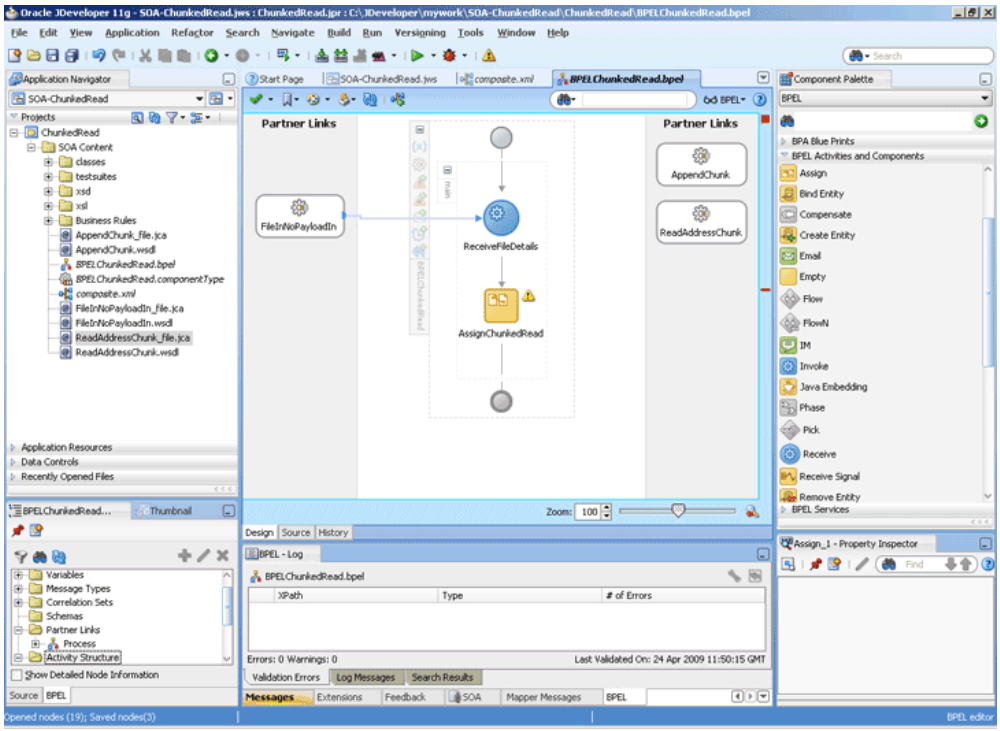
6. Set the default values for the headers, as shown in Figure 4-118.

Figure 4-118 The Assign Dialog



7. Click **OK**, the JDeveloper BPELChunkedRead.bpel page is displayed, as shown in Figure 4-119.

Figure 4-119 The JDeveloper - BPELChunkedRead.bpel

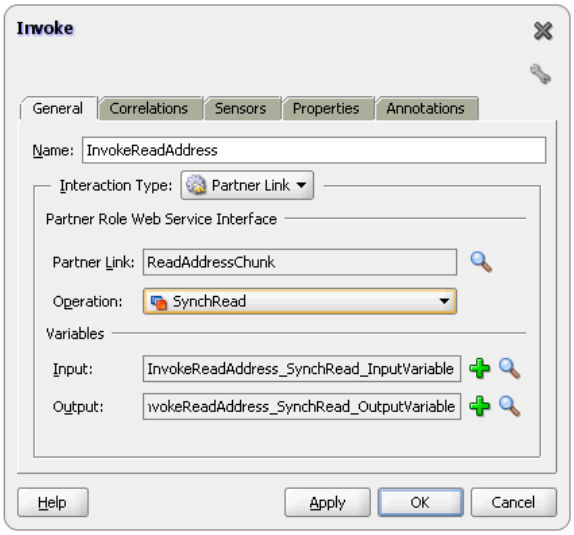


8. Click **File**, **Save All**.

Add an Invoke Activity

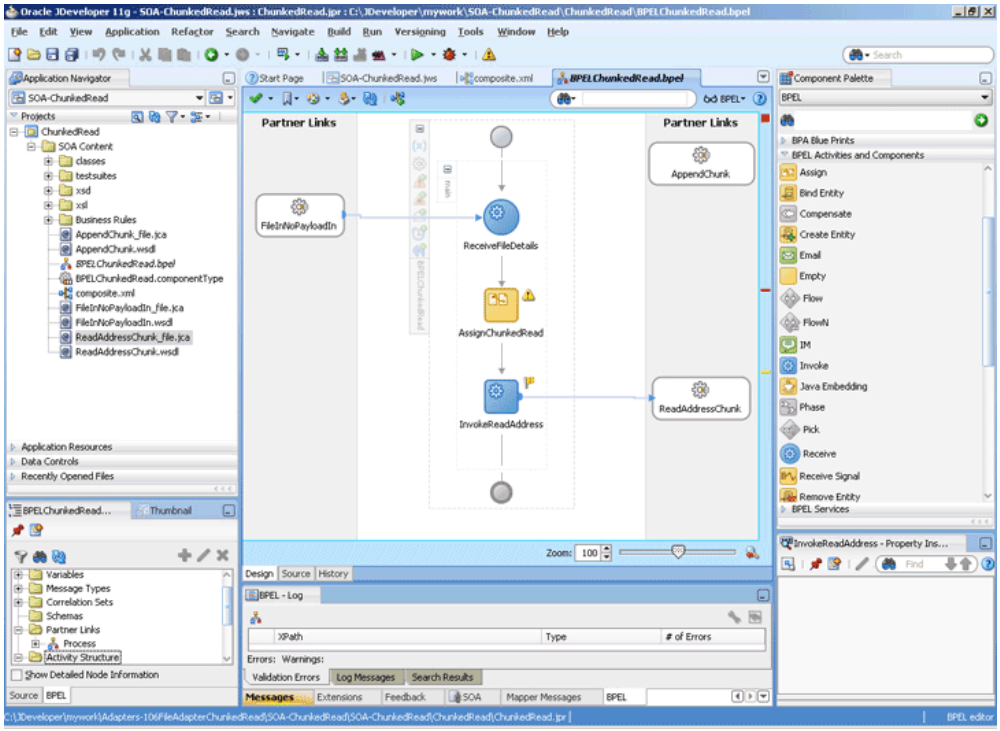
1. Drag and drop an **Invoke** activity below the Assign Activity from the **Components** window to the design area.
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `InvokeReadAddress` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.
5. Select **ReadAddressChunk**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The **Create Variable** dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name. The **Invoke** dialog is displayed with input variable populated.
8. Repeat the same to select the output variable. The **Invoke** dialog is displayed, as shown in Figure 4-120.

Figure 4-120 The Invoke Dialog



9. Click **OK**. The JDeveloper **BPELChunkedRead.bpel** page appears, as shown in Figure 4-121.

Figure 4-121 The JDeveloper - BPELChunkedRead.bpel



10. Click the **Source** tab for the **BPELChunkedRead.bpel** page, and add the following properties for the Invoke activity that you just created:

Example - Properties to Add to the Invoke Activity

```
<bpelx:inputProperty name="jca.file.Directory"
                variable="dir"/>
<bpelx:inputProperty name="jca.file.FileName"
                  variable="file"/>
<bpelx:inputProperty name="jca.file.LineNumber"
                               ="outLineNumber"/>
<bpelx:inputProperty name="jca.file.ColumnNumber"
variable="outColumnNumber"/>
<bpelx:inputProperty name="jca.file.IsEOF"
                          variable="outIsEOF"/>
<bpelx:outputProperty name="jca.file.LineNumber"
variable="returnLineNumber"/>
<bpelx:outputProperty name="jca.file.ColumnNumber"
variable="returnColumnNumber"/>
<bpelx:outputProperty name="jca.file.IsEOF"
                  variable="returnIsEOF"/>
<bpelx:outputProperty
               name="jca.file.IsMessageRejected"
variable="returnIsMessageRejected"/>
<bpelx:outputProperty
              name="jca.file.RejectionReason"
variable="returnRejectionReason"/>
<bpelx:outputProperty name="jca.file.NoDataFound"
variable="returnNoDataFound"/>
```

Note that this is different for BPEL 2.0. For that version of BPEL, this would appear as shown in the example below.

Example - Properties to Add for Invoke using BPEL 2.0

```
<bpelx:toProperties>
<bpelx:toProperty name="jca.file.Directory" variable="dir"/>
<bpelx:toProperty name="jca.file.FileName" variable="file"/>
<bpelx:toProperty name="jca.file.LineNumber"
                    variable="outLineNumber"/>
<bpelx:toProperty name="jca.file.ColumnNumber"
                 variable="outColumnNumber"/>
<bpelx:toProperty name="jca.file.IsEOF"
                variable="outIsEOF"/>
  </bpelx:toProperties>
  <bpelx:fromProperties>
<bpelx:fromProperty name="jca.file.LineNumber"
                  variable="returnLineNumber"/>
<bpelx:fromProperty name="jca.file.ColumnNumber"
variable="returnColumnNumber"/>
<bpelx:fromProperty name="jca.file.IsEOF"
                  variable="returnIsEOF"/>
<bpelx:fromProperty
              name="jca.file.IsMessageRejected"
variable="returnIsMessageRejected"/>
<bpelx:fromProperty name="jca.file.RejectionReason"
variable="returnRejectionReason"/>
<bpelx:fromProperty
              name="jca.file.NoDataFound"
variable="returnNoDataFound"/>
  </bpelx:fromProperties
```

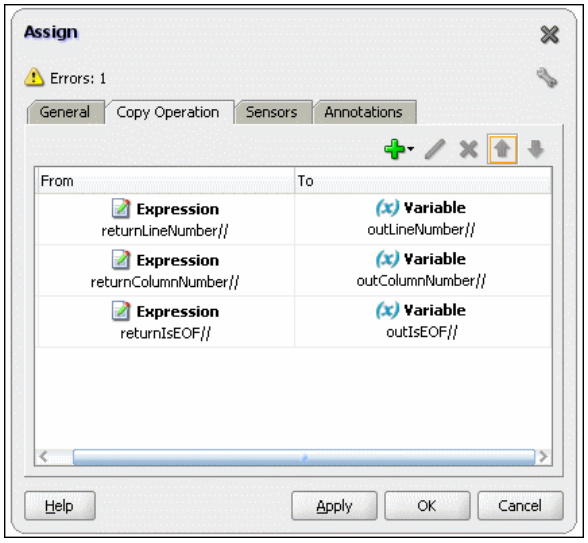
The Invoke activity appears as shown in the example below.

Example - Invoke Activity

```
<invoke name="InvokeReadAddress"
inputVariable="InvokeReadAddress_SynchRead_InputVariable"
outputVariable="InvokeReadAddress_SynchRead_OutputVariable"
partnerLink="ReadAddressChunk" portType="ns3:SynchRead_ptt"
operation="SynchRead">
<bpelx:inputProperty name="jca.file.Directory" variable="dir"/>
<bpelx:inputProperty name="jca.file.FileName" variable="file"/>
<bpelx:inputProperty name="jca.file.LineNumber"
              variable="outLineNumber"/>
<bpelx:inputProperty name="jca.file.ColumnNumber"
variable="outColumnNumber"/>
<bpelx:inputProperty name="jca.file.IsEOF"
               variable="outIsEOF"/>
<bpelx:outputProperty name="jca.file.LineNumber"
variable="returnLineNumber"/>
<bpelx:outputProperty name="jca.file.ColumnNumber"
variable="returnColumnNumber"/>
<bpelx:outputProperty name="jca.file.IsEOF"
                  variable="returnIsEOF"/>
<bpelx:outputProperty name="jca.file.IsMessageRejected"
variable="returnIsMessageRejected"/>
<bpelx:outputProperty name="jca.file.RejectionReason"
variable="returnRejectionReason"/>
<bpelx:outputProperty name="jca.file.NoDataFound"
variable="returnNoDataFound"/>
</invoke>
```

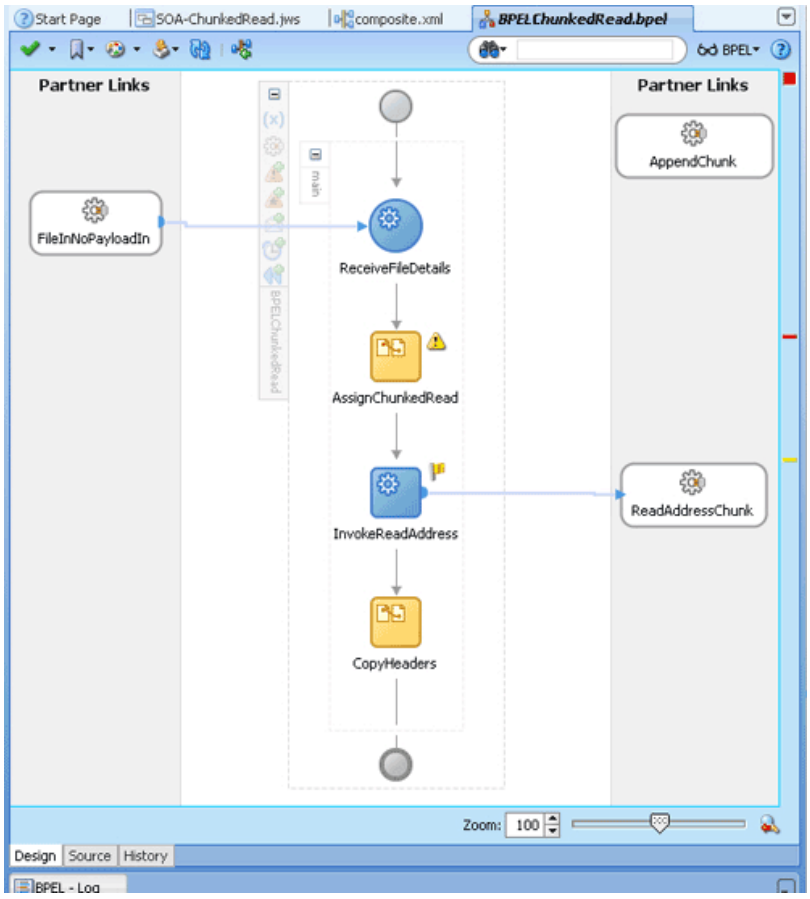
11. Add an assign activity called `CopyHeaders`, as given in Add an Assign Activity, to copy the return parameters from the invoke activity. The Assign dialog is displayed, as shown in Figure 4-122.

Figure 4-122 The Assign Dialog



12. Click **OK**. The JDeveloper BPELChunkedRead.bpel page is displayed, as shown in Figure 4-123.

Figure 4-123 The JDeveloper - BPELChunkedRead.bpel



Add a Switch Activity

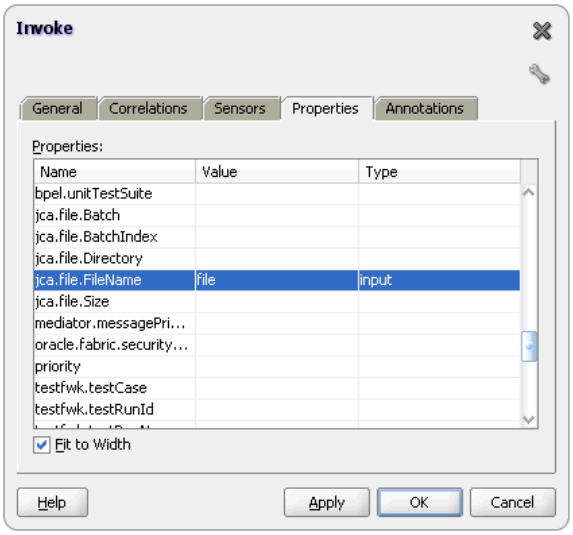
1. Drag and drop a **Switch** activity below the CopyHeaders Assign activity.

2. Double-click <case> in the **Switch** activity. The **Switch Case** dialog is displayed.
3. Enter `DATA FOUND` in the **Name** field and select the `returnNoDataFound` expression in the Expression box. The **Switch Case** dialog is displayed, as shown in Figure 4-124.

Figure 4-124 The Switch Case Dialog

4. Drag and drop an **Invoke** activity in the <Case DATA FOUND> for Switch Activity.
5. Double-click the **Invoke** activity. The Invoke dialog is displayed.
6. Enter `InvokeAppend` in the **Name** field.
7. Select AppendChunk in the Partner Link field.
8. Click the **Automatically Create Input Variable** icon to the right of the **Input** variable field in the Invoke dialog. The **Create Variable** dialog is displayed.
9. Select the default variable name and click **OK**. The Variable field is populated with the default variable name. The Invoke dialog is displayed with input variable populated.
10. Click the **Properties** tab and select file variable, as shown in Figure 4-125.

Figure 4-125 The Invoke Dialog

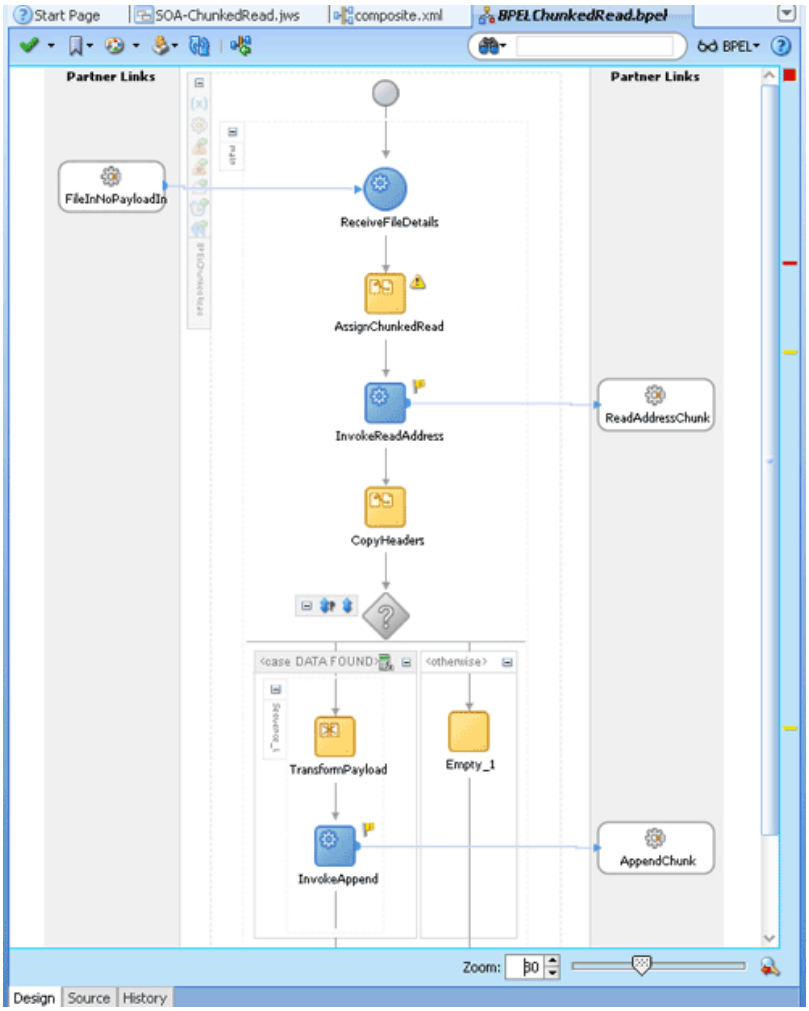


11. Click **OK**.

Add a Transform Activity

1. Drag and drop a **Transform** activity in the <case DATA FOUND> section just before the **InvokeAppend** activity.
2. Double-click the **Transform** activity.
3. Enter `TransformPayload` in the **Name** field.
4. Click the **Transformation** tab.
5. Click the **Create...** icon. The **Source Variable** dialog is displayed.
6. Select **InvokeReadAddress_SyncRead_InputVariable**, and click **OK**.
7. Select **InvokeAppend_Write_InputVariable** from the Target Variable list.
8. Click **Browse** at the end of the Mapper File field, and select the **addr1Toaddr2.xsl** file.
9. Click **OK**.
10. Drag and drop an **Empty** activity in the <otherwise> section in the Switch activity. The **BPELChunkedRead.bpel** page is displayed, as shown in Figure 4-126.

Figure 4-126 The JDeveloper - BPELChunkedRead.bpel



11. Click **File**, **Save All**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from .

Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.

2. Copy the `address-csv.txt` file to the input directory (see Prerequisites for the location of this file) and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the **Recent Instances** area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The **Flow Trace** page is displayed.
6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow. Additionally, click an activity to view the details of an activity.

Oracle File Adapter Read File As Attachments

This is an Oracle File Adapter feature to opaquely copy or move large amount of data, from a source directory on your file system to a destination directory, as attachments. For example, you can transfer large MS Word documents, images, and PDFs without processing their content within the composite application. The read file as attachment feature is available only when the Read File option is chosen.

This use case demonstrates the ability of the Oracle File Adapter to process a large `*.doc` file as an attachment. This feature of reading files as attachments is very similar to `Opaque` translation. However, attachments can be of the order of gigabytes depending on database limitations.

- Prerequisites
- Designing the SOA Composite
- Creating the Inbound Service
- Creating the Outbound Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using Fusion Middleware Control Console

Prerequisites

To use the Oracle File Adapter read file as attachments feature, you require a large MS Word document (`*.doc` file).

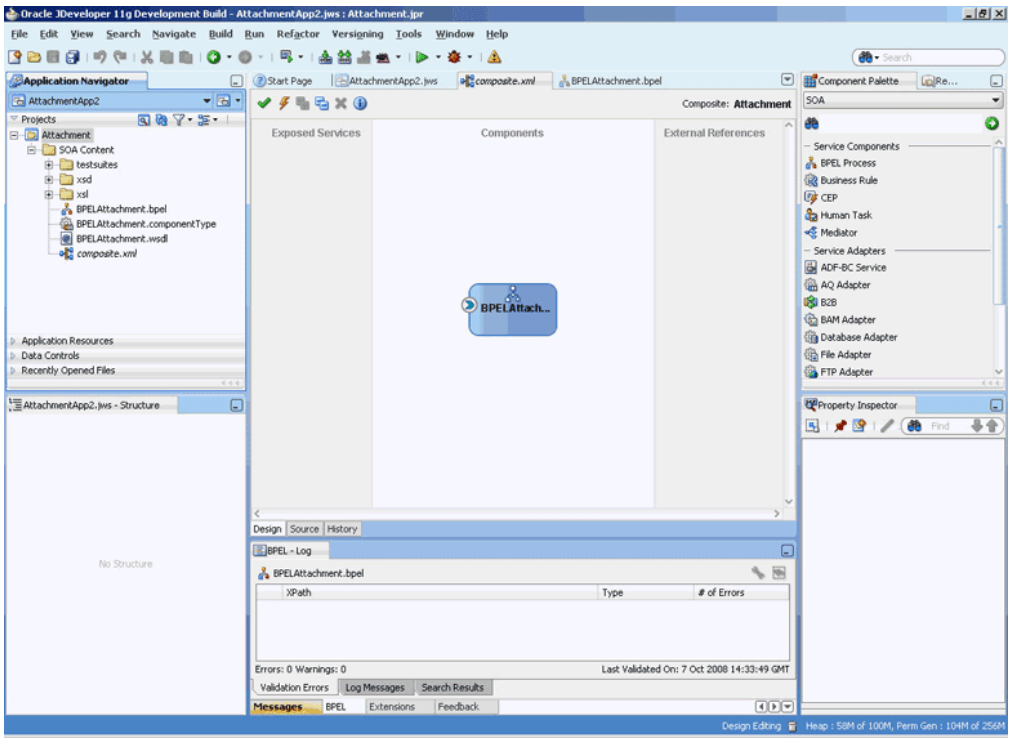
Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The **Create Generic Application - Name your application** page is displayed.

2. Enter `AttachmentApp` in the **Application Name** field, and click **Next**. The **Create Generic Application - Name your project** page is displayed.
3. Enter `Attachment` in the **Project Name** field.
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The **Configure SOA settings** dialog appears.
6. Select **Composite With BPEL** in the **Composite Template** box, and click **Finish**. The **Create BPEL Process - BPEL Process** page is displayed.
7. Enter `BPELAttachment` in the **Name** field, select **Define Service Later** from the Template list.
8. Click **OK**. The `AttachmentApp` application and the `Attachment` project appear in the design area, as shown in Figure 4-127.

Figure 4-127 The JDeveloper - Composite.xml



Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read a large file from a local directory:

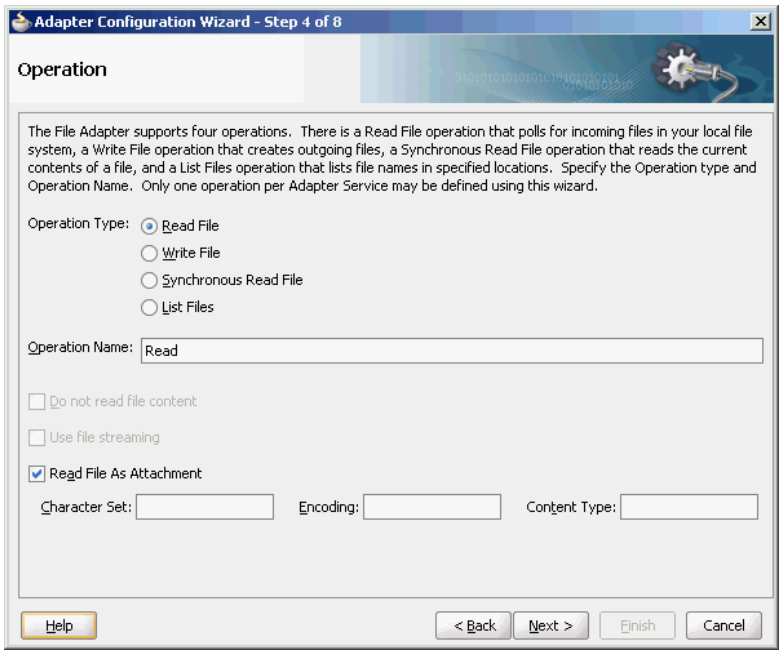
1. Drag and drop File Adapter from the **Components** window to the Exposed Services swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `AttachmentIn` in the **Service Name** field.

4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Read File** as the Operation Type and select **Read File As Attachment**, as shown in Figure 4-128, and then click **Next**. The **File Directories** page is displayed.

Note:

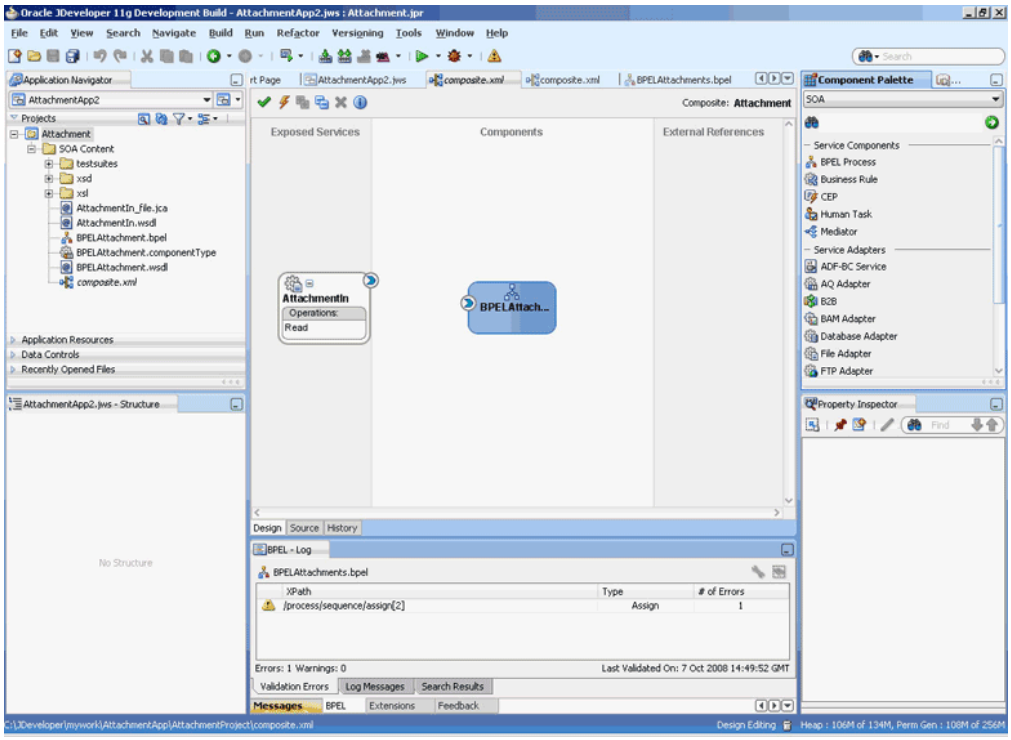
You must ignore Character Set, Encoding, and Content Type fields. These fields must be populated with values only if you are using third-party applications that must read this attachment. The attachment in this use case is finally consumed by an outbound Oracle File Adapter, hence these values are not required.

Figure 4-128 The Adapter Configuration Wizard Operation Page



7. Enter the physical path for the input directory, as shown in Figure 4-67 and click **Next**. The **File Filtering** page is displayed.
8. Enter `*.doc` in the **Include Files With Name Pattern** field, as shown in Figure 4-68.
9. Click **Next**. The **File Polling** page is displayed.
10. Click **Next**. The **Finish** page is displayed.
11. Click **Finish**. The inbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-129.

Figure 4-129 The JDeveloper - Composite.xml

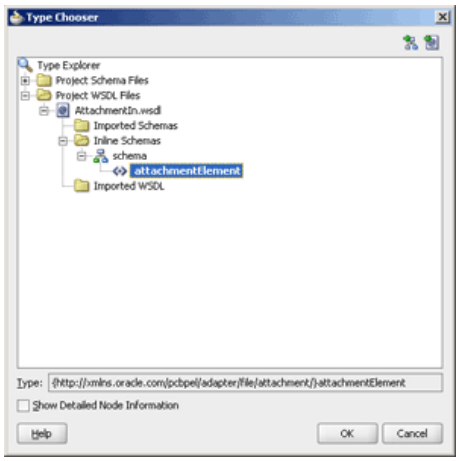


Creating the Outbound Oracle File Adapter Service

Perform the following steps to create an outbound Oracle File Adapter service to write the file from a local directory to the FTP server:

1. Drag and drop File Adapter from the **Components** window to the **External References** swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `AttachmentOut` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, and click **Next**. The File Configuration page is displayed.
7. Enter the physical path for the output directory and enter `attachment_%SEQ%.doc` in the **File Naming Convention(po_%SEQ%.txt)** field, as shown in Figure 4-72.
8. Click **Next**. The **Messages** page is displayed.
9. Click **Browse For Schema File** that appears at the end of the URL field. The **Type Chooser** dialog is displayed.
10. Click **Project WSDL Files**, **AttachmentIn.wsdl**, **Inline Schemas**, and **attachmentElement**, as shown in Figure 4-130.

Figure 4-130 The Type Chooser Dialog

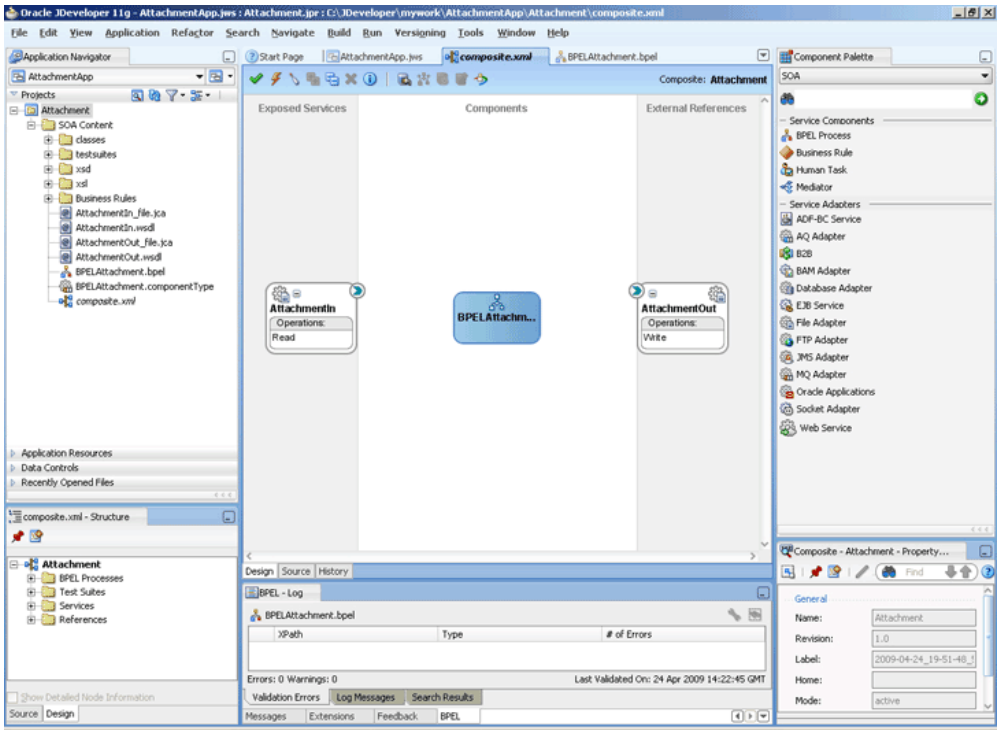


11. Click **OK**. The URL field in the **Messages** page is populated with AttachmentIn.wsdl.

12. Click **Next**. The **Finish** page is displayed.

13. Click **Finish**. The outbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-131.

Figure 4-131 The JDeveloper - Composite.xml



Wiring Services and Activities

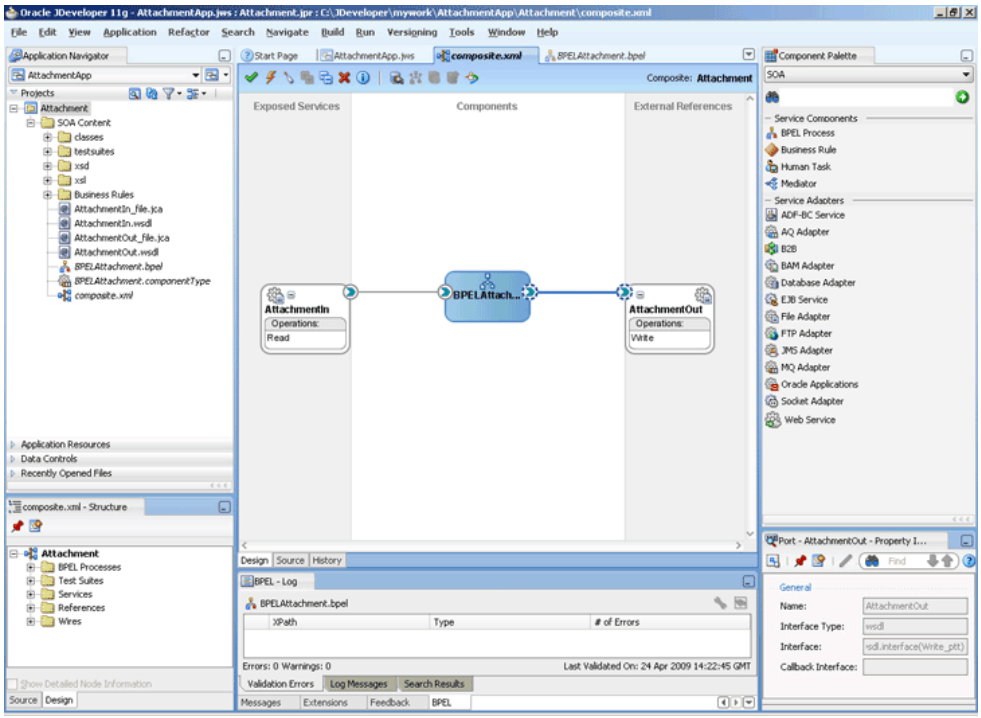
You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the AttachmentIn in the **Exposed Services** area to the drop zone that appears as a green triangle in the BPEL process in the **Components** area.

2. Drag the small triangle in the BPEL process in the **Components** area to the drop zone that appears as a green triangle in the AttachmentOut in the **External References** area.

   The JDeveloper `composite.xml` appears, as shown in Figure 4-132.

Figure 4-132 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.

Add a Receive Activity

1. Double-click **BPELAttachment**. The BPELAttachment.bpel page is displayed.

2. Drag and drop a **Receive** activity from the **Components** window to the design area.

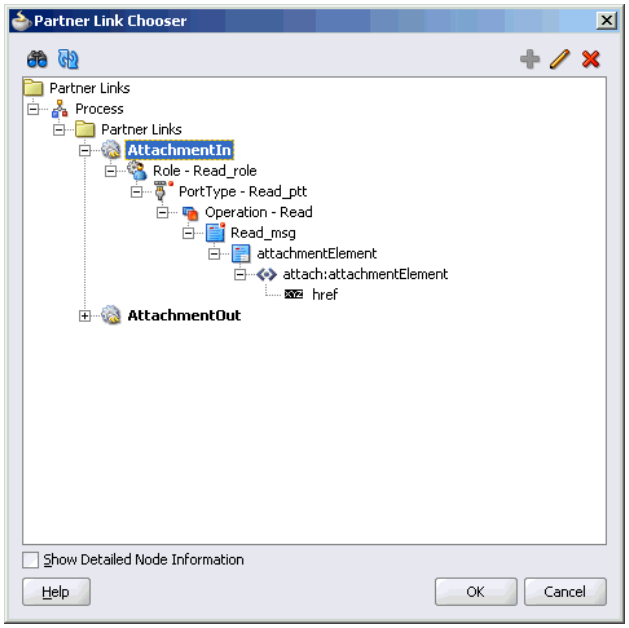
3. Double-click the **Receive** activity. The **Receive** dialog is displayed.

4. Enter `ReceiveInput` in the **Name** field.

5. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.

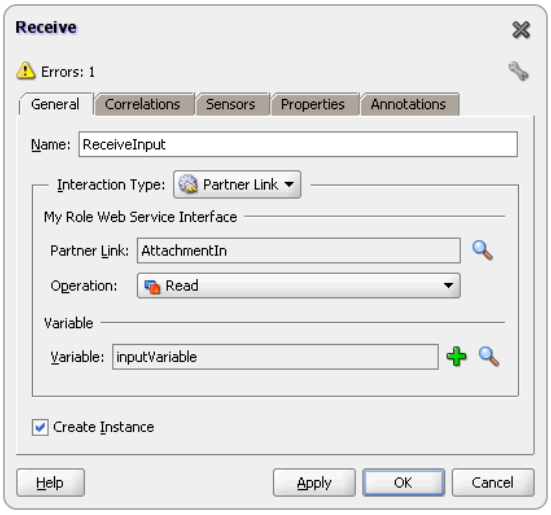
6. Select **AttachmentIn**, as shown in Figure 4-133 and click **OK**.

Figure 4-133 The Partner Link Chooser Dialog



7. Click the **Auto-Create Variable** icon to the right of the Variable field in the **Receive** dialog, as shown in Figure 4-134. The **Create Variable** dialog is displayed.

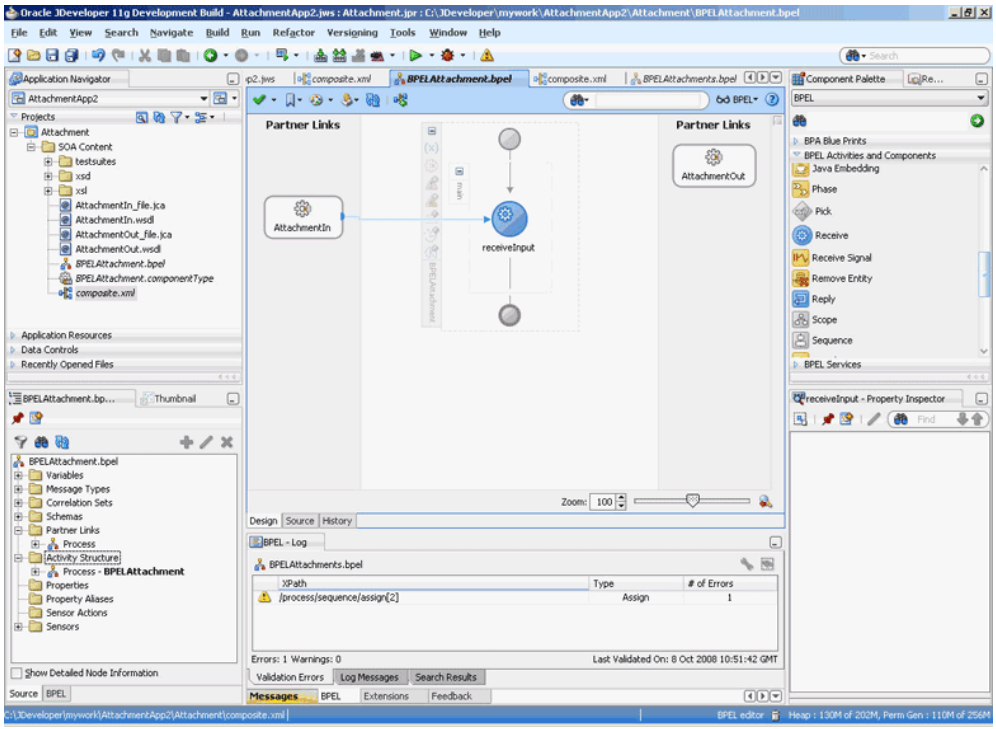
Figure 4-134 The Receive Dialog



8. Select the default variable name and click **OK**. The **Variable** field is populated with the default variable name.

9. Check **Create Instance**, and click **OK**. The JDeveloper **BPELAttachment.bpel** page appears, as shown in Figure 4-135.

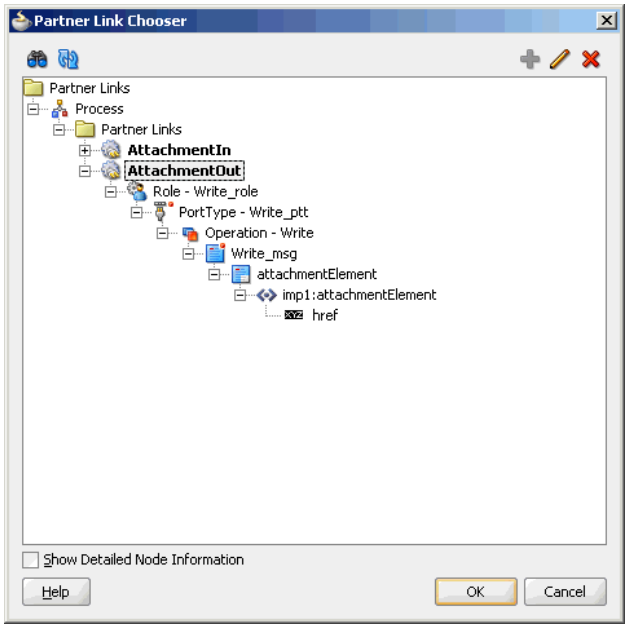
Figure 4-135 The JDeveloper - BPELXMLDebatching.bpel



Add an Invoke Activity

1. Drag and drop an **Invoke** activity from the **Components** window to the design area.
2. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.
3. Enter `Write_Attachment` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.
5. Select **AttachmentOut**, as shown in Figure 4-136, and click **OK**.

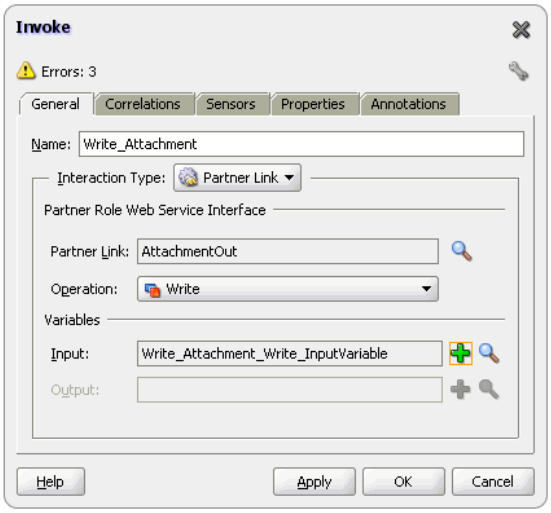
Figure 4-136 The Partner Link Chooser Dialog



6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.

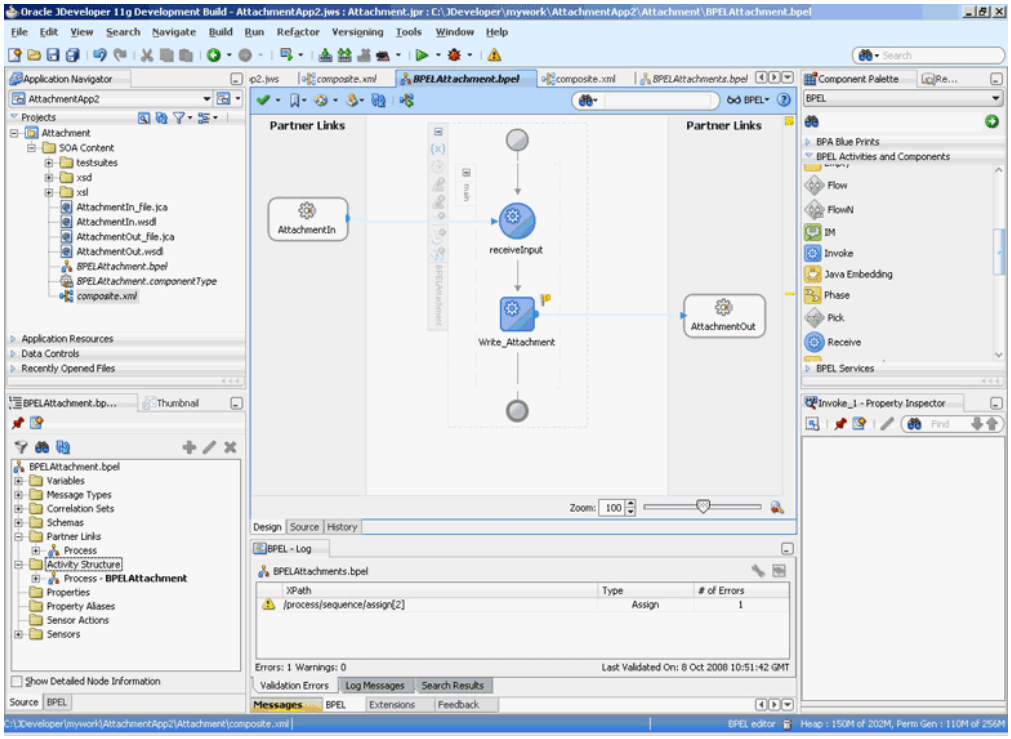
7. Select the default variable name and click **OK**. The **Variable** field is populated with the default variable name. The **Invoke** dialog is displayed, as shown in Figure 4-137.

Figure 4-137 The Invoke Dialog



8. Click **OK**. The JDeveloper **BPELAttachment.bpel** page appears, as shown in Figure 4-138.

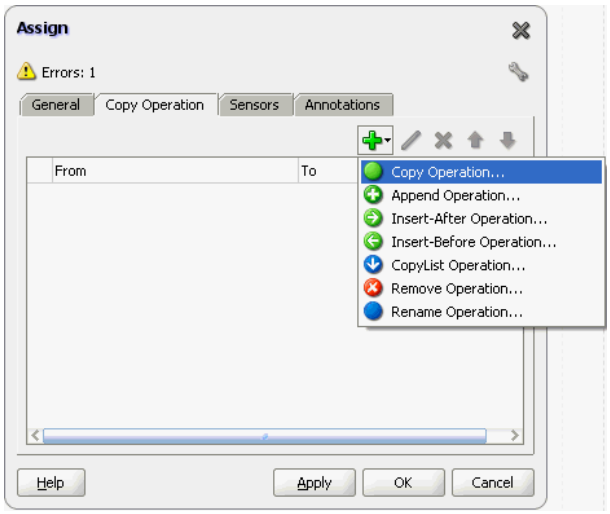
Figure 4-138 The JDeveloper - BPELXMLDebatching.bpel



Add an Assign Activity

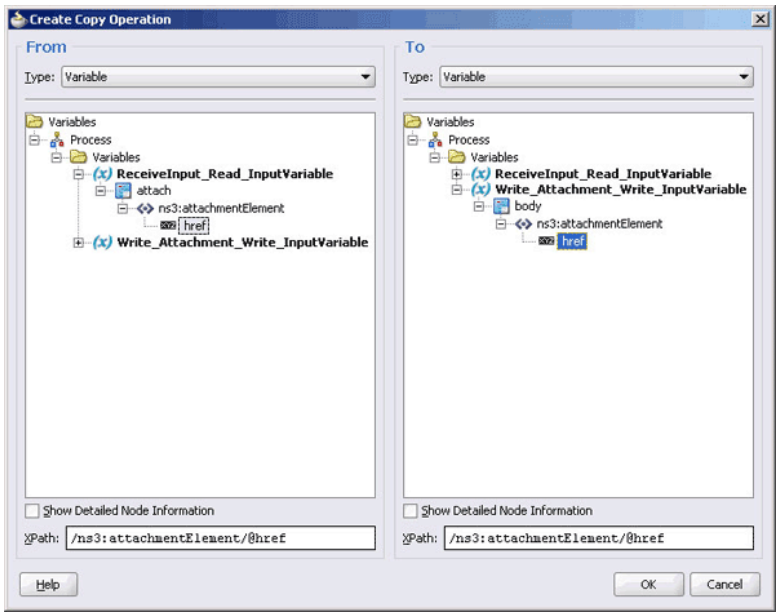
1. Drag and drop an **Assign** activity from the **Components** window in between the Receive and Invoke activities in the design area.
2. Double-click the **Assign** activity. The **Assign** dialog is displayed.
3. Enter `AssignReference` in the **Name** field.
4. Click the **Copy Operation** tab. The **Assign** dialog is displayed, as shown in Figure 4-139.

Figure 4-139 The Assign Dialog - Copy Operation Tab



5. Select **Copy Operation**. The **Create Copy Operation** dialog is displayed.
6. Expand the variables in the From and To panes, as shown in Figure 4-140.

Figure 4-140 The Create Copy Operation Dialog

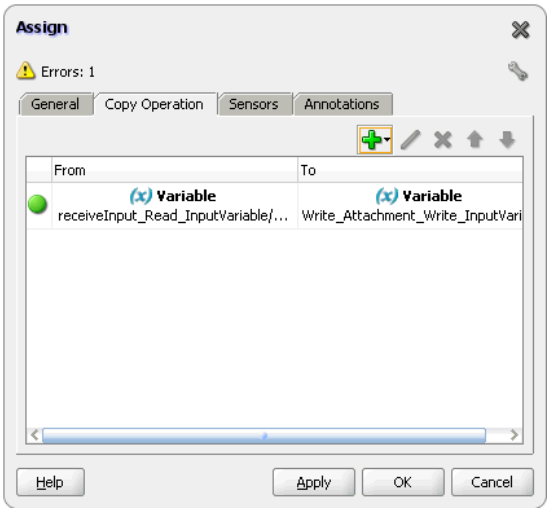


Note:

When variables are defined by reference to an element, both the source and the target must be the same element.

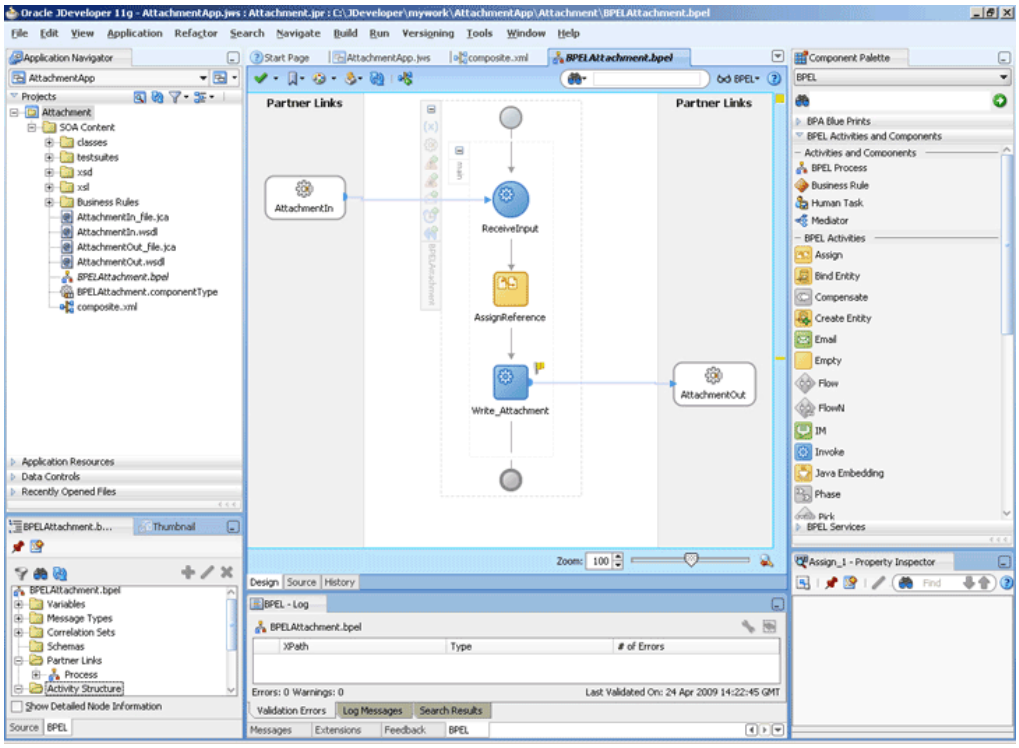
7. Click **OK**. The **Assign** dialog is displayed, as shown in Figure 4-141.

Figure 4-141 The Assign Dialog



8. Click **OK**, the JDeveloper BPELAttachment.bpel page is displayed, as shown in Figure 4-142.

Figure 4-142 The JDeveloper - BPELScalableDOM.bpel



9. Click **File**, **Save All**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from .

Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the `attachment.doc` file to the input directory (see Prerequisites for details) and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The **Dashboard** is displayed.

   Note your Instance ID in the **Recent Instances** area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.

5. Click the Instance ID that you noted in Step 3. The **Flow Trace** page is displayed.
6. Click your BPEL process instance. The **Audit Trail** of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow. Additionally, click an activity to view the details of an activity.

Oracle File Adapter File Listing

This is an Oracle File Adapter feature that lets you use an invoke activity to retrieve a list of files from a target directory. This list of files is returned as an XML document and contains information such as file name, directory name, file size, and last modified time.

This use case includes the following sections:

- Prerequisites
- Designing the SOA Composite
- Creating the Outbound Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using Fusion Middleware Control Console

Prerequisites

To perform Oracle File Adapter Listing, you require `*.txt` files. You must create and save the `*.txt` files in the target directory.

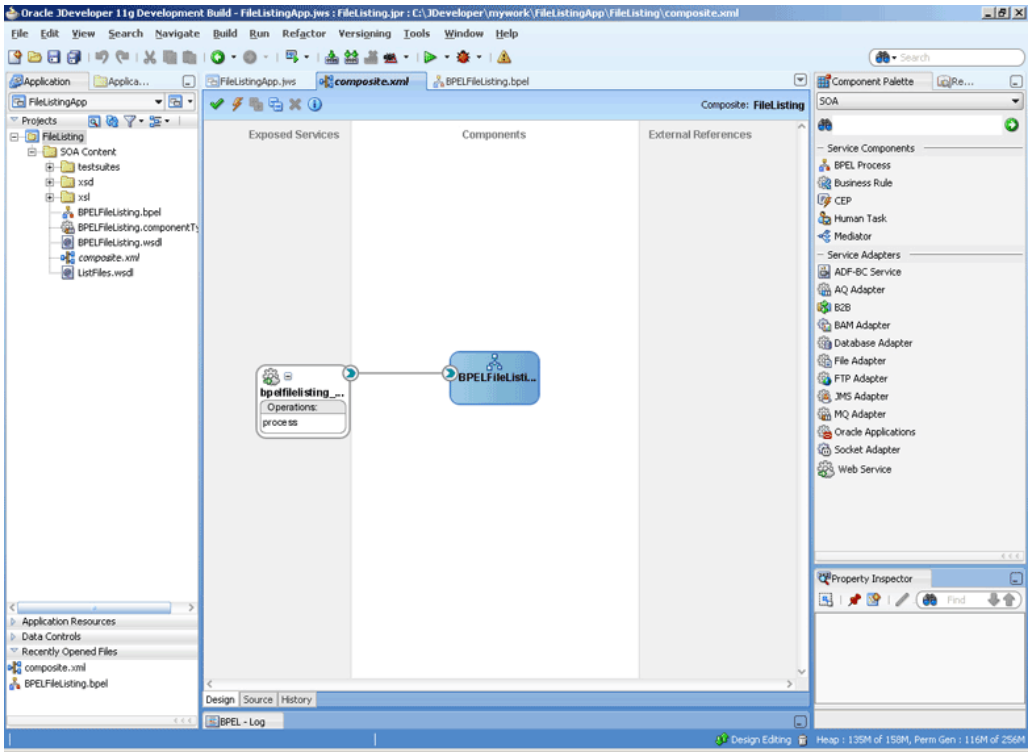
Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `FileListingApp` in the **Application Name** field, and click **Next**. The Create Generic Application - Name your project page is displayed.
3. Enter `FileListing` in the **Project Name** field.
4. In the Available list under the **Project Technologies** tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The **Configure SOA settings** dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The **Create BPEL Process - BPEL Process** page is displayed.
7. Enter `BPELFileListing` in the **Name** field, select **One Way BPEL Process** from the Template box.

8. Click **OK**. The FileListingApp application and the FileListing project appears in the design area, as shown in Figure 4-143.

Figure 4-143 The JDeveloper - Composite.xml



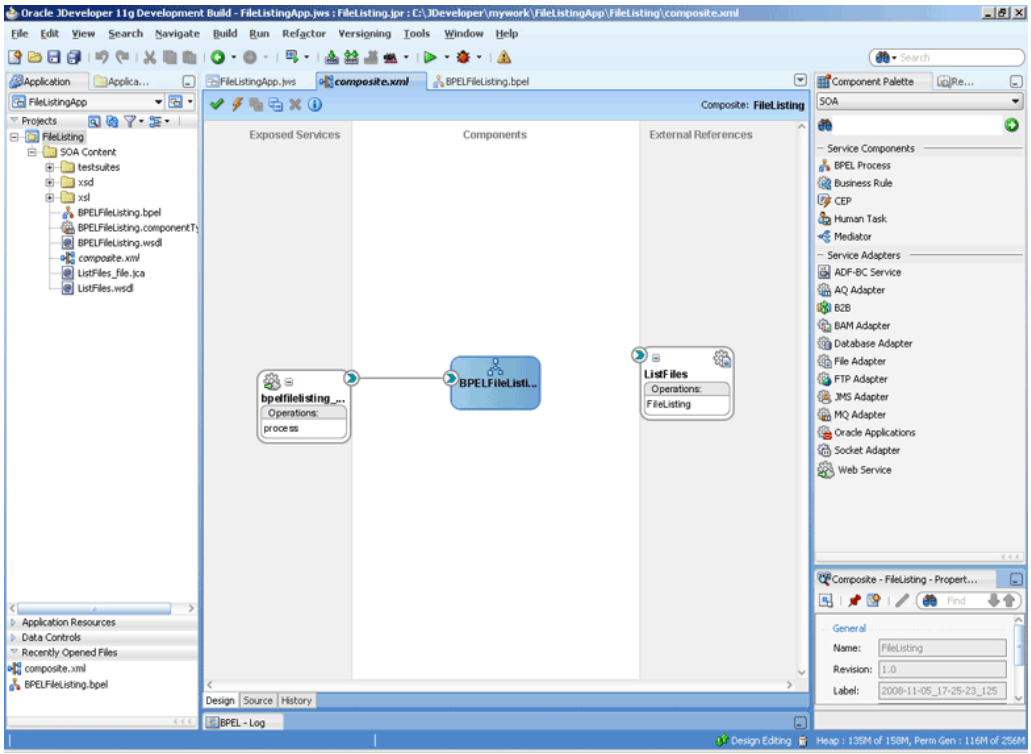
Creating the Outbound Oracle File Adapter Service

Perform the following steps to create an outbound Oracle File Adapter service to list the file from a target directory:

1. Drag and drop File Adapter from the **Components** window to the External References swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `ListFiles` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **List Files**, enter `FileListing` in the **Operation Name** field, and then click **Next**. The File Directories page is displayed.
7. Enter the physical path for the input directory, as shown in Figure 4-67.
8. Click **Next**. The File Filtering page is displayed.
9. Enter `*.txt` in the **Include Files with Name Pattern** field.
10. Click **Next**. The **Finish** page is displayed.

11. Click **Finish**. The outbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-144.

Figure 4-144 The JDeveloper - Composite.xml



12. Click **File**, **Save All**.

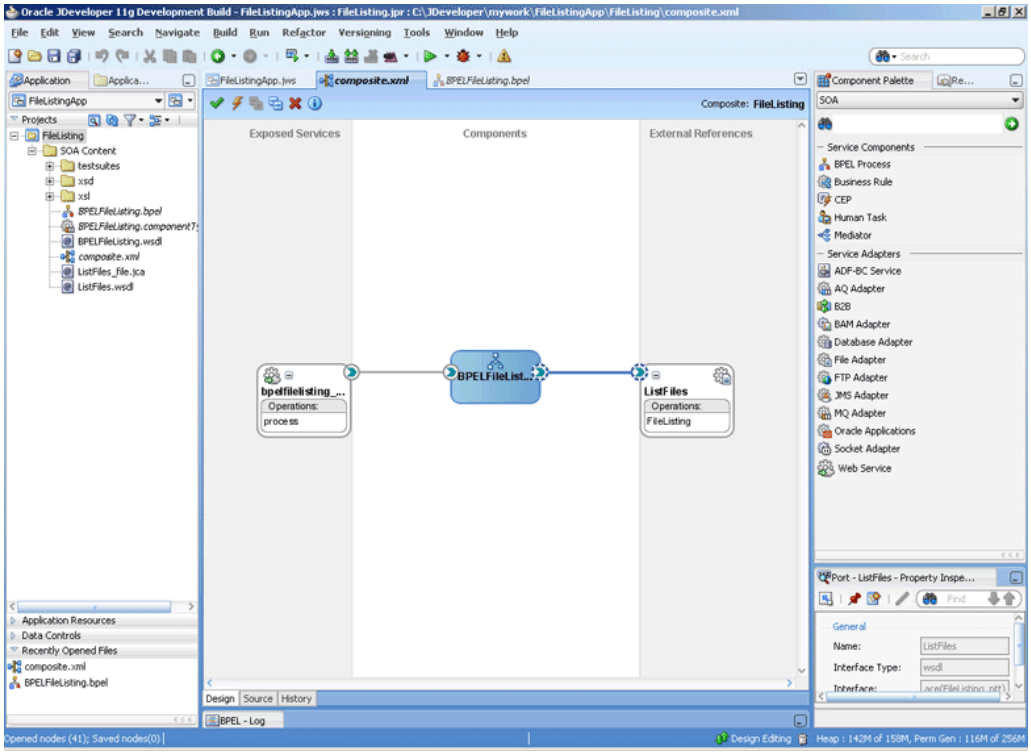
Wiring Services and Activities

You have to assemble or wire the two components that you have created: BPEL process, and the Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the BPEL process in the **Components** area to the drop zone that appears as a green triangle in ListFiles in the External References area.

   The JDeveloper `composite.xml` appears, as shown in Figure 4-145.

Figure 4-145 The JDeveloper - Composite.xml

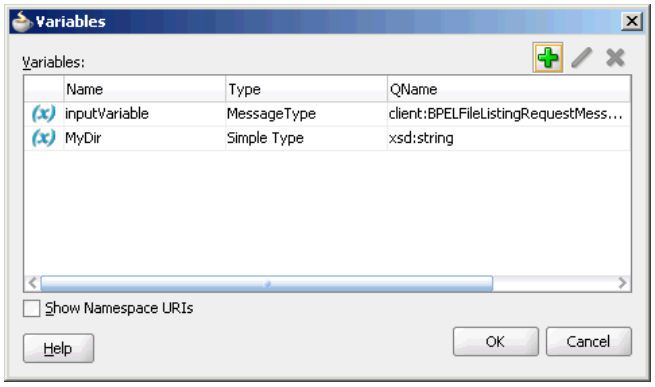


2. Click **File**, **Save All**.

Create a String Variable

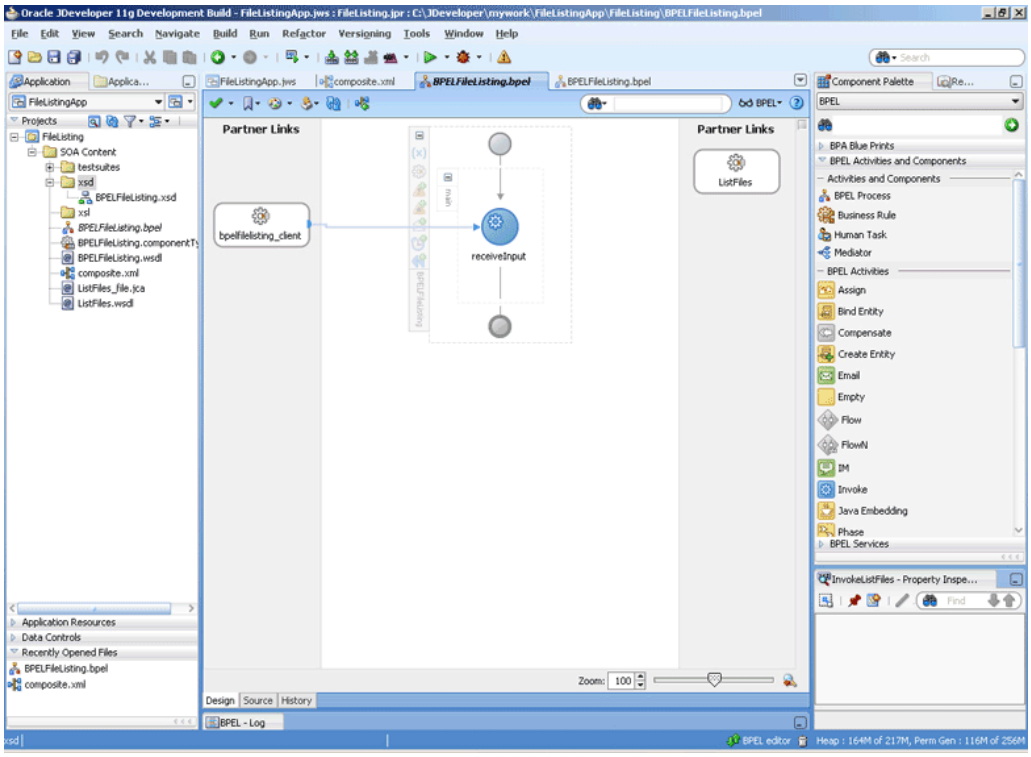
1. Double-click **BPELFileListing**. The BPELFileListing.bpel page is displayed.
2. Click the **Variables...** icon represented by (x). The **Variables** dialog is displayed.
3. Click the **Create...** icon. The **Create Variable** dialog is displayed.
4. Create a variable, `MyDir` of type `xsd:string`, as shown in Figure 4-146, for later use.

Figure 4-146 The Variables Dialog



5. Click **OK**. The JDeveloper BPELFileListing.bpel page appears, as shown in Figure 4-147

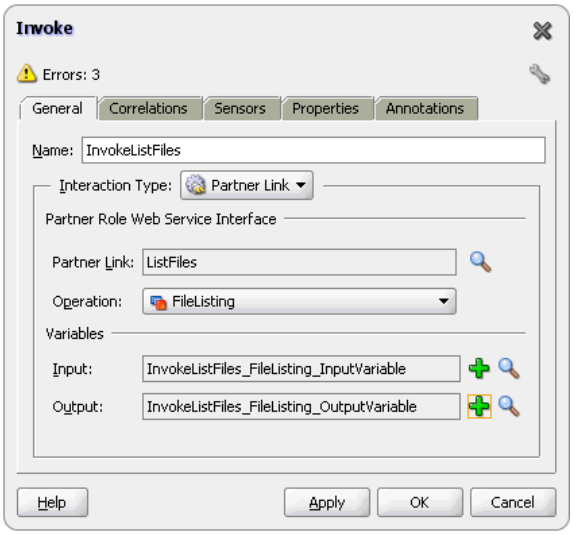
Figure 4-147 The JDeveloper - BPELFileListing.bpel



Add an Invoke Activity

1. Drag and drop an **Invoke** activity below the receive Activity from the **Components** window to the design area.
2. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.
3. Enter `InvokeListFiles` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **ListFiles**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The **Create Variable** dialog is displayed.
7. Select the default variable name and select **OK**. The Variable field is populated with the default variable name. The **Invoke** dialog is displayed with input variable populated.
8. Repeat the same to select the output variable. The **Invoke** dialog is displayed, as shown in Figure 4-148.

Figure 4-148 The Invoke Dialog



9. Click the **Properties** tab. The properties and the corresponding value column is displayed.

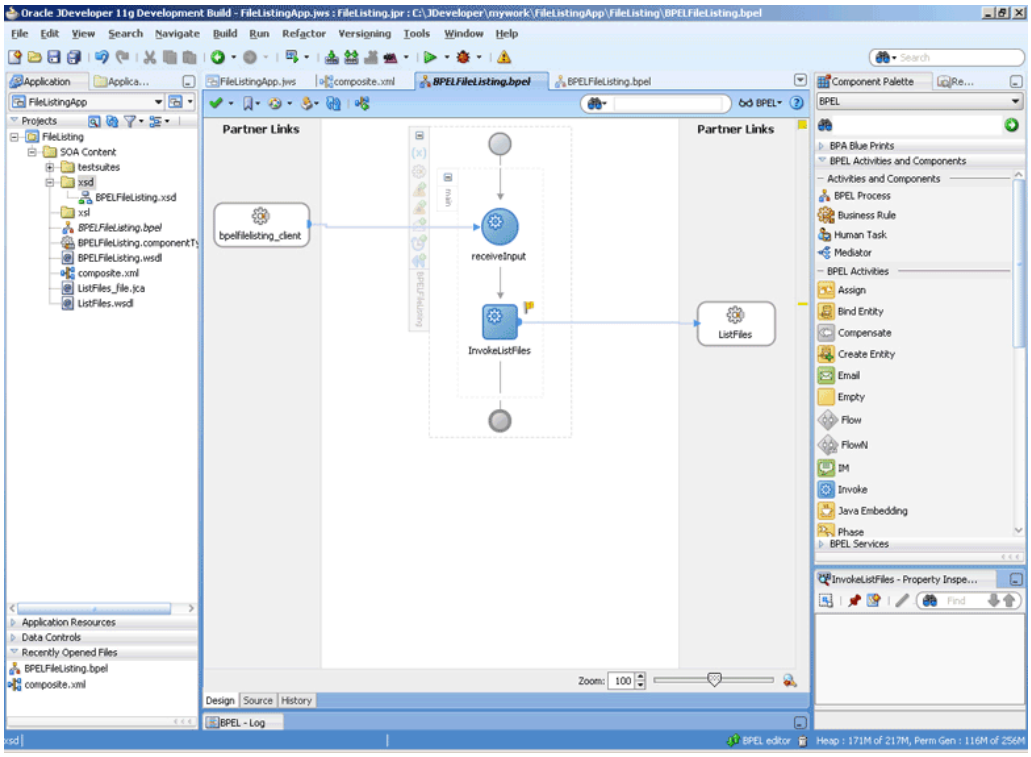
10. Select jca.file.Directory property. Double-click in the corresponding value column. The **Adapter Property Value** dialog is displayed.

11. Click the **Browse Variables** icon. The **Variable XPath Builder** dialog is displayed.

12. Expand **Variables**, select **MyDir**, and then click **OK**. The value of the `jca.file.Directory` is set to `Mydir`.

13. Click **OK**. The JDeveloper BPELFileListing.bpel page appears, as shown in Figure 4-149.

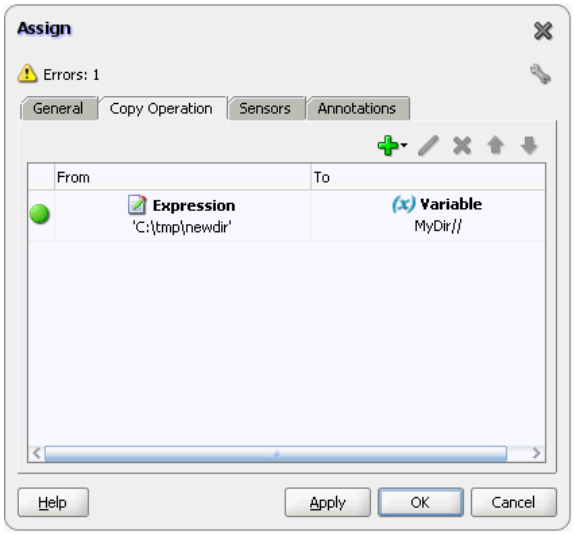
Figure 4-149 The JDeveloper - BPELFileListing.bpel



Add an Assign Activity

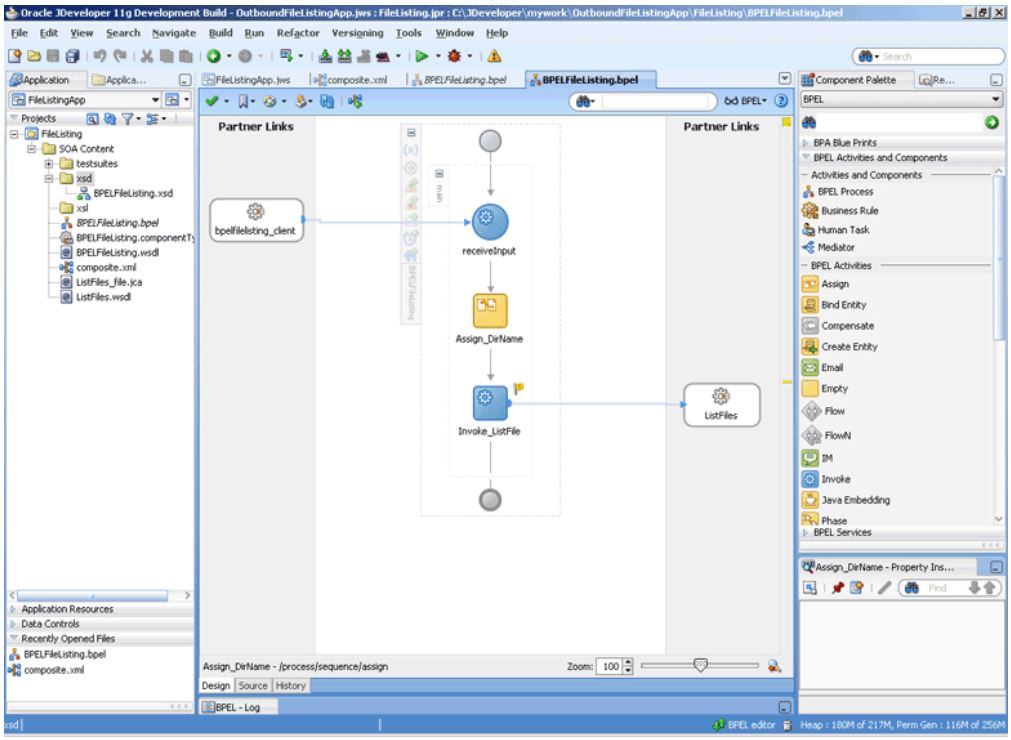
1. Drag and drop an **Assign** activity from the **Components** window in between the **Receive** activities and the **Invoke** activity in the design area.
2. Double-click the **Assign** activity. The **Assign** dialog is displayed.
3. Enter `AssignDirName` in the **Name** field.
4. Click the **Copy Operation** tab. The **Assign** dialog is displayed.
5. Select **Copy Operation**. The **Create Copy Operation** dialog is displayed.
6. Set the values for the headers, as shown in Figure 4-150.

Figure 4-150 The Assign Dialog



7. Click **OK**, the JDeveloper BPELFileListing.bpel page is displayed, as shown in Figure 4-151.

Figure 4-151 The JDeveloper - BPELFileListing.bpel



8. Click **File**, **Save All**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from .

Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the `*.txt` files to the input directory (see Prerequisites for details) and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The **Dashboard** is displayed.

   Note your Instance ID in the **Recent Instances** area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The **Flow Trace** page is displayed.
6. Click your BPEL process instance. The **Audit Trail** of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow. Additionally, click an activity to view the details of an activity.

Oracle File Adapter Complex Structure

This use case demonstrates the ability of the Oracle File Adapter to process native data defined in a custom format. In this sample, the custom format represents an invoice defined in `invoice-nxsd.xsd`. The Oracle File Adapter processes the `invoice.txt` file and publishes this to the ComplexStructure BPEL process. This is then transformed to a PurchaseOrder and written out as an xml file.

This use case includes the following sections:

- Prerequisites
- Designing the SOA Composite
- Creating the Inbound Service
- Creating the Outbound Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using Fusion Middleware Control Console

Prerequisites

To perform the complex structure business process, you require the following files from the `artifacts.zip` file contained in the `Adapters-104FileAdapterComplexStructure` sample:

- `artifacts/schemas/invoice-nxsd.xsd`
- `artifacts/schemas/po.xsd`
- `artifacts/xsl/InvToPo.xsl`
- `artifacts/input/invoice.txt`

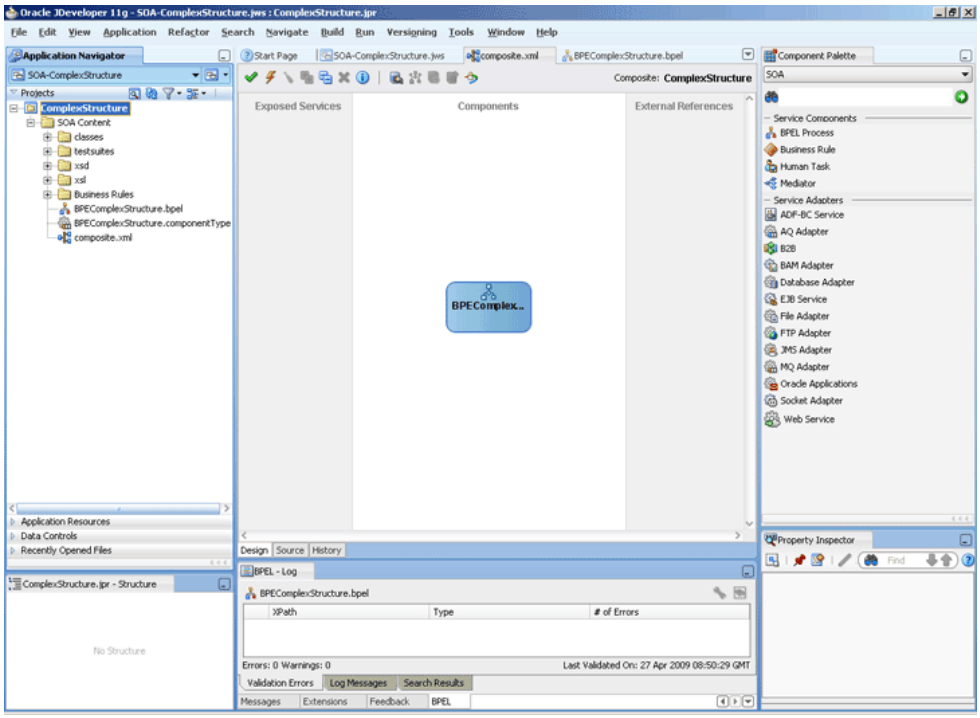
You can obtain the `Adapters-104FileAdapterComplexStructure` sample by accessing the Oracle SOA Sample Code site.

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `SOA-ComplexStructure` in the **Application Name** field, and click **Next**. The Create Generic Application - Name your project page is displayed.
3. Enter `ComplexStructure` in the **Project Name** field.
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process - BPEL Process page is displayed.
7. Enter `BPEComplexStructure` in the **Name** field, select **Define Service Later** from the Template box.
8. Click **OK**. The SOA-ComplexStructure application and the ComplexStructure project appear in the design area, as shown in Figure 4-152.

Figure 4-152 The JDeveloper - Composite.xml



9. Copy the **invoice-nxsd.xsd** and **po.xsd** files to the schema directory in your project (see Prerequisites for the location of these files).

10. Copy **InvToPo.xsl** to the xsl directory of your project (see Prerequisites for the location of this file).

Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory:

1. Drag and drop File Adapter from the **Components** window to the Exposed Services swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.

2. Click **Next**. The **Service Name** page is displayed.

3. Enter `Complex Structure In` in the **Service Name** field.

4. Click **Next**. The Adapter Interface page is displayed.

5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.

6. Select **Read File**, and click **Next**. The File Directories page is displayed.

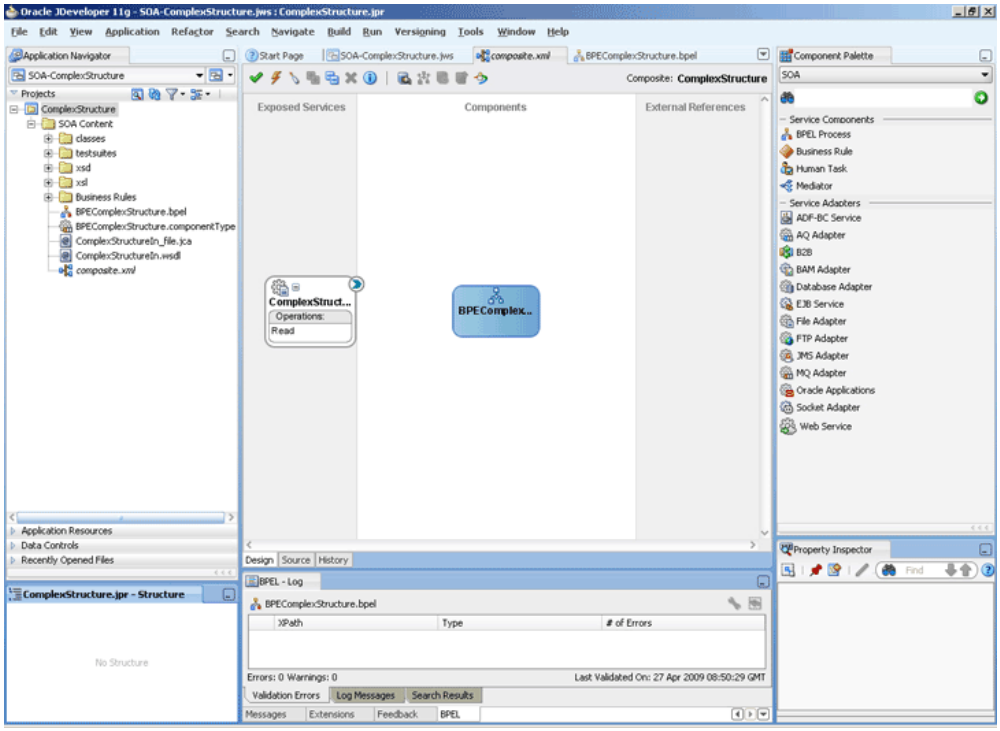
7. Enter the physical path for the input directory and click **Next**. The **File Filtering** page is displayed.

8. Enter `*.txt` in the **Include Files With Name Pattern** field, click **Next**. The File Polling page is displayed.

9. Click **Next**. The Messages page is displayed.

10. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
11. Click **Project Schema Files**, **invoice-nxsd.xsd**, and **invoice**.
12. Click **OK**. The URL field in the Messages page is populated with the `invoice-nxsd.xsd` file.
13. Click **Next**. The **Finish** page is displayed.
14. Click **Finish**. The inbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-153.

Figure 4-153 The JDeveloper - Composite.xml



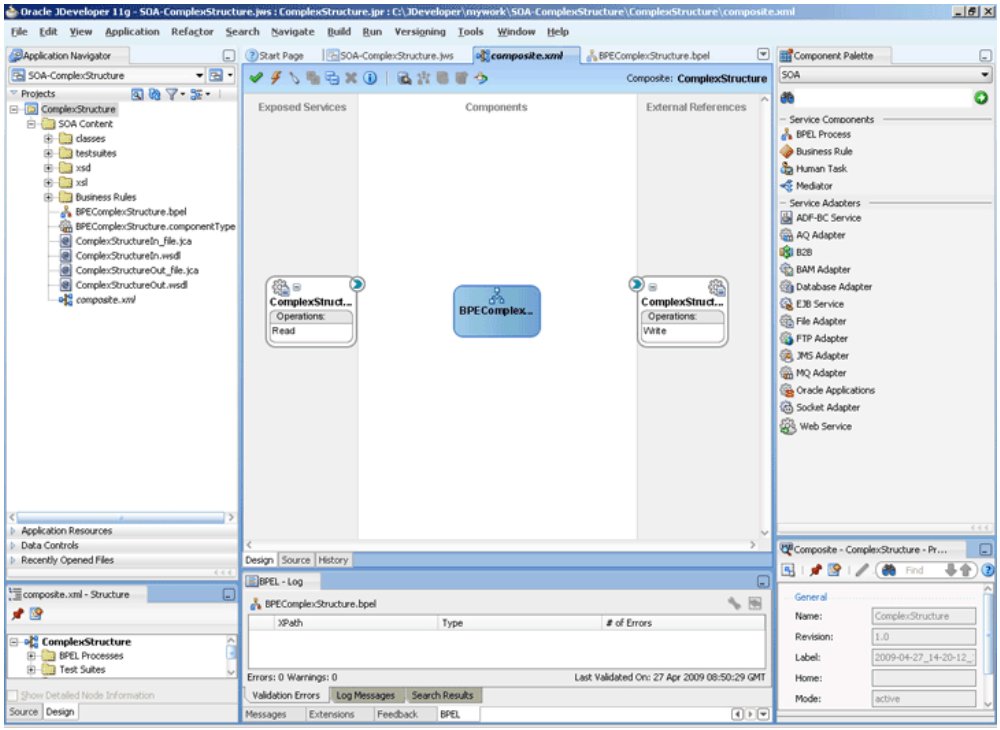
Creating the Outbound Oracle File Adapter Service

Perform the following steps to create an outbound Oracle File Adapter service to write the file from a local directory to the FTP server:

1. Drag and drop the Oracle File Adapter from the **Components** window to the **External References** swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `ComplexStructureOut` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, and click **Next**. The **File Configuration** page is displayed.

7. Enter the physical path for the output directory and enter `invoice_%SEQ%.txt` in the **File Naming Convention(po_%SEQ%.txt)** field.
8. Click **Next**. The **Messages** page is displayed.
9. Click **Browse For Schema File** that appears at the end of the URL field. The **Type Chooser** dialog is displayed.
10. Click **Project Schema Files**, **po.xsd**, and **po**.
11. Click **OK**. The URL field in the Messages page is populated with the po.xsd file.
12. Click **Next**. The **Finish** page is displayed.
13. Click **Finish**. The outbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-154.

Figure 4-154 The JDeveloper - Composite.xml



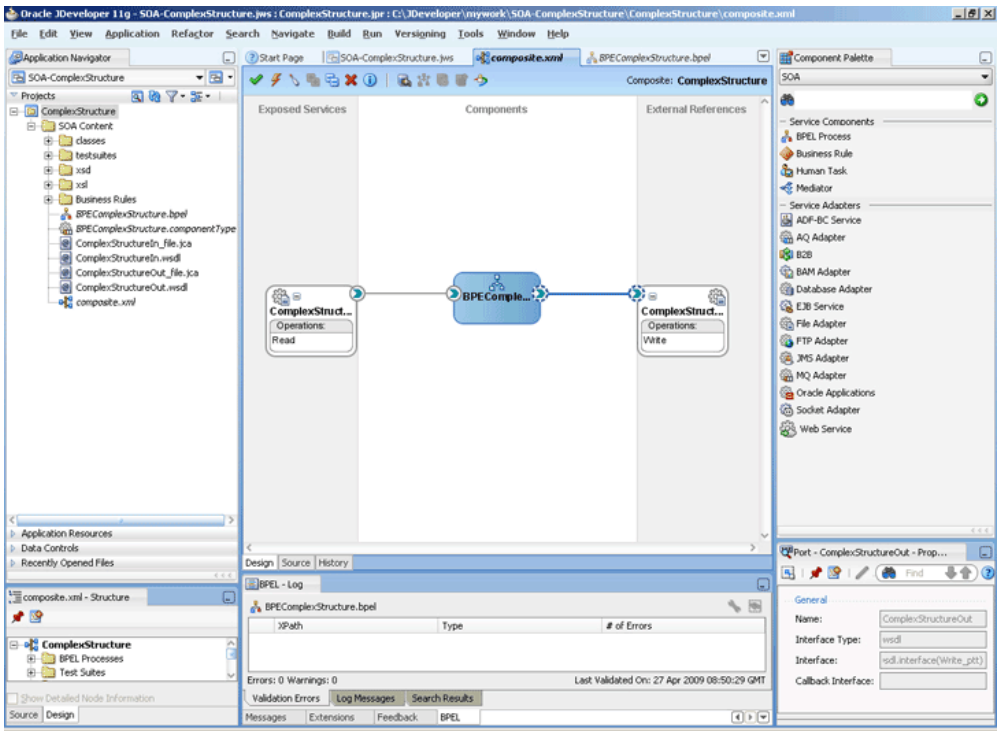
Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the ComplexStructureIn service in the **Exposed Services** area to the drop zone that appears as a green triangle in the BPEL process in the **Components** area.
2. Drag the small triangle in the BPEL process in the **Components** area to the drop zone that appears as a green triangle in the ComplexStructureOut reference in the **External References** area.

   The JDeveloper `composite.xml` appears, as shown in Figure 4-155.

Figure 4-155 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.

Add a Receive Activity

1. Double-click **BPELComplexStructure**. The BPELComplexStructure.bpel page is displayed.
2. Drag and drop a **Receive** activity from the **Components** window to the design area.
3. Double-click the **Receive** activity. The **Receive** dialog is displayed.
4. Enter `ReceiveInvoice` in the **Name** field.
5. Click **Browse Partner Links** at the end of the **Partner Link** field. The **Partner Link Chooser** dialog is displayed.
6. Select **ComplexStructureIn**, and click **OK**.
7. Click the **Auto-Create Variable** icon to the right of the Variable field in the **Receive** dialog. The **Create Variable** dialog is displayed.
8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper BPELComplexStructure.bpel page appears.

Add an Invoke Activity

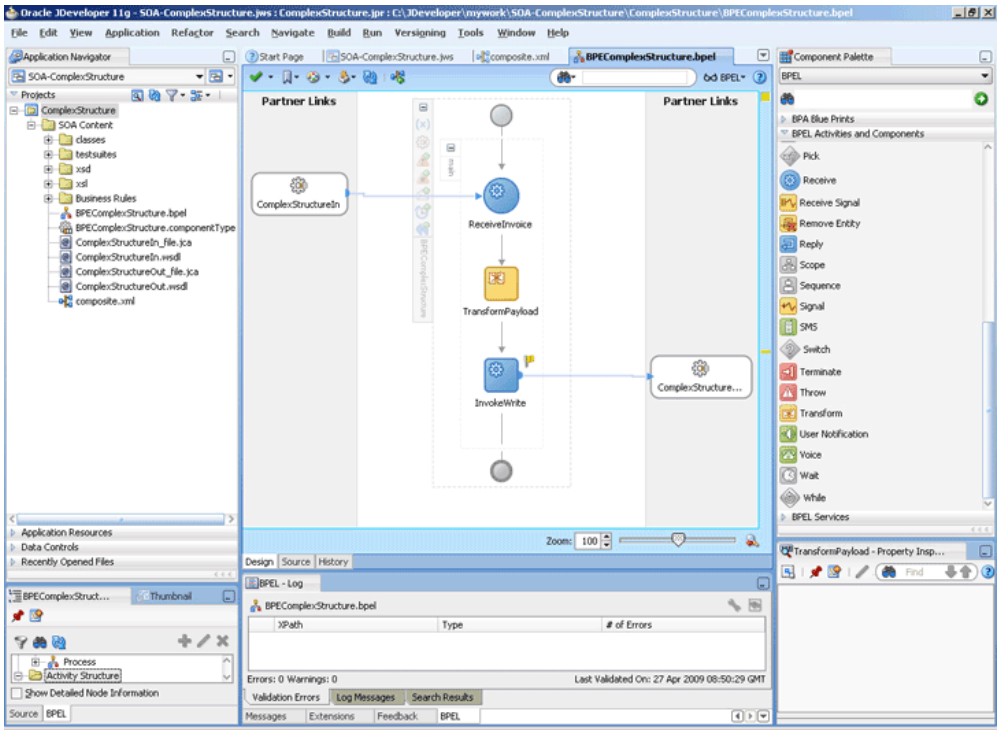
1. Drag and drop an **Invoke** activity from the **Components** window to the design area.

2. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.
3. Enter `InvokeWrite` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.
5. Select **ComplexStructureOut**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The **Create Variable** dialog is displayed.
7. Enter `InvokeWrite_Write_OutputVariable` in the variable name field and click **OK**. The Invoke dialog is displayed.
8. Click **OK**. The JDeveloper **BPELComplexStructure.bpel** page appears.

Add a Transform Activity

1. Drag and drop a **Transform** activity from the **Components** window in between the Receive and Invoke activities in the design area.
2. Double-click the **Transform** activity. The **Transform** dialog is displayed.
3. Enter `TransformPayload` in the **Name** field.
4. Click the **Transformation** tab. The **Transform** dialog is displayed.
5. Select the **Create...** icon. The **Source Variable** dialog is displayed.
6. Select **ReceiveInvoice_Read_InputVariable** in the Source Variable box, and select **body** in the Source Part box, and then click **OK**. The **Transform** dialog is displayed with the Source and Part selected.
7. Select **InvokeWrite_Write_OutputVariable** in the Target Variable list, select **body** in the Target Part.
8. Click the **Browse Mapping** icon at the end of the Mapper File field and select **InvToPo.xsl** file from the xsl directory in your project.
9. Click **OK**.
10. Click **File**, **Save All**. The BPELComplexStructure.bpel page is displayed, as shown in Figure 4-156.

Figure 4-156 The JDeveloper - BPELComplexStructure.bpel



Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from .

Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the `invoice.txt` file to the input directory (see Prerequisites for the location of this file) and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the **Recent Instances** area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The **Flow Trace** page is displayed.

6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow.
9. Click **ReceiveInvoice** to display the activity details

Oracle FTP Adapter Debatching

This is an Oracle FTP Adapter feature that debatches a large XML document into smaller individual XML fragments. This use case demonstrates how the debatching business process sample uses the Oracle FTP Adapter to process a file containing a batch of business records such as one or more invoice and purchase orders. The PurchaseOrders (POs) are then debatched and written to separate output files.

This use case includes the following sections:

- Prerequisites
- Designing the SOA Composite
- Creating the Inbound Service
- Creating the Outbound Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using Fusion Middleware Control Console

Prerequisites

To perform the complex structure business process, you require the following files from the `artifacts.zip` file contained in the `Adapters-101FTPAdapterDebatching` sample:

- `artifacts/schemas/container.xsd`
- `artifacts/schemas/po.xsd`
- `artifacts/xsl/InvToPo.xsl`
- `artifacts/xsl/PoToPo.xsl`
- `artifacts/input/container.txt`

You can obtain the `Adapters-101FTPAdapterDebatching` sample by accessing the Oracle SOA Sample Code site.

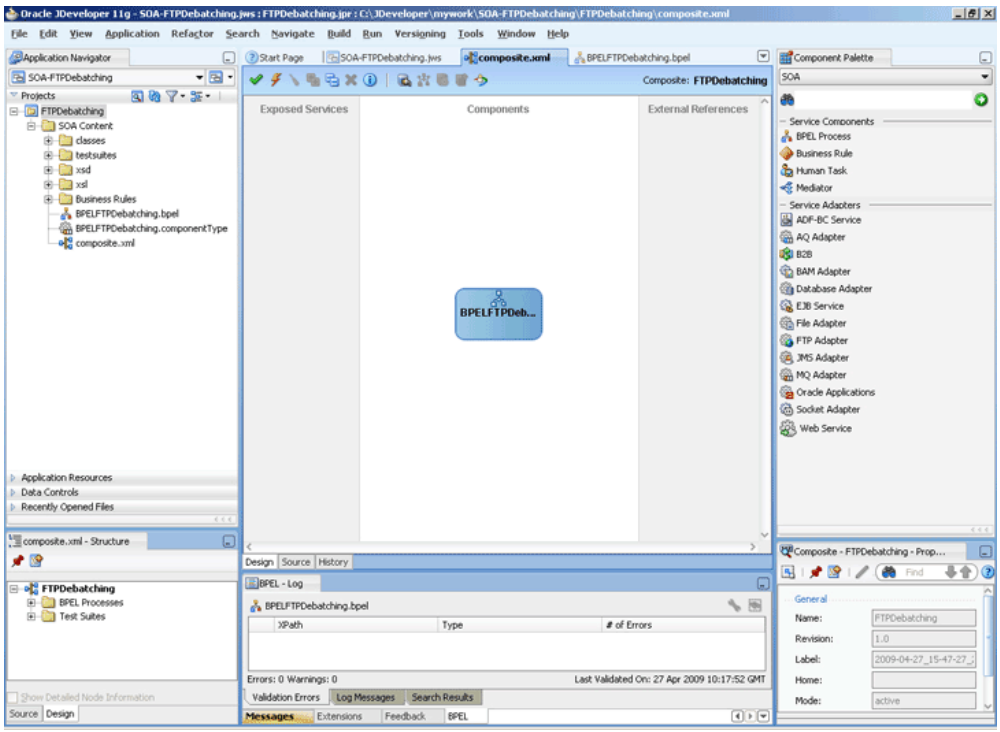
Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.

2. Enter `SOA-FTPDebatching` in the **Application Name** field, and click **OK**. The **Create Generic Application - Name** your project page is displayed.
3. Enter `FTPDebatching` in the **Project Name**.
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The **Configure SOA settings** dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The **Create BPEL Process - BPEL Process** page is displayed.
7. Enter `BPELFTPDebatching` in the **Name** field, select **Define Service Later** from the Template box.
8. Click **OK**. The SOA-FTPDebatching application and the FTPDebatching project appears in the design area, as shown in Figure 4-157.

Figure 4-157 The JDeveloper - Composite.xml



9. Copy the `container.xsd` and `po.xsd` files to the `xsd` directory of your project (see Prerequisites for the location of these files).
10. Copy the `InvToPo.xsl` and `PoToPo.xsl` files to the xsl directory of your project (see Prerequisites for the location of these files).

Creating the Inbound Oracle FTP Adapter Service

Perform the following steps to create an inbound Oracle FTP Adapter service to read the file from a local directory:

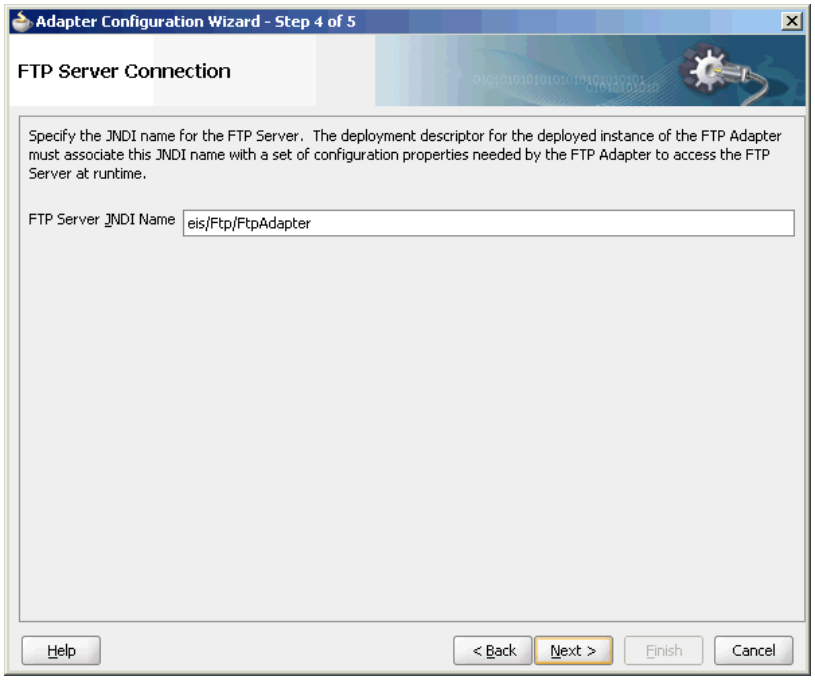
1. Drag and drop the Oracle FTP Adapter from the **Components** window to the Exposed Services swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.

2. Click **Next**. The **Service Name** page is displayed.
3. Enter `FTPDebatchingIn` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The FTP Server Connection page is displayed, as shown in Figure 4-158.

Note:

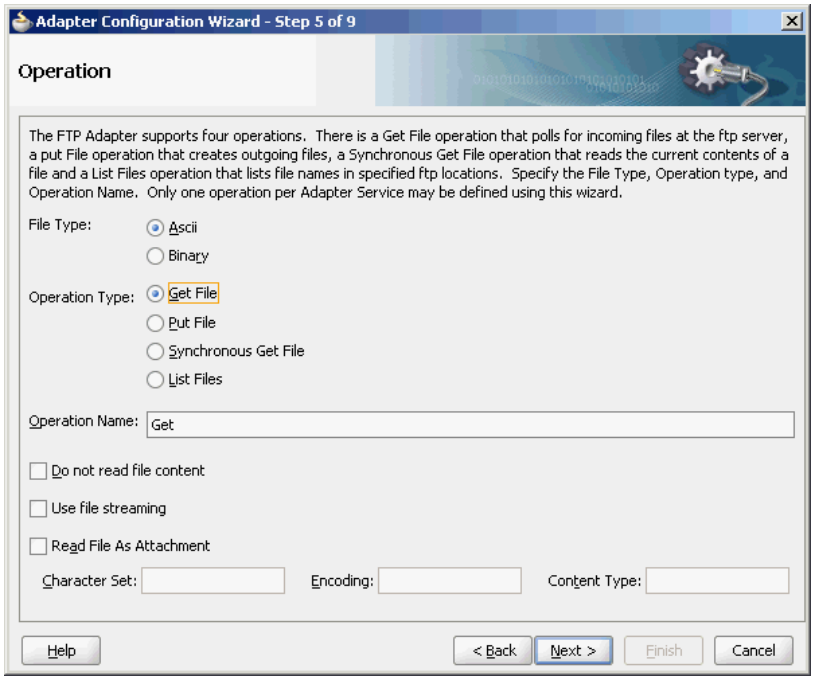
Ensure that you have configured the jndi-name in the deployment descriptor for Oracle FTP Adapter before deploying this application.

Figure 4-158 The Adapter Configuration Wizard FTP Server Connection Page



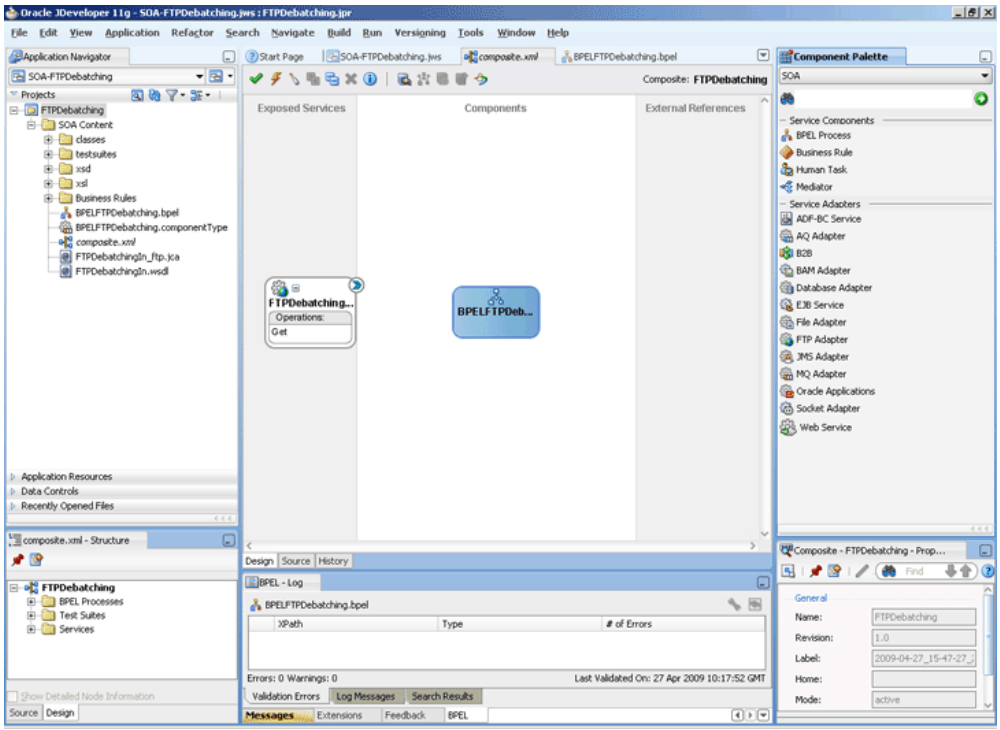
6. Click **Next**. The Operation page is displayed.
7. Select **Get File**, as shown in Figure 4-159, and click **Next**. The **File Directories** page is displayed.

Figure 4-159 The Adapter Configuration Wizard Operation Page



8. Enter the physical path for the input directory, and click **Next**. The File Filtering page is displayed.
9. Enter `*.txt` in the **Include Files With Name Pattern** field, select **Files Contain Multiple Messages** check box, specify `1` as the value for Publish Messages in Batches Of box.
10. Click **Next**. The **File Polling** page is displayed.
11. Click **Next**. The **Messages** page is displayed.
12. Click **Browse For Schema File** that appears at the end of the URL field. The **Type Chooser** dialog is displayed.
13. Click **Project Schema Files**, **container.xsd**, and **container**.
14. Click **OK**. The URL field in the **Messages** page is populated with the **container.xsd** file.
15. Click **Next**. The **Finish** page is displayed.
16. Click **Finish**. The inbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-160.

Figure 4-160 The JDeveloper - Composite.xml



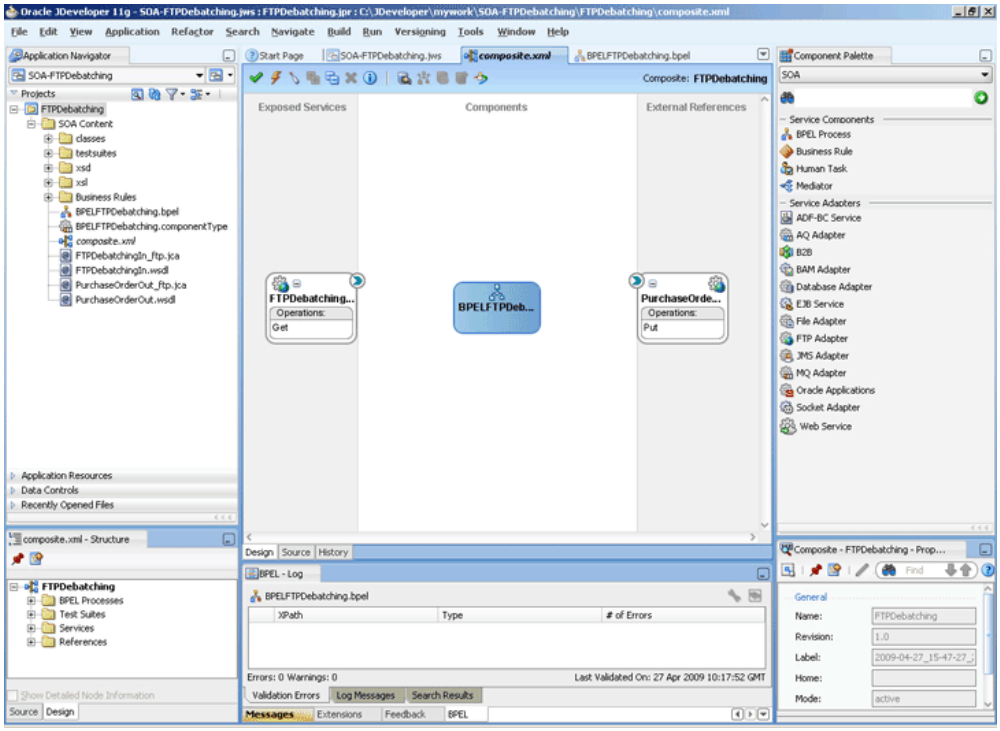
Creating the Outbound Oracle FTP Adapter Service

Perform the following steps to create an outbound Oracle FTP Adapter service to write the file from a local directory to the FTP server:

1. Drag and drop the FTP Adapter from the Components window to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `PurchaseOrderOut` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The **FTP Server Connection** page is displayed.
6. Click **Next**. The **Operation** page is displayed.
7. Select **Put File**, and click **Next**. The **File Configuration** page is displayed.
8. Enter the physical path for the output directory and enter `po_%SEQ%.txt` in the **File Naming Convention(po_%SEQ%.txt)** field.
9. Select **Number of Messages Equals** option, if not selected. The default value is `1`.
10. Click **Next**. The **Messages** page is displayed.
11. Click **Browse For Schema File** that appears at the end of the URL field. The **Type Chooser** dialog is displayed.
12. Click **Project Schema Files**, **po.xsd**, and **po**.

13. Click **OK**. The URL field in the **Messages** page is populated with the `po.xsd` file.
14. Click **Next**. The Finish page is displayed.
15. Click **Finish**. The outbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-161.

Figure 4-161 The JDeveloper - Composite.xml



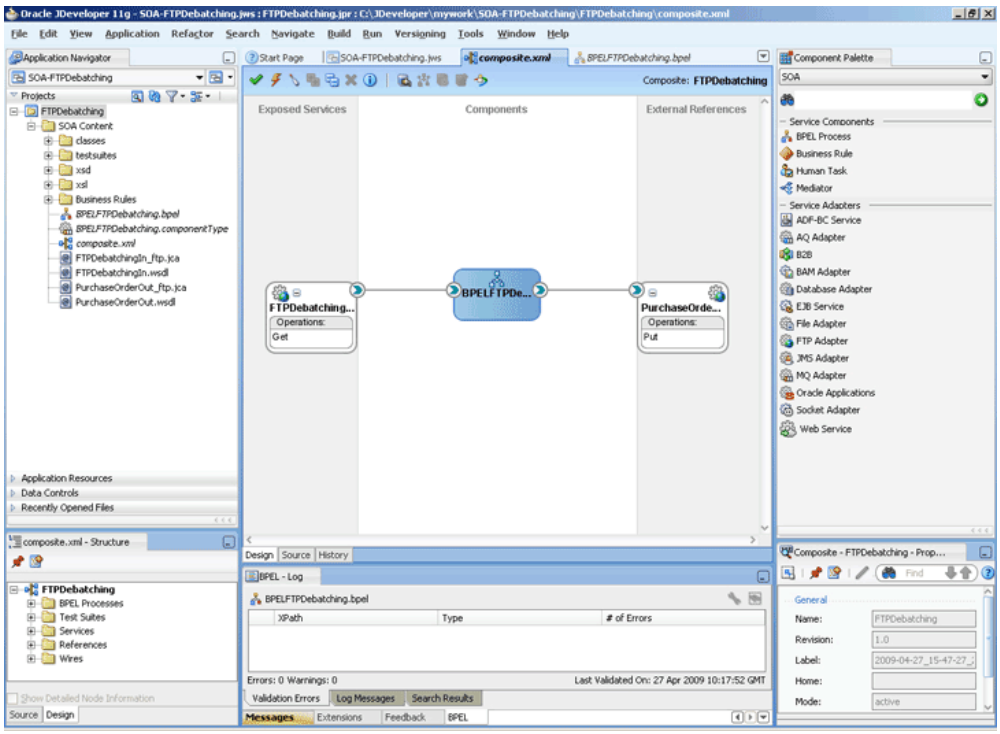
Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the FTPDebatchingIn service in the **Exposed Services** area to the drop zone that appears as a green triangle in the BPEL process in the **Components** area.
2. Drag the small triangle in the BPEL process in the **Components** area to the drop zone that appears as a green triangle in the PurchaseOrderOut reference in the **External References** area.

   The JDeveloper `composite.xml` appears, as shown in Figure 4-162.

Figure 4-162 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.

Add a Receive Activity

1. Double-click **BPELFTPDebatching**. The `BPELFTPDebatching.bpel` page is displayed.
2. Drag and drop a **Receive** activity from the **Components** window to the design area.
3. Double-click the **Receive** activity. The **Receive** dialog is displayed.
4. Enter `Receive` in the **Name** field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.
6. Select **FTPDebatchingIn**, and click **OK**.
7. Click the **Auto-Create Variable** icon to the right of the Variable field in the **Receive** dialog. The **Create Variable** dialog is displayed.
8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper `BPELFTPDebatching.bpel` page appears with the Receive activity added.

Add an Invoke Activity

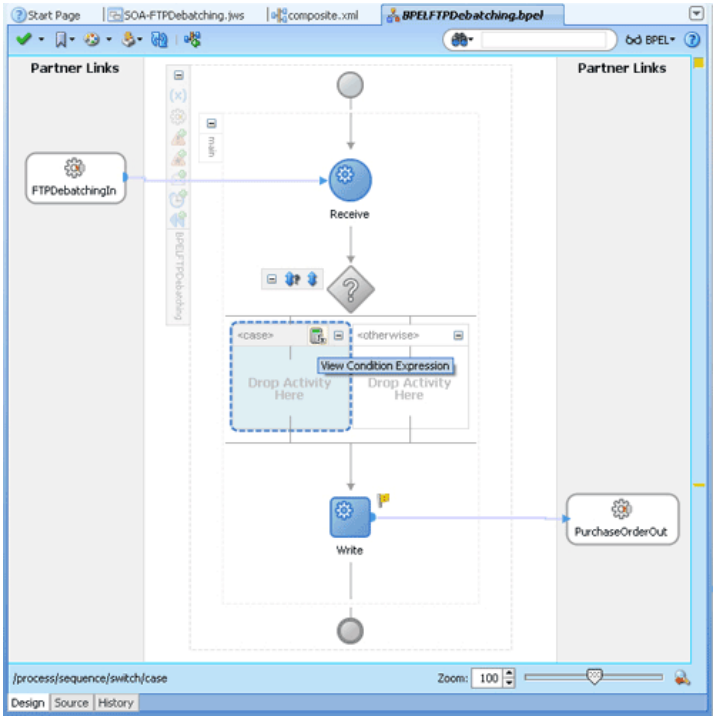
1. Drag and drop an **Invoke** activity from the **Components** window to the design area.

2. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.
3. Enter `Write` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.
5. Select **PurchaseOrderOut**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Receive dialog. The **Create Variable** dialog is displayed.
7. Enter `Write_Put_OutputVariable` in the Variable field and click **OK**. The **Invoke** dialog is displayed.
8. Click **OK**. The JDeveloper `BPELFTPDebatching.bpel` page appears with the invoke activity added.

Add a Switch Activity

1. Drag and drop a **Switch** activity from the **Components** window in between the Receive and Invoke activities in the design area.
2. Expand the **Switch** activity. This displays a screen to enter the values for <case> and <otherwise>.
3. In the `<case>` section, click the **View Condition Expression** icon, as shown in Figure 4-163. The **Condition Expression** pop-up window is displayed.

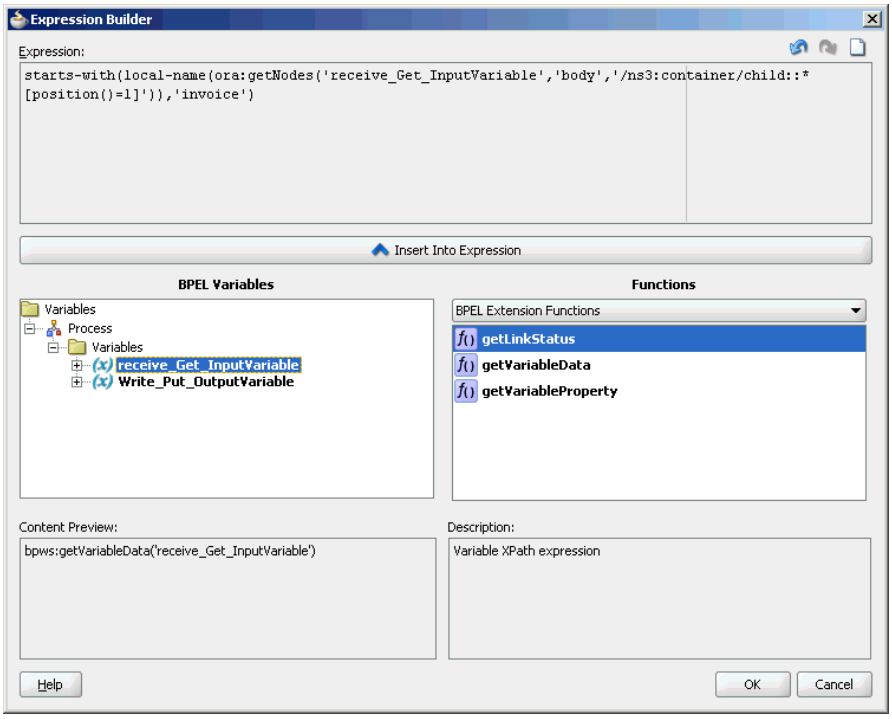
Figure 4-163 BPELFTPDebatching.bpel Page



4. Click the **Xpath Expression Builder** icon in the pop-up window. The Expression Builder dialog is displayed.

5. Enter `starts-with(local-name(ora:getNodes('receive_Get_InputVariable','body','/ns3:container/child::*[position()=1])),'invoice')` as the expression, as shown in Figure 4-164, and click **OK**. The screen returns to the **Condition Expression** pop-up window.

Figure 4-164 The Expression Builder Dialog



6. Add two transformation activities, one each for <case> and <otherwise> sections.

   a. Drag and drop a **Transform** activity in the <case> section.

   b. Double-click the **Transform** activity.

   c. Enter `InvToPo` in the **Name** field.

   d. Click the **Transformation** tab.

   e. Click the **Create...** icon. The **Source Variable** dialog is displayed.

   f. Accept the defaults and click **OK**.

   g. Select **Write_Put_OutputVariable** in the Target Variable list.

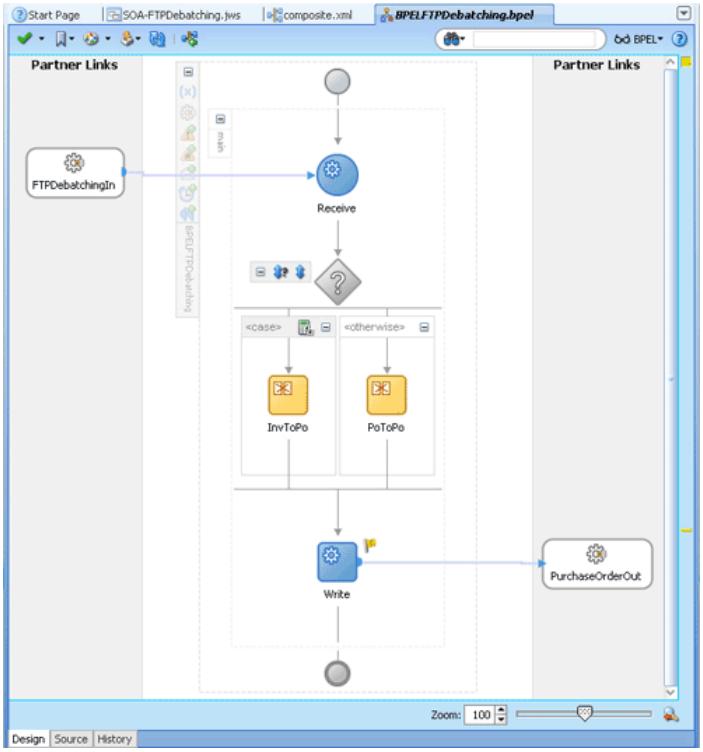
   h. Click the **Browse Mappings** icon at the end of the **Mapper File** field, and select the **InvToPo.xsl** file.

   i. Click **OK**.

   j. Repeat the same process for the second transformation. Select `PoToPo.xsl` as the Mapper File for this transform activity.

   The BPELFTPDebatching.bpel page is displayed, as shown in Figure 4-165.

Figure 4-165 The BPELFTPDebatching.bpel Page



7. Click **File**, **Save All**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from .

Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. To do so, perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the `container.txt` file to the input directory (see Prerequisites for the location of this file) and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The **Dashboard** is displayed.

   Note your Instance ID in the **Recent Instances** area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.

5. Click the Instance ID that you noted in Step 3. The **Flow Trace** page is displayed.
6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow. Additionally, click an activity (such as invoke, receive) to view the details of an activity.

Oracle FTP Adapter Dynamic Synchronous Read

This use case demonstrates the ability of the Oracle FTP Adapter to perform a mid-process synchronous read operation using an Invoke activity. This use case illustrates the following adapter functionality:

- Oracle File Adapter (Read Operation)
- Oracle FTP Adapter (Synchronous Read operation)

  Ability to specify the file name to be read during runtime
- Oracle File Adapter (Write Operation)

The process is initiated by the presence of a file appearing in a local directory monitored by the inbound Oracle File Adapter. The trigger file contains the name of the file to be read by the synchronous read operation. This file name is passed through headers to the adapter. This can be done using the **Properties** tab for the Invoke activity. This synchronous read file operation is performed against a remote directory on a FTP server. The result of the read is then transformed and written out to a local directory through the outbound Oracle File Adapter. This section includes the following topics:

Prerequisites

To perform FTP Dynamic Synchronous Read, you require the following files from the `artifacts.zip` file contained in the `Adapters-102FTPAdapterDynamicSynchronousRead` sample:

- `artifacts/schemas/address-csv.xsd`
- `artifacts/schemas/address-fixedLength.xsd`
- `artifacts/schemas/trigger.xsd`
- `artifacts/xsl/addr1Toaddr2.xsl`

- `artifacts/input/address_csv.txt`
- `artifacts/input/trigger.trg`

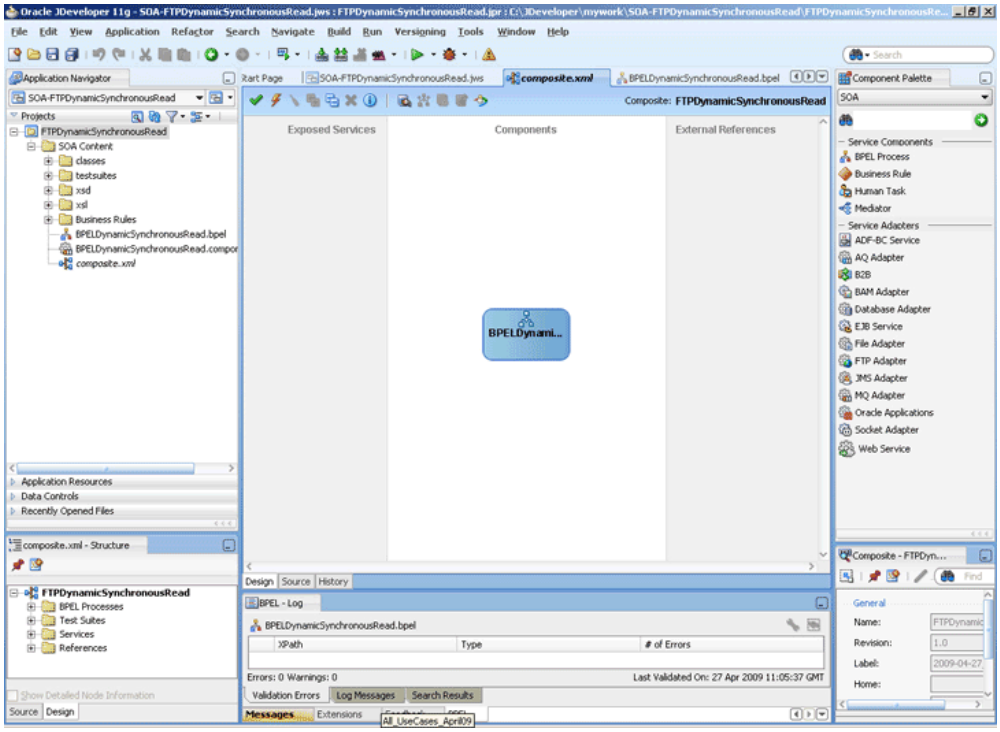
You can obtain the `Adapters-102FTPAdapterDynamicSynchronousRead` sample by accessing the Oracle SOA Sample Code site, and selecting the Adapters tab.

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `SOA-FTPDynamicSynchronousRead` in the **Application Name** field, and click **OK**. The Create Generic Application - Name your project page is displayed.
3. Enter `FTPDynamicSynchronousRead` in the **Project Name**.
4. In the **Available list** under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The **Configure SOA settings** dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The **Create BPEL Process - BPEL Process** page is displayed.
7. Enter `BPELDynamicSynchronousRead` in the **Name** field, select **Define Service Later** from the Template box.
8. Click **OK**. The SOA-FTPDynamicSynchronousRead application and FTPDynamicSynchronousRead project appears in the design area, as shown in Figure 4-166.

Figure 4-166 The JDeveloper - Composite.xml



9. Copy the `address-csv.xsd`, `address-fixedLength.xsd`, and `trigger.xsd` files to the xsd directory of your project (see Prerequisites for the location of these files).

10. Copy the `addr1Toaddr2.xsl` file to the xsl directory of your project (see Prerequisites for the location of this file).

Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory:

1. Drag and drop File Adapter from the **Components** window to the **Exposed Services** swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.

2. Click **Next**. The **Service Name** page is displayed.

3. Enter `ReadTrigger` in the **Service Name** field.

4. Click **Next**. The **Adapter Interface** page is displayed.

5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.

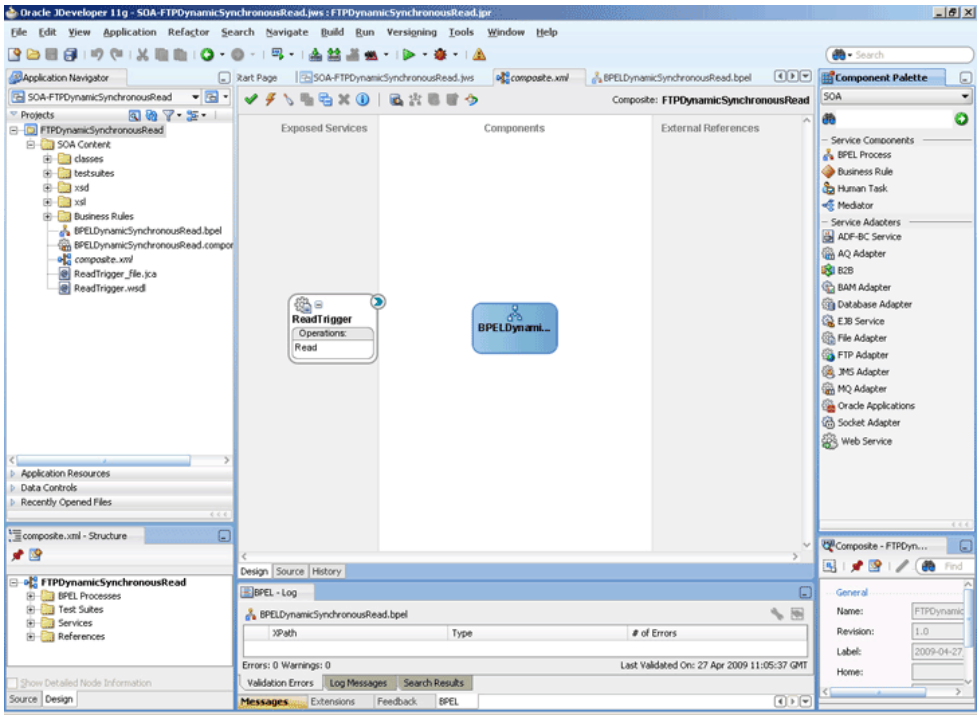
6. Select **Read File**, and click **Next**. The **File Directories** page is displayed.

7. Enter the physical path for the input directory and click **Next**. The **File Filtering** page is displayed.

8. Enter `*.trg` in the **Include Files With Name Pattern** field, click **Next**. The File Polling page is displayed.

9. Click **Next**. The Messages page is displayed.
10. Click **Browse For Schema File** that appears at the end of the URL field. The **Type Chooser** dialog is displayed.
11. Click **Project Schema Files**, **trigger.xsd**, and **trigger**.
12. Click **OK**. The URL field in the **Messages** page is populated with the `trigger.xsd` file.
13. Click **Next**. The **Finish** page is displayed.
14. Click **Finish**. The inbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-167.

Figure 4-167 The JDeveloper - Composite.xml



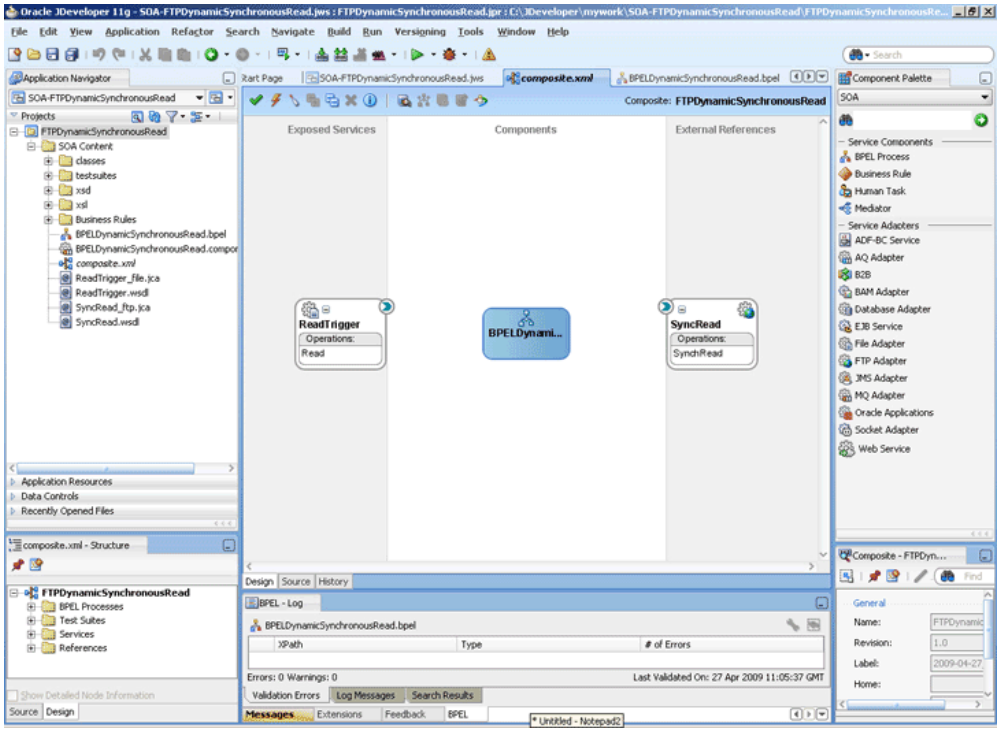
Creating the Outbound Oracle FTP Adapter Service

Perform the following steps to create an outbound Oracle FTP Adapter service to write the file from a local directory to the FTP server:

1. Drag and drop FTP Adapter from the **Components** window to the External References swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `SyncRead` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The FTP Server Connection page is displayed.

6. Click **Next**. The **Operation** page is displayed.
7. Select **Synchronous Get File**, and click **Next**. The **File Directories** page is displayed.
8. Enter the physical path for the output directory.
9. Click **Next**. The File Name page is displayed.
10. Enter `dummy.txt` in the **File Name** field and click **Next**. The **Messages** page is displayed.
11. Click **Browse For Schema File** that appears at the end of the URL field. The **Type Chooser** dialog is displayed.
12. Click **Project Schema Files**, **address-csv.xsd**, and **Root-Element**.
13. Click **OK**. The URL field in the **Messages** page is populated with the `address-csv.xsd` file.
14. Click **Next**. The **Finish** page is displayed.
15. Click **Finish**. The outbound Oracle FTP Adapter is now configured and `composite.xml` appears, as shown in Figure 4-168.

Figure 4-168 The JDeveloper - Composite.xml

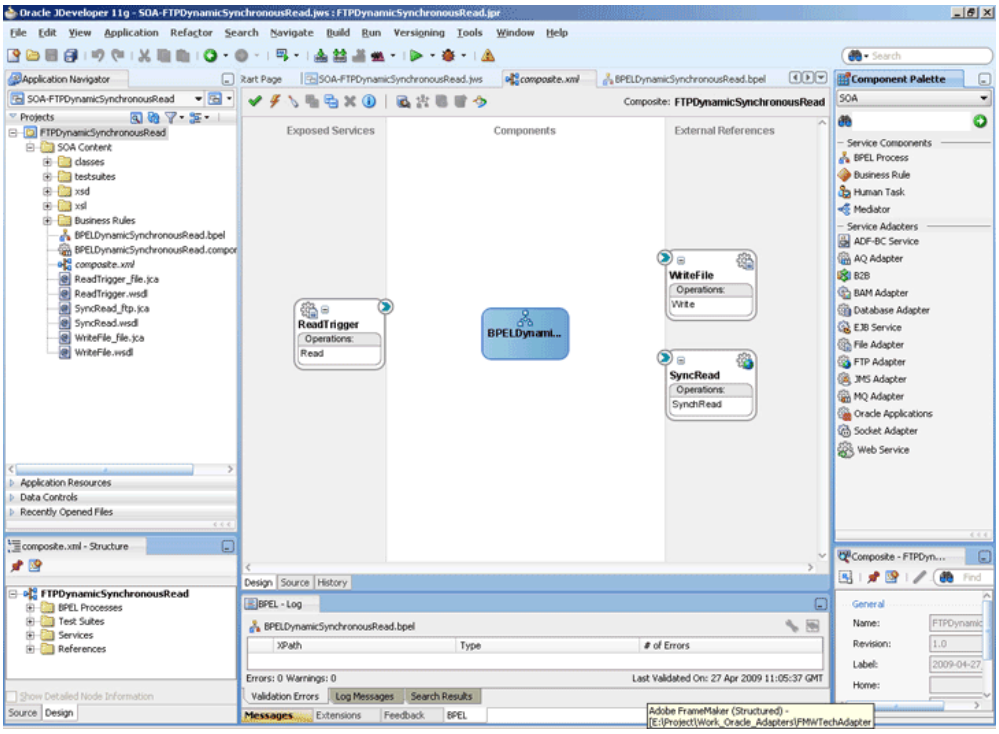


Add an Outbound Oracle File Adapter Service

1. Drag and drop the Oracle File Adapter from the **Components** window to the **External References** swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.
2. Click **Next**. The **Service Name** page is displayed.

3. Enter `WriteFile` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The **Operation** page is displayed.
6. Select **Write File**, enter `Write` in the **Operation Name** field, and then click **Next**. The File Configuration page is displayed.
7. Enter the physical path for the output directory, enter `address_%SEQ%.txt` in the **File Naming Convention (po_%SEQ%.txt)**.
8. Click **Next**. The **Messages** page is displayed.
9. Click **Browse For Schema File** that appears at the end of the URL field. The **Type Chooser** dialog is displayed.
10. Click **Project Schema Files**, **address-fixedLength.xsd**, and **Root-Element**.
11. Click **OK**. The URL field in the **Messages** page is populated with the `address-fixedLength.xsd` file.
12. Click **Next**. The **Finish** page is displayed.
13. Click **Finish**. The outbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4-169.

Figure 4-169 The JDeveloper - Composite.xml



Wiring Services and Activities

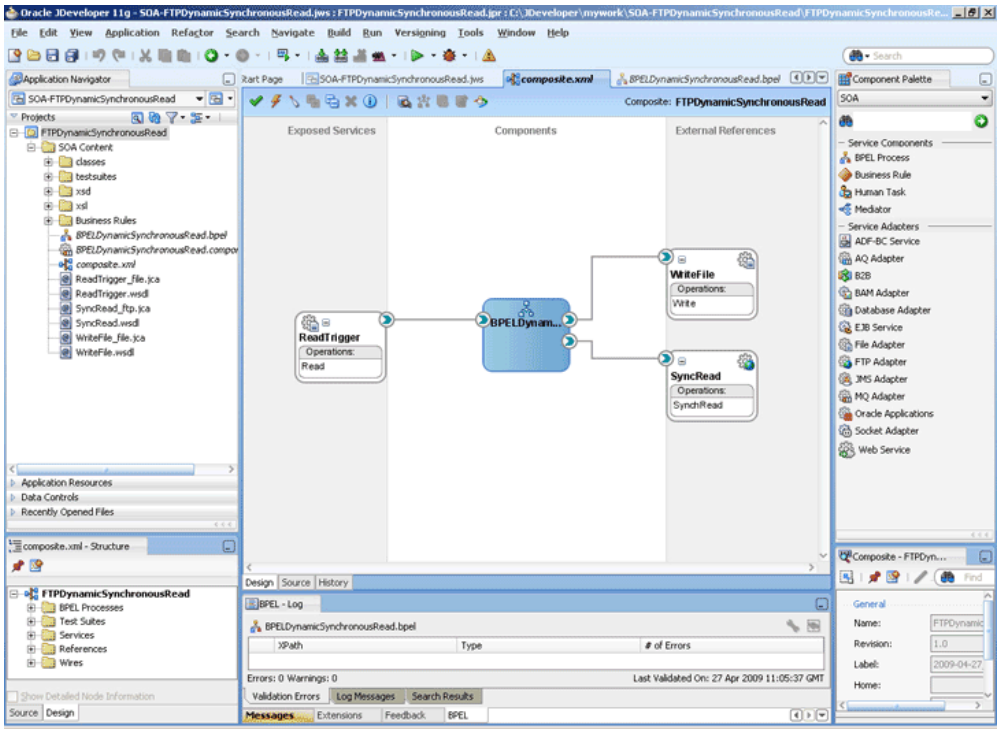
You have to assemble or wire the four components that you have created: Inbound adapter service, BPEL process, two Outbound adapter references. Perform the following steps to wire the components:

1. Drag the small triangle in the ReadTrigger in the **Exposed Services** area to the drop zone that appears as a green triangle in the BPEL process in the **Components** area.

2. Drag the small triangle in the BPEL process in the **Components** area to the drop zone that appears as a green triangle in the SyncRead in the **External References** area and to the drop zone that appears as a green triangle in WriteFile.

   The JDeveloper `composite.xml` appears, as shown in Figure 4-170.

Figure 4-170 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.

Add a Receive Activity

1. Double-click **BPELDynamicSynchronousRead**. The BPELDynamicSynchronousRead.bpel page is displayed.

2. Drag and drop a **Receive** activity from the **Components** window to the design area.

3. Double-click the **Receive** activity. The **Receive** dialog is displayed.

4. Enter `ReceiveTrigger` in the **Name** field.

5. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.

6. Select **ReadTrigger**, and click **OK**.

7. Click the **Auto-Create Variable** icon to the right of the Variable field in the **Receive** dialog. The **Create Variable** dialog is displayed.

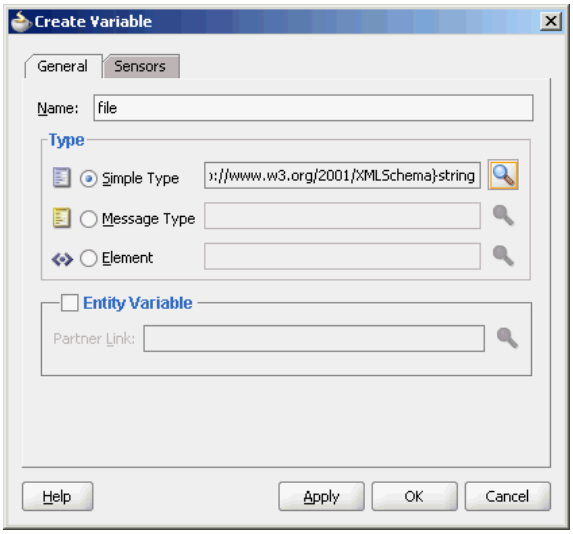
8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.

9. Check **Create Instance**, and click **OK**. The JDeveloper **BPELDynamicSynchronousRead.bpel** page appears.

Create a Variable and add an Invoke Activity

1. Click the **Variables...** icon represented by (x). The **Variables** dialog is displayed.
2. Click the **Create...** icon. The **Create Variable** dialog is displayed.
3. Create a variable called *file* of type xsd:string, as shown in Figure 4-171.

Figure 4-171 The Create Variable Dialog



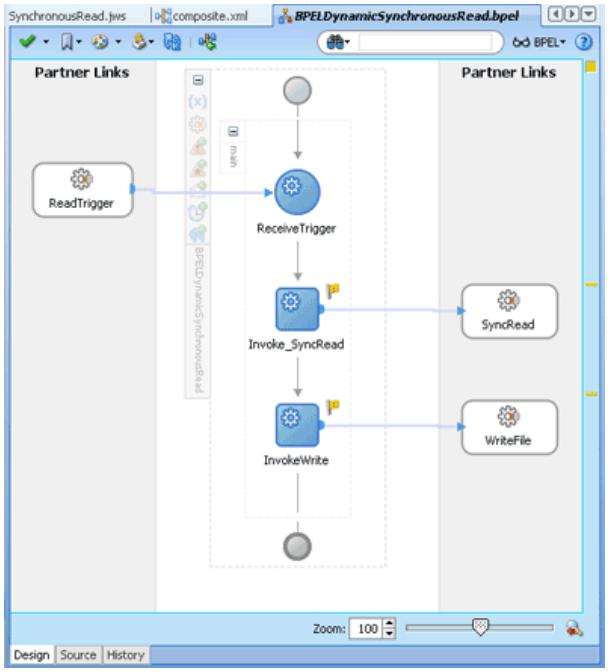
4. Click **OK** to return to BPELDynamicSynchronousRead.bpel page.
5. Drag and drop an **Invoke** activity from the **Components** window to the design area.
6. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.
7. Enter `Invoke_SyncRead` in the **Name** field.
8. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.
9. Select **SyncRead**, and click **OK**.
10. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The **Create Variable** dialog is displayed.
11. Select the default variable name and click **OK**. The Input variable field is populated with the default variable name.
12. Repeat the same for the Output Variable field.
13. Click the **Properties** tab. The properties and the corresponding value column is displayed.
14. Select `jca.ftp.FileName` property. Double-click in the corresponding value column. The **Adapter Property value** dialog is displayed.

15. Click the **Browse variables** icon. The **Variable XPath Builder** dialog is displayed.
16. Expand **Variables**, select **file**, and then click **OK**. The value of the `jca.ftp.FileName` is set to `file`.
17. Click **OK**. The JDeveloper **BPELDynamicSynchronousRead.bpel** page appears.

Add Another Invoke Activity

1. Drag and drop an **Invoke** activity from the Components window to the design area.
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `InvokeWrite` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.
5. Select **WriteFile**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The **Create Variable** dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
8. Click **OK**. The JDeveloper BPELDynamicSynchronousRead.bpel page appears, as shown in Figure 4-172.

Figure 4-172 The JDeveloper - BPELDynamicSynchronousRead.bpel Page

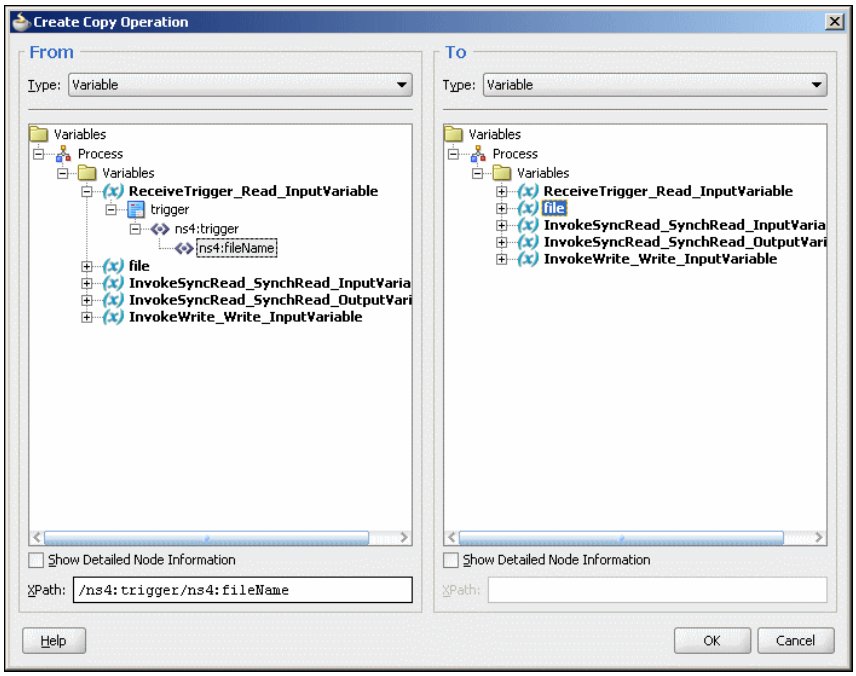


Add an Assign Activity

1. Drag and drop an **Assign** activity from the **Components** window in between the ReceiveTrigger and Invoke_SyncRead activities in the design area.

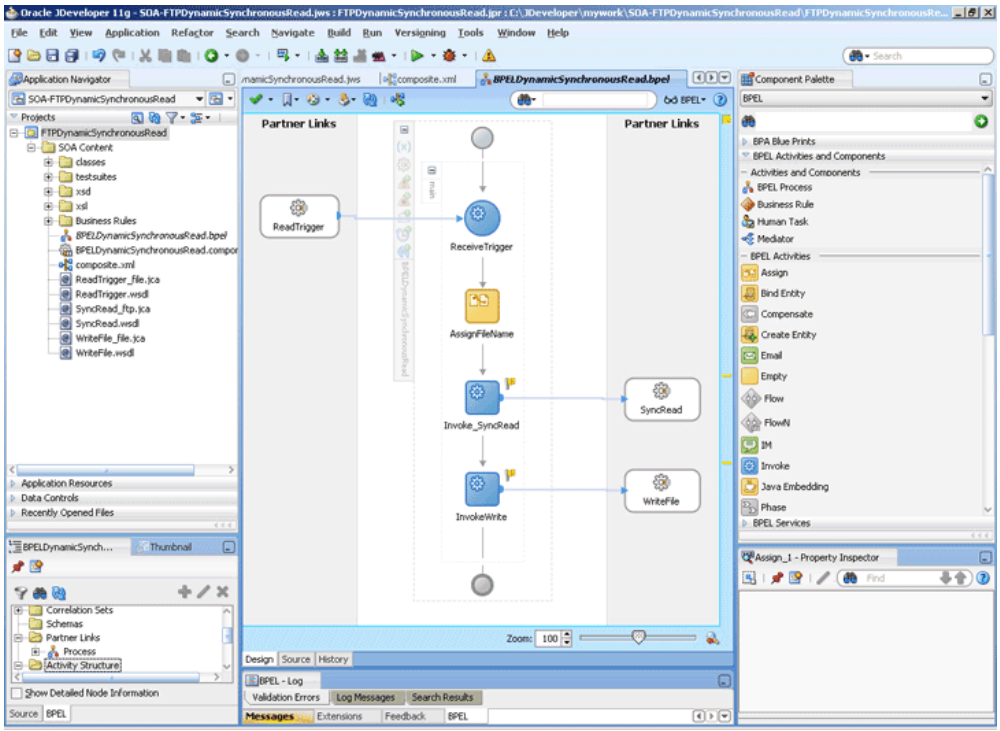
2. Double-click the **Assign** activity. The **Assign** dialog is displayed.
3. Enter `AssignFileName` in the **Name** field.
4. Click the **Copy Operation** tab. The **Assign** dialog is displayed.
5. Select **Copy Operation**. The **Create Copy Operation** dialog is displayed.
6. Create the copy operation between the trigger file name and the file variable, as shown in Figure 4-173.

Figure 4-173 The Create Copy Operation Dialog



7. Click **OK** in the **Create Copy Operation** dialog.
8. Click **OK** to return to the JDeveloper BPELDynamicSynchronousRead.bpel page, as shown in Figure 4-174.

Figure 4-174 The JDeveloper - BPELDynamicSynchronousRead.bpel

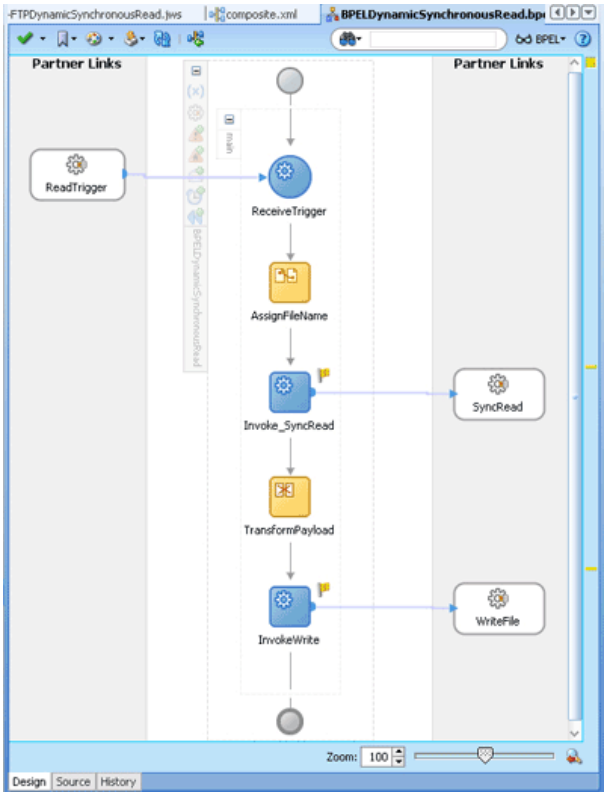


9. Click **File**, **Save All**.

Add a Transform Activity

1. Drag and drop a **Transform** activity from the **Components** window in between the Invoke_SyncRead and InvokeWrite activities in the design area.
2. Double-click the **Transform** activity. The **Transform** dialog is displayed.
3. Enter `TransformPayload` in the **Name** field.
4. Click the **Transformation** tab. The **Transform** dialog is displayed.
5. Click the **Create...** icon. The **Source Variable** dialog is displayed.
6. Select **InvokeSyncRead_SyncRead_OutputVariable** in the **Source Variable** box, and select **body** in the **Source Part** box, and then click **OK**. The **Transform** dialog is displayed with the Source and Part selected.
7. Select **InvokeWrite_Write_InputVariable** in the Target Variable list, select **body** in the Target Part.
8. Click the **Browse Mappings** icon at the end of the **Mapper File** field and select `addr1Toaddr2.xsl` file from the xsl directory in your project.
9. Click **OK**.
10. Click **File**, **Save All**. The **BPELDynamicSynchronousRead.bpel** page is displayed, as shown in Figure 4-175.

Figure 4-175 The JDeveloper - BPELDynamicSynchronousRead.bpel



Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from .

Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the `address-csv.txt` file to the input directory (see Prerequisites for the location of this file) and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The **Dashboard** is displayed.

   Note your Instance ID in the **Recent Instances** area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.

5. Click the Instance ID that you noted in Step 3. The **Flow Trace** page is displayed.
6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow.
9. Click **ReceiveTrigger** to display the activity details.

Copying, Moving, and Deleting Files

The Oracle File and FTP Adapters let you copy or move a file from one location to another, or delete a file from the target directory. Additionally, the Oracle FTP Adapter lets you move or copy files from a local file system to a remote file system and from a remote file system to a local file system. This feature is implemented as a interaction specification for outbound services. So, this feature can be accessed either by using a BPEL invoke activity or a Mediator routing rule.

At a high level, you must create an outbound service and configure this service with the source and target directories and file names.

The following use cases demonstrate the new functionality supported by Oracle File and FTP Adapters that allow you to copy, move, and delete files by using an outbound service:

- Moving a File from a Local Directory on the File System to Another Local Directory
- Copying a File from a Local Directory on the File System to Another Local Directory
- Deleting a File from a Local File System Directory
- Using a Large CSV Source File
- Moving a File from One Remote Directory to Another Remote Directory on the Same FTP Server
- Moving a File from a Local Directory on the File System to a Remote Directory on the FTP Server
- Moving a File from a Remote Directory on the FTP Server to a Local Directory on the File System
- Moving a File from One FTP Server to another FTP Server

Moving a File from a Local Directory on the File System to Another Local Directory

You can model only a part of this procedure by using the wizard because the corresponding Adapter Configuration Wizard is not available. You must complete the remaining procedure by manually configuring the generated JCA file.

You must perform the following steps to move a file from a local directory on the file system to another local directory:

1. Create an empty BPEL process.
2. Drag and drop File Adapter from the **Components** window to the **External References** swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.

3. Click **Next**. The **Service Name** page is displayed.
4. Enter a service name in the **Service Name** field.
5. Click **Next**. The **Adapter Interface** page is displayed.
6. Select **Define from operation and schema (specified later)**, and click **Next**. The **Operation** page is displayed.
7. Select **Synchronous Read File**, enter `FileMove` in the **Operation Name** field, and then click **Next**. The **File Directories** page is displayed.

   Note:

   You have selected Synchronous Read File as the operation because the `WSDL` file that is generated because this operation is similar to the one required for the file I/O operation.

8. Enter a dummy physical path for the directory for incoming files, and then click **Next**. The File name page is displayed.

   Note:

   The dummy directory is not used. You must manually change the directory in a later step.

9. Enter a dummy file name, and then click **Next**. The **Messages** page is displayed.

   Note:

   The dummy file name you enter is not used. You must manually change the file name in a later step.

10. Select **Native format translation is not required (Schema is opaque)**, and then click **Next**. The **Finish** page is displayed.
11. Click **Finish**. The outbound Oracle File Adapter is now configured.
12. Drag the small triangle in the BPEL process in the **Components** area to the drop zone that appears as a green triangle in `FileMove` in the **External References** area. The BPEL component is connected to the Oracle File Adapter outbound service.
13. Create an invoke activity for the `FileMove` service that you just created by selecting the default settings.

    The next step is to modify the generated `WSDL` file for `MoveFileService` service and configure it with the new interaction specification for the move operation.

14. Open the `FileMove_file.jca` file and modify the endpoint interaction, as shown in the following example.

    You must configure the JCA file with the source and target directory and file details. You can either hardcode the source and target directory and file details in

the JCA file or use header variables to populate them. In this example, header variables are used.

Example - Configuring the JCA File with Source and Target Directory and File Details

```
<adapter-config name="FileMove" adapter="File Adapter" xmlns="http://
platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter" adapterRef=""/>
  <endpoint-interaction portType="FileMove_ptt" operation="FileMove">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.FileIoInteractionSpec">
      <property name="SourcePhysicalDirectory" value="foo1"/>
      <property name="SourceFileName" value="bar1"/>
      <property name="TargetPhysicalDirectory" value="foo2"/>
      <property name="TargetFileName" value="bar2"/>
      <property name="Type" value="MOVE"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

Note:

You have modified the `className` attribute, and added `SourcePhysicalDirectory`, `SourceFileName`,`TargetPhysicalDirectory`, `TargetFileName` and `Type`. Currently, the values for the source and target details are dummy. You must populate them at runtime. You can also hardcode them to specific directories or file names.

The `Type` attributes des the type of operation. Apart from `MOVE`, the other acceptable values for the `Type` attribute are `COPY` and `DELETE`.

---

15. Map the actual directory and file names to the source and target file parameters by performing the following procedure:

    a. Create 4 string variables with appropriate names. You must populate these variables with the source and target directory details. The BPEL source view shows you this:

    ```
    <variables>
        <variable name="InvokeMoveOperation_FileMove_InputVariable"
    messageType="ns1:Empty_msg"/>
        <variable name="InvokeMoveOperation_FileMove_OutputVariable"
    messageType="ns1:FileMove_msg"/>
        <variable name="sourceDirectory" type="xsd:string"/>
        <variable name="sourceFileName" type="xsd:string"/>
        <variable name="targetDirectory" type="xsd:string"/>
        <variable name="targetFileName" type="xsd:string"/>
      </variables>
    ```

    b. Create an assign activity to assign values to `sourceDirectory`, `sourceFileName`, `targetDirectory`, and `targetFileName` variables. The assign operation appears in the BPEL source view as in the following example:

    Example - Creating an Assign Activity to Assign Values

```
<assign name="AssignFileDetails">
      <copy>
        <from expression="'/home/alex'"/>
        <to variable="sourceDirectory"/>
      </copy>
      <copy>
        <from expression="'input.txt'"/>
        <to variable="sourceFileName"/>
      </copy>
      <copy>
        <from expression="'/home/alex'"/>
        <to variable="targetDirectory"/>
      </copy>
      <copy>
        <from expression="'output.txt'"/>
        <to variable="targetFileName"/>
      </copy>
    </assign>
```

In the preceding example, `input.txt` is moved from `/home/alex` to `output.txt` in `/home/alex`.

Note:

The source and target details are hard-coded in the preceding example. You can also provide these details as runtime parameters.

---

c. Pass these parameters as headers to the Invoke operation. The values in these variables override the parameters in the JCA file.

Example - Passing Parameters as Headers to the Invoke Operation

```
<invoke name="InvokeMoveOperation"
            inputVariable="InvokeMoveOperation_FileMove_InputVariable"
            outputVariable="InvokeMoveOperation_FileMove_OutputVariable"
            partnerLink="FileMove" portType="ns1:FileMove_ptt"
            operation="FileMove">
      <bpelx:inputProperty name="jca.file.SourceDirectory"
variable="sourceDirectory"/>
      <bpelx:inputProperty name="jca.file.SourceFileName"
variable="sourceFileName"/>
      <bpelx:inputProperty name="jca.file.TargetDirectory"
variable="targetDirectory"/>
      <bpelx:inputProperty name="jca.file.TargetFileName"
variable="targetFileName"/>
    </invoke>
```

16. Finally, add an initial Receive or Pick activity.

    You have completed moving a file from a local directory on the file system to another local directory.

Copying a File from a Local Directory on the File System to Another Local Directory

Perform the following procedure to copy a file from a local directory on the file system to another local directory:

1. Follow steps 1 through 12 of Moving a File from a Local Directory on the File System to Another Local Directory.

2. Change the value of the TYPE attribute to COPY instead of MOVE in the endpoint interaction, in Step 14 of Moving a File from a Local Directory on the File System to Another Local Directory as shown in the following example:

   Example - Value of Type Property Name Changed to Copy

```
<adapter-config ...>
  <connection-factory .../>
  <endpoint-interaction ...>
    <interaction-spec
className="oracle.tip.adapter.file.outbound.FileIoInteractionSpec">
      <property .../>
      <property name="Type" value="COPY"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

Deleting a File from a Local File System Directory

To delete a file, you require TargetPhysicalDirectory and TargetFileName parameters.

> **Note:**
>
> You do not require SourcePhysicalDirectory and SourceFileName to delete a file from a local file system directory.

To delete a file, delete_me.txt, from /home/alex directory, you must perform the following:

1. Follow steps 1 through 12 in Moving a File from a Local Directory on the File System to Another Local Directory

2. Change the value of the TYPE attribute to DELETE in the endpoint interaction in Step 14 of Moving a File from a Local Directory on the File System to Another Local Directory, as shown in the following example:

   Example - Changing the Value of the TYPE Attribute to DELETE

```
<adapter-config name="FileDelete" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter" adapterRef=""/>
  <endpoint-interaction portType="FileDelete_ptt" operation="FileDelete">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.FileIoInteractionSpec">
      <property name="TargetPhysicalDirectory" value="/home/alex"/>
      <property name="TargetFileName" value="delete_me.txt"/>
      <property name="Type" value="DELETE"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

Using a Large CSV Source File

Consider the following scenario, where you have a large CSV file of size 1 gigabyte coming on the source directory, and you must perform the following:

1. Translate the CSV into XML.

2. Transform the resulting XML using XSL.
3. Translate the result from the Transform operation into a fixed length file.

This use case is similar to the `FlatStructure` sample in the BPEL samples directory. The difference is that the three steps occur in a single File I/O interaction.

Note:

All the three steps occur in a single File I/O interaction. This works only if all the records in the data file are of the same type.

To use a large CSV file and perform the operations listed in the preceding scenario, you must perform the following steps:

1. Copy the `address-csv.xsd` and `address-fixedLength.xsd` files from the FlatStructure sample into the `xsd` directory of your project.
2. Copy `addr1Toaddr2.xsl` from the FlatStructure sample into the `xsl` directory of your project.
3. Configure the File I/O interaction, as shown in the following example. At a high level, you must specify the source schema, the target schema, and the XSL in the interaction specification along with the source and target directory or file details, as shown in the following example:

   Example - File I/O Interaction

```
<adapter-config name="FileMove" adapter="File Adapter" xmlns="http://
platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter" adapterRef=""/>
  <endpoint-interaction portType="FileMove_ptt" operation="FileMove">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.FileIoInteractionSpec">
      <property name="SourcePhysicalDirectory" value="foo1"/>
      <property name="SourceFileName" value="bar1"/>
      <property name="SourceSchema" value="xsd/address-csv.xsd"/>
      <property name="SourceSchemaRoot value="Root-Element"/>
      <property name="SourceType" value="native"/>
      <property name="TargetPhysicalDirectory" value="foo2"/>
      <property name="TargetFileName" value="bar2"/>
      <property name="TargetSchema" value="xsd/address-fixedLength.xsd"/>
      <property name="TargetSchemaRoot value="Root-Element"/>
      <property name="TargetType" value="native"/>
      <property name="Xsl value="xsl/addr1Toaddr2.xsl"/>
      <property name="Type" value="MOVE"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

   You have provided the following additional parameters:

   - `SourceSchema`: Relative path to the source schema.
   - `SourceSchemaRoot`: The root element in the source schema.
   - `SourceType`: The type of data. The other possible type is XML.
   - `TargetSchema`: Relative path to the target schema.

- `TargetSchemaRoot`: The root element in the target schema.
- `TargetType`: The type of data. The other possible type is XML.
- `Xsl`: Relative path to the Xsl file.

Moving a File from One Remote Directory to Another Remote Directory on the Same FTP Server

The I/O use cases for the Oracle FTP Adapter are very similar to those for Oracle File Adapter. However, there are a few nuances that require attention.

In this use case you move a file within the same directory, which is similar to a rename operation on the same server. Most FTP servers support the `RNFR/RNTO` FTP commands that let you rename a file on the FTP server.

However, even if the `RNFR/RNTO` commands are not supported, moving a file within the same directory is still possible because of a binding property, `UseNativeRenameOperation`. By default, this property is set to `TRUE`, and in this case the Oracle FTP Adapter uses the native `RNFR/RNTO` commands. However, if this property is set to `FALSE`, then the Oracle FTP Adapter uses the `Get` and `Put` commands followed by a `Delete` command to emulate a move operation.

You can model only a part of this procedure by using the wizard because the corresponding Adapter Configuration Wizard is not available. You must complete the remaining procedure by manually configuring the generated JCA file.

You must perform the following steps to move a file from a remote directory to another remote directory on the same FTP server:

1. Create an empty BPEL process.
2. Drag and drop FTP Adapter from the **Components** window to the External References swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.
3. Click **Next**. The **Service Name** page is displayed.
4. Enter a service name in the **Service Name** field.
5. Click **Next**. The **Adapter Interface** page is displayed.
6. Click **Next**. The **FTP Server Connection** page is displayed.
7. Enter the JNDI name for the FTP server, and click **Next**. The **Operation** page is displayed.
8. Select **Synchronous Get File**, enter *FTPMove* in the **Operation Name** field, and then click **Next**. The **File Directories** page is displayed.

   Note:

   You have selected Synchronous Get File as the operation because the `WSDL` file that is generated because this operation is similar to the one required for the file I/O operation.

   ---

9. Enter a dummy physical path for the directory for incoming files, and then click **Next**. The File name page is displayed.

Note:

The dummy directory is not used. You must manually change the directory in a later step.

10. Enter a dummy file name, and then click **Next**. The File Name page is displayed.

Note:

The dummy file name you enter is not used. You must manually change the file name in a later step.

11. Click **Next**. The **Messages** page is displayed.
12. Select **Native format translation is not required (Schema is opaque)**, and then click **Next**. The **Finish** page is displayed.
13. Click **Finish**. The outbound Oracle File Adapter is now configured.
14. Drag the small triangle in the BPEL process in the **Components** area to the drop zone that appears as a green triangle in `FTPMove` in the **External References** area. The BPEL component is connected to the Oracle FTP Adapter outbound service.
15. Click **File**, **Save All**.
16. Create an invoke activity for the `FTPMove` service that you just created.

    The next step is to modify the generated `WSDL` file for `FTPMove` service and configure it with the new interaction specification for the move operation.
17. Open the `FTPMove_ftp.jca` file and modify the `interaction-spec`, as shown in the following example.

    You must configure the JCA file with the source and target directory and file details. You can either hard-code the source and target directory and file details in the JCA file or use header variables to populate them.

    In this example, header variables are used.

    Example - Modifying the interaction-spec Part of the jca File for the Move Operation

```
<adapter-config name="FTPMove" adapter="Ftp Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">

  <connection-factory location="eis/Ftp/FtpAdapter" adapterRef=""/>
  <endpoint-interaction portType="FTPMove_ptt" operation="FTPMove">
    <interaction-spec
className="oracle.tip.adapter.ftp.outbound.FTPIoInteractionSpec">
      <property name="SourcePhysicalDirectory" value="foo1"/>
      <property name="SourceFileName" value="bar1"/>
      <property name="TargetPhysicalDirectory" value="foo2"/>
      <property name="TargetFileName" value="bar2"/>
      <property name="Type" value="MOVE"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

Note:

You have modified the `className` attribute, and added `SourcePhysicalDirectory`, `SourceFileName`, `TargetPhysicalDirectory`, `TargetFileName`, and `Type`. Currently, the values for the source and target details are dummy. You must populate them at runtime. You can also hardcode them to specific directories or file names.

The `Type` attributes des the type of operation. Apart from `MOVE`, the other acceptable values for the `Type` attribute are `COPY` and `DELETE`.

18. Map the actual directory and file names to the source and target file parameters by performing the following procedure:

    a. Create 4 string variables with appropriate names. You must populate these variables with the source and target directory details. The BPEL source view shows you this:

```
  <variables>
    <variable name="InvokeMoveOperation_FileMove_InputVariable"
messageType="ns1:Empty_msg"/>
    <variable name="InvokeMoveOperation_FileMove_OutputVariable"
messageType="ns1:FileMove_msg"/>
    <variable name="sourceDirectory" type="xsd:string"/>
    <variable name="sourceFileName" type="xsd:string"/>
    <variable name="targetDirectory" type="xsd:string"/>
    <variable name="targetFileName" type="xsd:string"/>
  </variables>
```

    b. Create an assign activity to assign values to `sourceDirectory`, `sourceFileName`, `targetDirectory`, and `targetFileName` variables. The assign operation appears in the BPEL source view as shown in the example below.

Example - Creating an Assign Activity

```
   <assign name="AssignFTPFileDetails">
     <copy>
       <from expression="'/home/ftp'"/>
       <to variable="sourceDirectory"/>
     </copy>
     <copy>
       <from expression="'input.txt'"/>
       <to variable="sourceFileName"/>
     </copy>
     <copy>
       <from expression="'/home/ftp/out'"/>
       <to variable="targetDirectory"/>
     </copy>
     <copy>
       <from expression="'output.txt'"/>
       <to variable="targetFileName"/>
     </copy>
   </assign>
```

In the preceding example, `input.txt` is moved or renamed from `/home/ftp` to `output.txt` in `/home/ftp/out`.

Note:

The source and target details are hard coded in the preceding example. You can also provide these details as runtime parameters.

c. Pass these parameters as headers to the invoke operation. The values in these variables override the parameters in the JCA file.

Example - Passing Parameters as Headers to the Invoke Operation

```
<invoke name="InvokeRenameService"
inputVariable="InvokeRenameService_RenameFile_InputVariable"
partnerLink="RenameFTPFile" portType="ns2:RenameFile_ptt"
operation="RenameFile">
<bpelx:inputProperty name="jca.file.SourceDirectory"
variable="returnDirectory"/>
<bpelx:inputProperty name="jca.file.SourceFileName"
variable="returnFile"/>
<bpelx:inputProperty name="jca.file.TargetDirectory"
variable="returnDirectory"/>
<bpelx:inputProperty name="jca.file.TargetFileName"
variable="targetFile"/>
</invoke>
```

19. Finally, add an initial receive or pick activity.

You have completed moving or renaming a file from a remote directory to another remote directory on the same FTP server.

Note:

If the FTP server does not support the `RNFR/RNTO` FTP commands, then you must set `UseNativeRenameOperation` to `FALSE` and define the property in `composite.xml`, as shown in the following example:

```
<reference name="FTPMove" ui:wsdlLocation="FTPMove.wsdl">
  <interface.wsdl interface="http://xmlns.oracle.com/pcbpel
    /adapter/ftp/SOAFtpIO/SOAFtpIO/
          FTPMove/#wsdl.interface(FTPMove_ptt)"/>
    <binding.jca config="FTPMove_ftp.jca">
      <property name="UseNativeRenameOperation"
         type="xs:string" many="false" override="may">false</property>
    </binding.jca>
</reference>
```

Moving a File from a Local Directory on the File System to a Remote Directory on the FTP Server

The steps for this use case are the same as the steps for the use case in Moving a File from One Remote Directory to Another Remote Directory on the Same FTP Server except that you must configure the source directory as local and the target directory as remote.

Use the `SourceIsRemote` and `TargetIsRemote` properties to specify whether the source and target file are on the local or remote file system, as shown in the following example:

Example - Using the SourceIsRemote and TargetIsRemote Properties

```
<adapter-config name="FTPMove" adapter="Ftp Adapter"
xmlns="http://platform.integration.oracle/
                  blocks/adapter/fw/metadata">
  <connection-factory location="eis/Ftp/FtpAdapter"
                         adapterRef=""/>
  <endpoint-interaction portType="FTPMove_ptt"
                        operation="FTPMove">
    <interaction-spec
className="oracle.tip.adapter.ftp.outbound.FTPIoInteractionSpec">
      <property name="SourcePhysicalDirectory" value="foo1"/>
      <property name="SourceFileName" value="bar1"/>
      <property name="SourceIsRemote" value="false"/>
      <property name="TargetPhysicalDirectory" value="foo2"/>
      <property name="TargetFileName" value="bar2"/>
      <property name="Type" value="MOVE"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

Note:

In this example, you have configured `SourceIsRemote` as `false`. In this case, the FTP input and output operation assumes that the source file comes from a local file system. Also, notice that you did not specify the parameter for target because `TargetIsRemote` is set to `true` by default.

Moving a File from a Remote Directory on the FTP Server to a Local Directory on the File System

The steps for this use case are the same as the steps for the use case in Moving a File from a Local Directory on the File System to a Remote Directory on the FTP Server except that you must configure the source directory as remote and the target directory as local, as shown in the following example:

Example - Configuring the Source Directory as Remote and the Target Directory as Local

```
<adapter-config name="FTPMove" adapter="Ftp Adapter"
xmlns="http://platform.integration.oracle
                /blocks/adapter/fw/metadata">
  <connection-factory location="eis/Ftp/FtpAdapter"
               adapterRef=""/>
  <endpoint-interaction portType="FTPMove_ptt"
                     operation="FTPMove">
    <interaction-spec
className="oracle.tip.adapter.ftp.outbound.FTPIoInteractionSpec">
      <property name="SourcePhysicalDirectory" value="foo1"/>
      <property name="SourceFileName" value="bar1"/>
      <property name="TargetPhysicalDirectory" value="foo2"/>
      <property name="TargetFileName" value="bar2"/>
      <property name="TargetIsRemote" value="false"/>
      <property name="Type" value="MOVE"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

Note:

In this example, you have configured `TargetIsRemote` as `false`. In this case, the FTP I/O assumes that the source file comes from a remote file system whereas the target is on a local file system. Also, notice that you did not specify the parameter for source because `SourceIsRemote` is set to true by default.

Moving a File from One FTP Server to another FTP Server

To move a file from one FTP server to another FTP server you must sequentially perform the use cases documented in the following sections:

1. Moving a File from a Remote Directory on the FTP Server to a Local Directory on the File System to upload the file from the local directory to another FTP server
2. Moving a File from a Local Directory on the File System to a Remote Directory on the FTP Server to download the file from the FTP server to a local directory

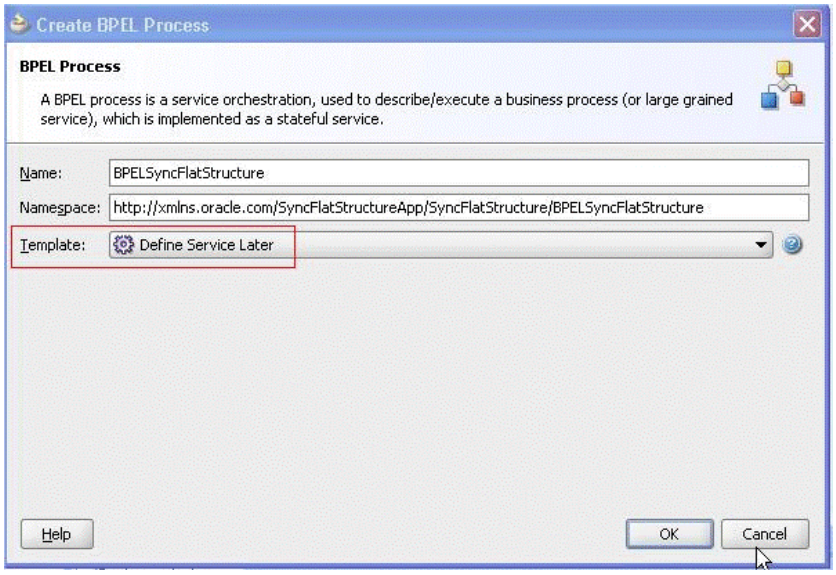
Creating a Synchronous BPEL Composite using File Adapter

By default, the JDeveloper Adapter Wizard generates asynchronous WSDLs when you use technology adapters. Typically, you follow these steps when creating an adapter scenario in 11g:

1. Create a SOA Application with either **Composite with BPEL** or an **Empty Composite** selected. If you choose **Empty Composite**, you must drop the BPEL Process from the **Service Components** pane onto the SOA Composite Editor.

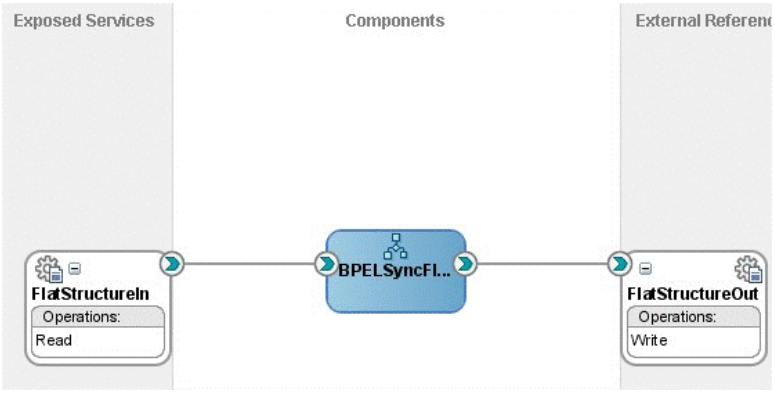
   You arrive at the screen shown in Figure 4-176 where you fill in the process details. You are required to choose **Define Service Later** as the template.

   Figure 4-176 Create Process Screen

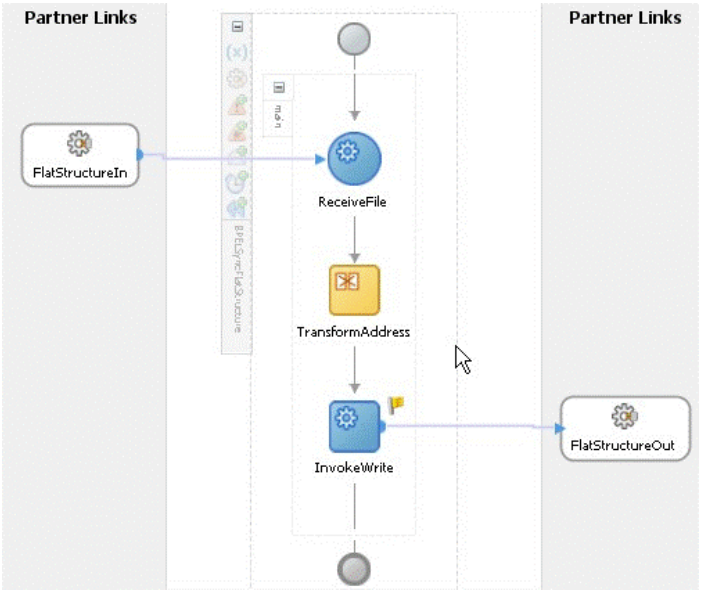
2. Next, you create the inbound service and outbound references and wire them with the BPEL component.

Figure 4-177 Cresting the Inbound Service and Outbound References



3. And, finally you create the BPEL process with the initiating <receive> activity to retrieve the payload and an **<invoke>** activity to write the payload

Figure 4-178 Receive Activity and Invoke Activity



This is how most BPEL processes that use Adapters are modeled. The generated WSDL implies one-way directionally one way and that makes the BPEL process asynchronous:

Figure 4-179 A Generated One-Way WSDL


```
<?binding.jca FlatStructureIn_file.jca?>
<wsdl:definitions name="FlatStructureIn"
                  targetNamespace="http://xmlns.oracle.com/...." ...>
    <plt:partnerLinkType name="Read_plt">
        <plt:role name="Read_role">
            <plt:portType name="tns:Read_ptt"/>
        </plt:role>
    </plt:partnerLinkType>
    <wsdl:types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://xmlns.oracle.com/pcbpel/demoSchema/csv"
              schemaLocation="xsd/address-csv.xsd"/>
        </schema>
    </wsdl:types>
    <wsdl:message name="Read_msg">
        <wsdl:part name="body" element="imp1:Root-Element"/>
    </wsdl:message>
    <wsdl:portType name="Read_ptt">
        <wsdl:operation name="Read">
            <wsdl:input message="tns:Read_msg"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

In other words, the inbound File Adapter polls for files in the directory and for each file that it finds there, it translates the content into XML and publishes to BPEL.

However, because the BPEL process is asynchronous, the File Adapter returns immediately after the publish and performs the required post processing-for example. deletion/archival of data.

The disadvantage with such asynchronous BPEL processes is that it becomes difficult to throttle the inbound adapter. In other words, the inbound adapter would keep sending messages to BPEL without waiting for the downstream business processes to complete. This can lead to issues such as higher memory usage and CPU usage.

To mitigate the occurrence of these problems, you can manually change the WSDL and BPEL artifacts into synchronous processes. Once you have changed the synchronous to synchronous BPEL processes, the inbound File Adapter automatically throttles itself because the File Adapter is forced to wait for the downstream process to complete with a <reply> before processing the next file or message.

Refer to the altered WSDL shown in Figure 4-180. Here, you convert the one-way WSDL to a two-way WSDL-- thereby making the WSDL synchronous.

Figure 4-180 Asynchronous WSDL Altered to be Two-Way WSDL

```
<?binding.jca FlatStructureIn_file.jca?>
<wsdl:definitions name="FlatStructureIn"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/SyncFlatStructureApp/SyncFlatStructure/FlatStructureIn%2F"
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/file/SyncFlatStructureApp/SyncFlatStructure/FlatStructureIn%2F"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:impl="http://xmlns.oracle.com/pcbpel/demoSchema/csv" xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
    <plt:partnerLinkType name="Read_plt">
        <plt:role name="Read_role">
            <plt:portType name="tns:Read_ptt"/>
        </plt:role>
    </plt:partnerLinkType>
    <wsdl:types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace=
"http://xmlns.oracle.com/pcbpel/adapter/file/SyncFlatStructureApp/SyncFlatStructure/FlatStructureIn%2F">
            <import namespace="http://xmlns.oracle.com/pcbpel/demoSchema/csv"
              schemaLocation="xsd/address-csv.xsd"/>
              <element name="empty">
                <complexType/>
              </element>
        </schema>
    </wsdl:types>

    <wsdl:message name="Read_msg">
        <wsdl:part name="body" element="impl:Root-Element"/>
    </wsdl:message>

    <wsdl:message name="Reply_msg">
        <wsdl:part name="body" element="tns:empty"/>
    </wsdl:message>

    <wsdl:portType name="Read_ptt">
        <wsdl:operation name="Read">
            <wsdl:input message="tns:Read_msg"/>
            <wsdl:output message="tns:Reply_msg"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

targetNamespace same as that of the BPEL Process WSDL

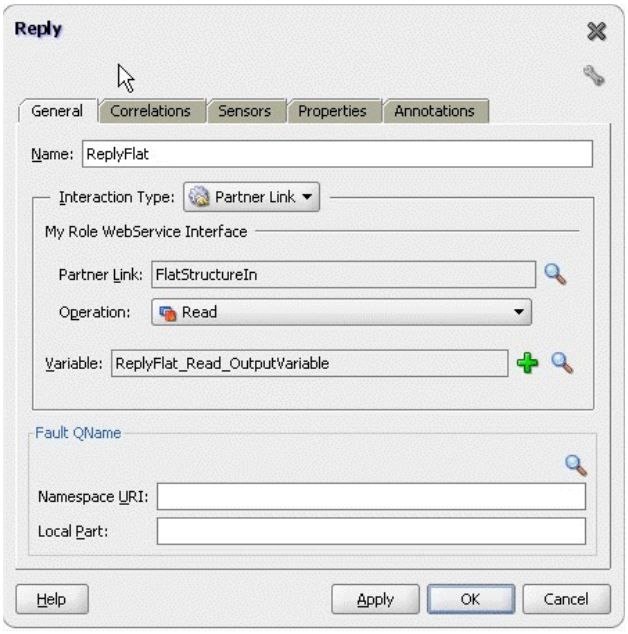
New dummy element

New WSDL Message

New Output Message

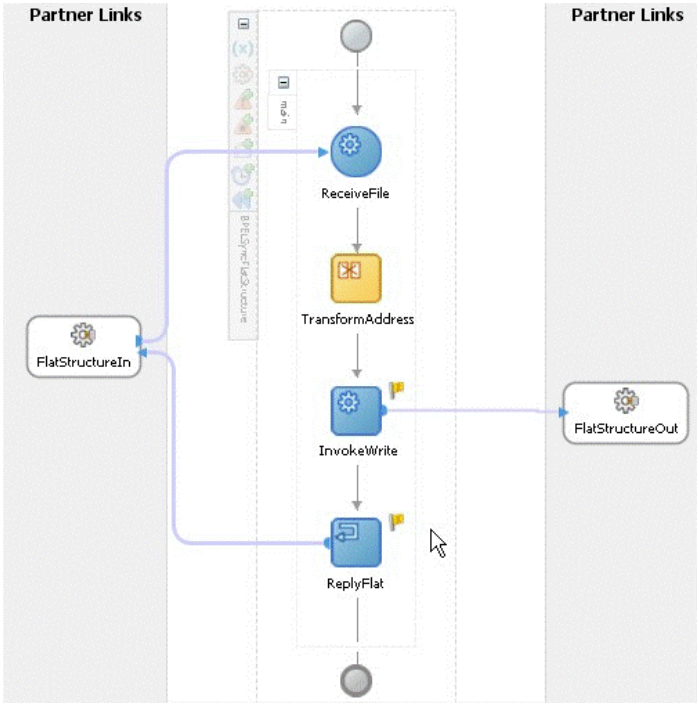
The next step is to add a <reply> activity to the inbound adapter Partner Link at the end of your BPEL process, for example:

Figure 4-181 Specifying a Reply to the Inbound Adapter



Finally, the process looks like Figure 4-182.

Figure 4-182 The Synchronous File Adapter Process with Receive and Reply BPEL Activities



This type of exercise is not required for the Mediator because the Mediator routing rules are sequential by default. In other words, the Mediator uses the caller thread (inbound file adapter thread) for processing routing rules. This is the case even if the WSDL for mediator is one-way.

Changing the Connection Factory JNDI Dynamically in the FTP Adapter

Where there is a requirement to send the same file to five different FTP servers, you could create, for example, five FtpAdapter references, one for each connection-factory location. However, this is not the most optimal approach; instead, you can use the concept of Dynamic Partner Links.

If you're running the adapter in managed mode, it requires you to configure the connection factory JNDI in the WebLogic Server console for the FtpAdapter.

In the sample in Figure 4-183, the connection-factory JNDI location `eis/Ftp/FtpAdapter` has been mapped with the FTP server running on localhost.

Figure 4-183 Connection Factory JNDI Location Mapped with the FTP Server Running on Localhost - weblogic.ra Sample

Figure 4-184 Connection Factory Showing Mapping of FTP Adapter

After you have configured the connection factory on your application server, you must refer to the connection-factory JNDI in the jca artifact of your SCA process. In the example, the `FTPOut` reference in the .jca file uses the FTP server corresponding to `eis/Ftp/FtpAdapter`

You can change this connection-factory location dynamically using JCA header properties in both BPEL and Mediator service engines. To do so, the business scenario involving BPEL or Mediator is required to use a reserved JCA header property `jca.jndi` as shown in Figure 4-185.

Figure 4-185 Using the Reserved JCA Header Property jca.jndi

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<process name="BPELDynamicPL"...>
  -->
  <partnerLinks.../>
  <variables>
    <variable name="Receive_File_Get_InputVariable" messageType="ns1:Get_msg"/>
    <variable name="Invoke_FTPServer1_Put_InputVariable" messageType="ns2:Put_msg"/>
    <variable name="jndiLocation" type="xsd:string"/>
  </variables>
  <sequence name="main">
    <receive .../>
    <assign name="Assign_Payload".../>
    <assign name="Assign_FtpServer1_JNDI">
      <copy>
        <from expression="'eis/Ftp/FtpAdapter1'"/>
        <to variable="jndiLocation"/>
      </copy>
    </assign>
    <invoke name="Invoke_FTPServer1"
            inputVariable="Invoke_FTPServer1_Put_InputVariable"
            partnerLink="FTPOut" portType="ns2:Put_ptt" operation="Put">
      <bpelx:inputProperty name="jca.jndi" variable="jndiLocation"/>
    </invoke>

    <assign name="Assign_FtpServer2_JNDI">
      <copy>
        <from expression="'eis/Ftp/FtpAdapter2'"/>
        <to variable="jndiLocation"/>
      </copy>
    </assign>
    <invoke name="Invoke_FTPServer2"
            inputVariable="Invoke_FTPServer1_Put_InputVariable"
            partnerLink="FTPOut" portType="ns2:Put_ptt" operation="Put">
      <bpelx:inputProperty name="jca.jndi" variable="jndiLocation"/>
    </invoke>

  </sequence>
</process>
```

You must remember the following when using dynamic partner links:

- You must preconfigure the connection factories on the SOA server. In the BPEL example, both `eis/Ftp/FtpAdater1` and `eis/Ftp/FtpAdater2` must be configured in the WebLogic Server deployment descriptor for the FtpAdapter before your deployment of the scenario.
- Dynamic Partner Links are applicable to *outbound* invocations only

Retrieving the Details of the File from an Outbound Write Operation

There is an option called **Add Output Header** in the File Adapter Configuration wizard that enables you to obtain output headers in a request/response operation.

Changing the Sequencing Strategy for FILE/FTP Adapter

The File/FTP Adapter enables you to configure outbound writes to use a sequence number. For example, if you choose `address-data_%SEQ%.txt` as the FileNamingConvention, all files would be generated as `address-data_1.txt`, `address-data_2.txt`, ...

See Figure 4-186.

Figure 4-186 Configuring Outbound Writes

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FlatStructureOut" adapter="File Adapter"
 xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">

  <connection-factory location="eis/FileAdapter" adapterRef=""/>
  <endpoint-interaction operation="Write">
    <interaction-spec className="oracle.tip.adapter.file.outbound.FileInteractionSpec">
      <property name="PhysicalDirectory" value="/tmp/file/out"/>
      <property name="FileNamingConvention" value="address-data_%SEQ%.txt"/>
      <property name="Append" value="false"/>
      <property name="NumberMessages" value="1"/>
      <property name="OpaqueSchema" value="false"/>
    </interaction-spec>
  </endpoint-interaction>
```

The sequence number comes from the control directory for the particular adapter project(or scenario). For each project that use the File or Ftp Adapter, a unique directory is created for book-keeping purposes. Because this control directory must be unique, the adapter uses a digest to ensure that no two control directories are the same.

For example, for the `FlatStructure` sample in the example, the control information for my project would go under `FMW_HOME/user_projects/domains/soainfra/fileftp/controlFiles/[DIGEST]/outbound` where the value of `DIGEST` would differ from one project to another.

Within this directory, there is a file `control_ob.properties` where the sequence number is maintained. The sequence number is maintained in binary form and you might require a hexadecimal editor to view its content. There is another zero byte file, `SEQ_nnn`. This extra file is maintained as a backup.

One of the challenges faced by the Adapter runtime is to guard all writes to the control files so no two threads inadvertently attempt to update the control files at the same time. It does this guarding with the help of a mutex.

The mutex is of different types:

- In-memory
- Database-based
- Coherence-based
- User-defined

There might be scenarios, particularly when the Adapter is under heavy transactional load, where the mutex is a bottleneck. The Adapter, however, enables you to change the configuration so the adapter sequence value is derived from a database sequence or a stored procedure. In such a situation, the mutex is by-passed, and the process results in improved throughput.

The simplest way to achieve improved throughput is by switching your JNDI connection factory location for the outbound JCA file to use the `eis/HAFileAdapter`:

Figure 4-187 Switching the JNDI Connection Factory to Use the HAFileAdapter

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FlatStructureOut" adapter="File Adapter"
 xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">

  <connection-factory location="eis/HAFileAdapter" adapterRef=""/>
  <endpoint-interaction operation="Write">
    <interaction-spec className="oracle.tip.adapter.file.outbound.FileInteractionSpec">
      <property name="PhysicalDirectory" value="/tmp/file/out"/>
      <property name="FileNamingConvention" value="address-data_%SEQ%.txt"/>
      <property name="Append" value="false"/>
      <property name="NumberMessages" value="1"/>
      <property name="OpaqueSchema" value="false"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

With this change, the Adapter runtime creates a sequence on the Oracle database. For example, if you do a `select * from user_sequences` in your soa-infra schema, you see a new sequence being created with name as `SEQ_<GUID>__` (where the `GUID` differs by project).

However, to use your *own* sequence, you must add a new property to your JCA file called `SequenceName`. You must create this sequence on your soainfra schema before. See Figure 4-188.

Figure 4-188 Adding the SequenceName Property

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FlatStructureOut" adapter="File Adapter"
 xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">

  <connection-factory location="eis/HAFileAdapter" adapterRef=""/>
  <endpoint-interaction operation="Write">
    <interaction-spec className="oracle.tip.adapter.file.outbound.FileInteractionSpec">
      <property name="PhysicalDirectory" value="/tmp/file/out"/>
      <property name="FileNamingConvention" value="address-data_%SEQ%.txt"/>
      <property name="Append" value="false"/>
      <property name="NumberMessages" value="1"/>
      <property name="OpaqueSchema" value="false"/>
      <property name="SequenceName" value="FILEADAPTER_SEQ"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

The scenario when using a DB2 or MSSQL Server as the dehydration support is a bit different. DB2 supports sequences natively but MSSQL Server does not.

The Adapter runtime uses a natively generated sequence for DB2, but, for MSSQL server, the Adapter relies on a stored procedure that ships with the product.

To achieve the same result for a SOA Suite running DB2 as the dehydration store, change the connection factory JNDI name in the JCA file to `eis/HAFileAdapterDB2`.

To achieve the same result for MSSQL as the dehydration store, use `eis/HAFileAdapterMSSQL`.

To use a stored procedure other than the one that ships with the product, you must rely on binding properties to override the adapter behavior; specifically, you must instruct the adapter to use a stored procedure as in Figure 4-189.

Figure 4-189 Instructing the Adapter to Use a Stored Procedure

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SOA Modeler version 1.0 at [7/24/07 1:26 PM]. -->
<composite name="FlatStructure"...>
  <service name="FlatStructureIn">
    [..]
  </service>
  <component name="BPELFlatStructure">
    <implementation.bpel src="BPELFlatStructure.bpel"/>
  </component>
  <reference name="FlatStructureOut">
    <interface.wsdl interface="..."/>
    <binding.jca config="FlatStructureOut_file.jca">
    <property name="useStoredProcedure" type="xs:string"
      many="false" override="may">true</property>
    <property name="storedProcedure" type="xs:string"
    many="false" override="may">bvtuser1.FILEADAPTERNEXTVALUE</property>
    </binding.jca>
  </reference>
  <wire>
    <source.uri>FlatStructureIn</source.uri>
    <target.uri>BPELFlatStructure/FlatStructureIn</target.uri>
  </wire>
  <wire>
    <source.uri>BPELFlatStructure/FlatStructureOut</source.uri>
    <target.uri>FlatStructureOut</target.uri>
  </wire>
</composite>
```

When the File/FTP Adapter is used in Append mode, the Adapter runtime degrades the mutex to use pessimistic locks to prevent writers from different nodes appending to the same file at the same time.

Controlling the Order in which Files Are Processed

The File/FTP Adapter enables you to control the order in which files get processed. For example, you might want the files to be processed in sequence of their modified times/ file sizes, or other determiners.

The File/FTP adapter enables you to achieve controlling the order in which files gets processed through a FileSorter attribute that you can define in the JCA file for your inbound File/FTP Adapter service. See Figure 4-190.

Figure 4-190 Specifying the File Sorter

```
<adapter-config name="FlatStructureIn" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">

  <connection-factory location="eis/FileAdapter" UIincludeWildcard="*.*" adapterRef=""/>
  <endpoint-activation portType="Read_ptt" operation="Read">
    <activation-spec className="oracle.tip.adapter.file.inbound.FileActivationSpec">
      <property name="UseHeaders" value="false"/>
      <property name="PhysicalDirectory" value="/tmp/flat/in"/>
      <property name="Recursive" value="true"/>
      <property name="SingleThreadModel" value="true"/>
      <property name="ListSorter" value="file.sorter.FileSizeSorter"/>
      <property name="DeleteFile" value="true"/>
      <property name="IncludeFiles" value=".*\..*"/>
      <property name="PollingFrequency" value="1"/>
      <property name="MinimumAge" value="0"/>
    </activation-spec>
  </endpoint-activation>

</adapter-config>
```

Required

The File/FTP Adapter provides two predefined sorters that use the last modified times. For example:

Figure 4-191 Using a Predefined Sorter

```
<adapter-config name="FlatStructureIn" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter" UIincludeWildcard="*.*" adapterRef=""/>
  <endpoint-activation portType="Read_ptt" operation="Read">
    <activation-spec className="oracle.tip.adapter.file.inbound.FileActivationSpec">
      [..]
     <property name="ListSorter" value="oracle.tip.adapter.file.inbound.listing.TimestampSorterAscending"/>
      [or]
      <property name="ListSorter" value="oracle.tip.adapter.file.inbound.listing.TimestampSorterDescending"/>
      [..]
    </activation-spec>
  </endpoint-activation>

</adapter-config>
```

However, there are times when you want to define the sort order yourself. To do so, you can implement a Java Comparator and register that with the File Adapter. Follow these steps to do so:

1. Write a comparator. For example, the `FileSizeSorter` comparator sorts the files in descending order of their sizes. See Figure 4-192.

   Figure 4-192 FileSorterSize Comparator

```
package file.sorter;


import java.util.Comparator;
import oracle.tip.adapter.file.FileInfo;

public class FileSizeSorter implements Comparator {
        public FileSizeSorter() {
            System.out.println("FileSizeSorter constructed");
        }
        public int compare(Object a, Object b) {
                FileInfo first = (FileInfo)a;
                FileInfo second = (FileInfo)b;
                if(first.getSize() < second.getSize())
                        return 1;
                else if(first.getSize() > second.getSize())
                        return -1;
                return 0;

        }
}
```

2. To compile this class, you must place `fileAdapter`.jar in the classpath. From `[FMW_HOME]/AS11gR1SOA/soa/connectors`, expand `FileAdapter.rar`, for example, `jar xvf FileAdapter.rar` to extract `fileAdapter.jar`. You must place `fileAdapter.jar` in your classpath to compile the `FileSizeSorter.java` class, for example:

   ```
   setenv CLASSPATH fileAdapter.jar
   javac -d .FileSizeSorter.java
   ```

3. After compilation, bundle the class in its own jar file, for example,

   ```
   jar cvf fileadapter-sorter.jar file
   ```

4. Bundle the `fileadapter-sorter.jar` in the `FileAdapter.rar`, for example,

   ```
   jar uvf FileAdapter.rar fileadapter-sorter.jar
   ```

   This step is required since the `fileadapter-sorter.jar` becomes visible to the ClassLoader that loads `FileAdapter.rar`. Though, there are other ways for example, copying the `fileadapter-sorter.jar` and `fileadapter.jar` under `[DOMAIN_HOME]/lib`, but the one provided is the simplest and easiest to do.

5. Setting `SingleThreadModel` as `true` in the JCA file (see image at the beginning of this section.)
6. If you are using BPEL, ensure you model a synchronous process.

Extending FTP Adapter

The current FTP Adapter is compliant with RFC 959 that defines the standard FTP commands and valid return codes for each of the commands. However, some FTP servers use proprietary commands; for example, some FTP servers running on mainframes use "QUOTE SITE" commands to send the save format; for example, QUOTE SITE BLKSIZE=30000 (which sets the Block size for to 30000 bytes for the interaction).

Additionally, some FTP servers return non-standard FTP return codes for standard FTP commands that should be handled by the FTP Adapter as well. The FTP Adapter provides a layer of pluggability to handle these slight nuances in the functioning of FTP servers. In summary, there is a requirement to expose the internal of the FTP Adapter for the following reasons:

- Some FTP servers respond to standard FTP commands in different manners.
- Certain FTP servers require the use of proprietary FTP commands.
- Some FTP servers return data in a slightly different format. For example, the LIST command in some FTP servers returns timestamp information as "Aug 29 2011" but it is returned as "29AUG-2011" in others.

The FTP Adapter uses a set of FTP commands in addition to their return codes. To support pluggability, this functionality is implemented by the FTP Adapter:

FTP and File Adapter Extension Use Cases

There are four sets of FTP and File Adapter Extension Use Cases:

- FTP Adapter Extension of FTP Client Login
- Configuring FTP Adapter to Handle Response from MLSD Command
- Extend the Listing Operation to Send MLSD Commands Rather than the LIST Commands
- Extend the Store Operation to Send Additional Proprietary FTP Commands to FTP Server Running on the MVS Platform

FTP Adapter Extension of FTP Client Login

The FTP Adapter can be extended to override the `login()` operation in the default FTPClient implementation.

You can extend the default implementation because the user might be required to specify the Account name while logging into the FTP server. As a part of the authorization process, a typical FTP client normally begins each FTP connection with a USER command followed by the PASS command to the ftp server, for example:

```
USER <SP><user-name><CRLF>
PASS <SP> <password><CRLF>
```

Here, <SP> means space, and <CRLF> means carriage return followed by a Line Feed. However, in certain cases, the FTP server expects an Account name (ACCT command) passed after the PASS command, for example,

```
ACCT <SP><Account Name><CRLF>
```

The Account Name specifies the account name on the server for which the user is requesting authorization.

With this extension, the FTP Adapter is enabled to send the ACCT command during authorization immediately after the PASS command has been sent.

To achieve this behavior, you must override the `login()`method exposed by the FTPClient implementation (shipped with the FTP Adapter).

The client side of the FTP protocol in FTP Adapter is exposed using an interface IFTPClient and the default implementation is provided by FTPClient.

In the FTP Adapter, you can override the default FTPClient implementation and add support for ACCT in the `login()` method.

Once the FTPClient has been extended in this manner, the class name must be configured in the connection factory for FTP Adapter.

Extending FTPClient Implementation to Override login()

Follow these steps to extend the FTP Client.

1. Create `fileftp.jar`. This is a one-time activity and required for compilation only. Refer to Using a Manifest.MF file to Generate a fileftp.jar for Compilation on how to create `fileftp.jar`.
2. Add `fileftp.jar` (which you created in the step previously) to the classpath; for example,

   ```
   set CLASSPATH=$ORACLE_SOA_HOME/soa/modules/oracle.soa.adapter_11.1.1/
   fileftp.jar
   ```

3. Create `MyFtpClient.java`. Refer to Sample FTPClient Implementationon how to override the `login()`method to add support for ACCT command.
4. Compile `MyFTPClient.java`.

   ```
   javac -d . MyFTPClient.java
   ```

   Once compiled, package it as a jar, for example, `myftpclient.jar`

   ```
   jar cvf myftpclient.jar << compile folder>>
   ```

5. Copy`myftpclient.jar` under `$FMW_HOME/soa/soa/modules/oracle.adapter.ext_12.1.2`

   ```
   cp myftpclient.jar $FMW_HOME/soa/soa/modules/oracle.adapter.ext_12.1.2
   ```

6. Run `ant` to add the `myftpclient.jar` to adapter classpath.

   ```
   ant
   ```

7. Set the Connection factory entry `FtpClientClass="com.acme.MyFTPClient"` in the WebLogic deployment descriptor for the FTP Adapter.

This enables the FTP Adapter to use `MyFTPClient` instead of the default FTPClient.

8. Restart the server.

Configuring FTP Adapter to Handle Response from MLSD Command

The FTP Adapter sends the LIST command to return a list of files from the server. The FTP Adapter uses the response from the LIST command to retrieve a list of files that must be processed

However, the response to the LIST command differs slightly from one FTP server type to another (see Table 4-21).

To counter the differences in the structure of the response, the FTP Adapter employs `FtpListResponseParser` and `FtpTimestampParser` instances for parsing. The Ftp Adapter ships with pre-built implementations for the `FtpListResponseParser` and `FtpTimestampParser` for different FTP server types.

You can configure these implementations in the connection factory for the Web Logic deployment descriptor for the FTP Adapter by setting appropriate values for `listParserKey` and `timeParserKey` respectively.

See Table 4-21to understand the different responses to the LIST command from different FTP server types.

Table 4-21 Responses to the LIST Command from Different FTP Server Types

FTP Server Type	Results from LIST Command
UNIX	-rwxr-xr-x 2 root root 1024 Apr 1 14:12 test.txt
Windows	09-13-11 12:10 23 test.txt
MVS	SAVE00 3390 2004/06/23 1 1 FB 128 6144 PS IN.TXT

Extending MLSD

`MLSD` is a new FTP command that provides standardized directory listings regardless of the type of the FTP server.

You can extend the FTP Adapter to support the `MLSD` command.

Note that the response from the `MLSD` command differs from using the `LIST` command as the format of the `MLSD` response is defined by the examples shown:

```
type=file;modify=20110101010101;size=1024; address-csv.txt
type=dir;modify=20110101010101; my_files
type=file;size=1023;modify=20112011011001; index.htm
```

Currently `MLSD` is supported by some, but not all, FTP servers.

Configuring Plugin Implementations to Support the MLSD Command

You can use plugin implementations of `FtpListResponseParser` and `FtpTimestampParser` (see FtpTimestampParser Interface) to support MLSD command. To do so.

1. Create `fileftp.jar`. This is a one-time activity and required for compilation only. Refer to Using a Manifest.MF file to Generate a fileftp.jar for Compilation for information on creating `fileftp.jar`.

2. Add `fileftp.jar` to classpath, for example,`set CLASSPATH= $ORACLE_SOA_HOME/soa/modules/oracle.soa.adapter_11.1.1/ fileftp.jar` (jar from Step 1)

3. Create `MLSDListResponseParserImpl.java`. See FtpListResponseParser Interface for information on implementing a parser to parse a response from the MLSD FTP command.

4. Create `MLSDTimestampParserImpl.java`. See FtpTimestampParser Interface for information on implementing a parser to parse last modified time from MLSD response

5. Compile the two classes and package them as a jar, for example, `my_ftpextension.jar`. If there are any compilation failures, make sure that `fileftp.jar` is in classpath.

```
javac -d . MLSDListResponseParserImpl.java
javac -d . MLSDTimestampParserImpl.java
jar cvf my_ftpextension.jar <<compile folder>>
```

6. Copy `my_ftpextension.jar` under `$FMW_HOME/soa/soa/modules/ oracle.adapter.ext_12.1.2`

```
cp my_ftpextension.jar $FMW_HOME/soa/soa/modules/oracle.adapter.ext_12.1.2
```

7. Run `ant` to add the `myftpclient.jar` to the Adapter classpath.

8. Set the Connection factory entry `listParserKey ="com.acme.MLSDListResponseParserImp"` to the WebLogic deployment descriptor for the FTP Adapter to point to the new parser.

9. Add Connection factory entry timeParserKey `="com.acme. MLSDTimestampParserImpl"` to the WebLogic deployment descriptor for the FTP Adapter.

10. Restart the server

The implementation of the `FtpTimestampParser` exposes two different date-time-formats - `defaultDateFormat` and `recentDateFormat`, because most FTP servers show the modified time for recently created files in a slightly different format compared to older files.

For example, on UNIX systems older files are shown as `Aug 29 2011`, but newer files are shown as `Mar 12 10:00` for files created on March 12th 2014.

When using MLSD, you must set the `defaultDateFormat` to `yyyyMMddhhmmss` and `recentDateFormat` to `yyyyMMddhhmmss.SSS` in the connection factory in `weblogic-ra.xml`

Extend the Listing Operation to Send MLSD Commands Rather than the LIST Commands

The FTP Adapter exposes the following operations. Each of these operations map to corresponding FTP commands.

Table 4-22 FTP Adapter Mapping to FTP Commands

FTP Adapter Operation	Description	FTP Operation
Listing	Returns a list of files from the server.	LIST/NLST/MLSD
Store	Stores a single file on the server as an outbound operation.	STOR
Retrieve	Retrieves a single file on the server	RETR
Delete	Deletes a single file on the server	DELE

You can extend these operations (Listing, Store, Delete, Retrieve) as you can configure the FTP Adapter scenario to send `MLSD` instead of `LIST` for Listing operation, `STOU` (FTP command for Store Unique) rather than STOR. You can similarly alter other commands.In this case, you extend the Listing operation to support `MLSD,` in addition to `LIST/NLST` commands which it already supports.

Earlier, you configured the FTP Adapter to parse responses coming from MLSD. This section provides information on ensuring that the FTP Adapter is going to send the MLSD command rather than the LIST/NLST command that it sends otherwise.To send the MLSD command, you create a mapping file (for example, `my_mapping.xml`) that corresponds to `ftpmapping.xsd` (refer to ftpmapping Schema) and which overrides the Listing operation. For example,

```
<ftpmapping xmlns="http://schemas.oracle.com
         /ftpadapter/mapping">
    <listing command="MLSD="${default}"
           success="$ftp.code = 150 or $ftp.code=125
                 or $ftp.code=226">
    </listing>
</ftpmapping>
```

In this example, the listing operation uses the `MLSD` FTP command rather than the vanilla `LIST` command so the listing is sorted at the FTP server.

You can use the `success` attribute to specify the success/error condition. If the control channel returns 125, 150 or 226, it is considered successful by the FTP Adapter, otherwise it results in an Adapter exception that propagates to the binding component pipeline that is making the inbound or outbound listing operation.

The mapping file MLSD argument can have these values:

- `${default}` is replaced by the original directory path sent as a result of the request. Hence, if you have configured your composite with `/home/foo1` as your `PhysicalDirectory` in the outbound jca and used `/home/foo2` as the header value for the `jca.ftp.Directory` parameter in the outbound header, this parameter is replaced with the `/home/foo2`.

  In most cases, you do not change the value of argument from `${default}` to anything else

- A hard-coded value, for example, `argument="/home/bar"`in the mapping file. If you use this, the hard-coded value is used. This overrides any jca parameter/ header you might have configured in the composite configuration.

Once you have created the mapping file, you must copy it to the composite application. Add the mapping file to the same directory as the JCA file of your composite application. You must configure the mapping file as a part of JCA activation or interaction, for example,

Example - Configuring the Mapping as a Part of JCA Activation or Interaction

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FTPIn" adapter="Ftp Adapter" …>
   <connection-factory location="eis/Ftp/FtpAdapter" …/>
  <endpoint-activation operation="Get">
    <activation-spec className="oracle.tip.adapter.ftp.inbound.
         FTPActivationSpec">
             <[..]>
             <property name="FtpMappingFile"
                     value="my_mapping.xml"/>
            <[..]>
    </activation-spec>
  </endpoint-activation>
```

Extend the Store Operation to Send Additional Proprietary FTP Commands to FTP Server Running on the MVS Platform

Disk space (`PRIMARY/SECONDARY`), Logical Record Length (`LRECL`), Block Size (BLKSIZE) are characteristics of a dataset on MVS systems. FTP servers running on MVS systems are aware of these configurations.

Though each of these configurations have default values, there are times where you would want to override the defaults depending on the dataset that you wish to download/upload from the MVS system.

For example, you might want to upload a large dataset to the FTP Server of 50 MB. The requirements is then that additional space must be allocated before the STOR operation executes. You must configure BLKSIZE of, for example, 6192 with LRECL of 86. If these parameters are not set, the FTP transfer (STOR operation) fails.

Fortunately, for an FTP server, there is a way to send proprietary commands by using `QUOTE SITE` followed by the command to be sent.

Note that the return code from `QUOTE SITE` might differ from one proprietary command to another proprietary command. In that case, the following commands are sent prior to STOR command.

```
QUOTE<SP>SITE<SP>BLKSIZE=6192<CRLF>
QUOTE<SP>SITE<SP>LRECL=86<CRLF>
STOR<SP>SM.KAR.SARL<CRLF>
```

In this case, you extended the `Store` operation to send these additional `QUOTE SITE` commands to the FTP server.

Note that you can only send control commands to the FTP server. The control commands are those commands which do not require the client to read any data from the data channel. For example, commands such as `CWD`, `PWD`, `MKD`, `QUOTE SITE`, `DELE` are control commands.

However, commands such as `RETR`, `STOR`, `STOU`, `LIST`, `NLST`, MLSD are data commands.

When the data commands are issued, the client must read the response from a data channel. For control commands, however, the response is sent back on the control channel itself.To send a control command such as `STOR` to the FTP server, you must

write a mapping file (for example `my_mapping.xml`) corresponding to `ftpmapping.xsd` (refer to ftpmapping Schema) which overrides the Store operation, as shown in the example:

```
<ftpmapping xmlns="http://schemas.oracle.com/ftpadapter/mapping">
     <store>
     <executecommandsbefore>
        <command value="QUOTE SITE" argument="BLKSIZE=6192"
success="$ftp.code >=   200 and $ftp.code <= 300">
       <command value="QUOTE SITE" argument="LRECL=86"            success="$ftp.code
>= 200 and $ftp.code <= 300">
    </executecommandsbefore>
   </store>
</ftpmapping>
```

In this example, the `executecommandsbefore` statement precedes the `STOR` operation with additional `QUOTE SITE` commands that would be sent.

The same configuration can be expressed as shown in the example below.

Example - Mapping File Configuration

```
<ftpmapping xmlns="http://schemas.oracle.
              com/ftpadapter/mapping">
<globalcommandsconfiguration >
 <global-ftp-command id="BLKSIZE" command="QUOTE SIZE" argument="$
{jca.ftp.BlockSize}"
         success="$ftp.code >= 200 and $ftp.code <= 300"/>
    <global-ftp-command id="LRECL" command ="QUOTE SIZE" argument="$
{jca.ftp.LogicalRecordSize"
         success="$ftp.code >l= 200 and $ftp.code <= 300"/>
</globalcommandsconfiguration>
<store>
<executecommandsbefore>
<ftp-command commandref="BLKSIZE" >
<ftp-command commandref="LRECL" >
</executecommandsbefore>
</store>
</ftpmapping>
```

In the example, you have configured the commands globally and you are referencing them using the `commandref` attribute. Note that both the approaches of expressing the mapping metadata are identical and you can choose one based on your requirements.

`jca.ftp.BlockSize` and `jca.ftp.RecordSize` can be passed as normalized message properties, for example headers in the BPEL `<invoke>` activity, for example

Example - Passing BlockSize and RecordSize as Normalized Message Properties

```
    <assign name="AssignFtpMetadata">
      <copy>
        <from expression="'BLKSIZE=6192'"/>
        <to variable="blockSize"/>
      </copy>
      <copy>
        <from expression="LRECL=86"/>
        <to variable="logicalRecordSize"/>
      </copy>
    <invoke name="WriteWeatherData"
                 bpelx:invokeAsDetail="no"
          inputVariable=
```

```
                "WriteWeatherData_Put_InputVariable"
           partnerLink="XmlOut"
                portType="ns2:Put_ptt" operation="Put">
           <bpelx:inputProperty name="jca.ftp.BlockSize"
                variable="blockSize"/>
             <bpelx:inputProperty name=
                    "jca.ftp.LogicalRecordSize"
                     variable="logicalRecordSize"/>
      </invoke>
```

You must copy the mapping file to the same directory as your .jca file for interaction spec. This mapping file, in turn, must be configured as a part of the JCA interaction spec as shown in the example below.

Example - Configuring the Mapping File as Part of the JCA Interaction Spec

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FTPOut" adapter="Ftp Adapter" …>
   <connection-factory location="eis/Ftp/FtpAdapter" …/>
  <endpoint- interaction operation="Get">
    <interaction-spec
       className="oracle.tip.adapter.
            ftp.outbound.FTPInteractionSpec">
            <[..]>
             <property name="FtpMappingFile"
               value="my_mapping.xml"/>
            <[..]>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

Additional Configuration Parameters, Implementations, Interfaces and Schema

Following is a list of additional configuration parameters that you can use when extending the FTP Adapter.

- **FtpMappingFile**

  You can provide this in the jca file for either the inbound or the outbound interaction. It stores the path to the mapping file relative to the composite application. An example of the ftpmapping schema is provided in ftpmapping Schema.

  Use the same mapping file for both the inbound and outbound operation.

  As an example, `<property name="FtpMappingFile" value="xsd/ftp_mapping.xml"/>` means the mapping file resides in the xsd folder of the composite application.

- **ftpClientClass**

  You can provide this as a connection factory parameter in the deployment descriptor for the FTP Adapter. If you want to provide your own FTPClient implementation, this is where you place the fully qualified classname for the implementation.

- **listParserKey**

  You can provide this as a connection factory parameter in the deployment descriptor for the FTP Adapter. If want to provide your own `FtpListResponseParser` implementation, this is where you place the fully qualified classname for the implementation.

- **timeParserKey**

  You can provide this as a connection factory parameter in the deployment descriptor for FTP Adapter.

  If you want to provide your own `FtpTimestampParser` implementation, this is where you place the fully qualified classname for the implementation.

Sample FTPClient Implementation

See below for the sample FTPClient Implementation.

Example - Sample FTP Client Implementation

```
import oracle.tip.adapter.file.LoggerUtil;
import oracle.tip.adapter.file.FileResourceException;
import oracle.tip.adapter.ftp.IFtpDescriptor;
import oracle.tip.adapter.ftp.FTPReply;
import oracle.tip.adapter.ftp.FTPClient;
import oracle.tip.adapter.ftp.FTPCommand;
import oracle.tip.adapter.ftp.IFTPClient;

import javax.resource.ResourceException;
import java.io.IOException;
import oracle.tip.pc.infra.exception.
                        PCExceptionIndex;
import javax.resource.spi.security.
                        PasswordCredential;
import java.net.Socket;
import java.util.logging.Logger;
import java.util.logging.Level;

public class TestFTPClient extends FTPClient
                      implements
                        IFTPClient {

    private Logger _logger = Logger.getLogger
                   (TestFTPClient.class.getName());
    public TestFTPClient() {
        super();
Sample FTPClient Implementation

    }
    public void initialize(IFtpDescriptor ftpDescriptor) {
        super.initialize(ftpDescriptor);
    }

    public Logger getLogger() {
       return _logger;
    }

    public boolean login(Socket controlSocket,
                 PasswordCredential pc)
                          throws IOException,
                       ResourceException
    {
        intrc = 0;
        String replyStr = null;
        logDebug("TestFTPClient::login(...) invoked ");

        replyStr = user(controlSocket,
                    pc.getUserName());
```

```
        rc = getReplyCode(replyStr,
                    m_ftpDesc.getHost());

        logDebug("TestFTPClient::
                     command[USER] =>" + rc + "<=");

        if(FTPReply.isPositiveCompletion(rc)) {
            return true;
        }
        if(!FTPReply.isPositiveIntermediate(rc)) {
            // USER failed
            logError("Unable to login
                     to server '"
                         + m_ftpDesc.getHost() + "'; " +
                   "FTP command USER returned
                         unexpected reply code : " + rc);

            FileResourceException frex = new
FileResourceException(PCExceptionIndex.ERROR_LOGIN);
            frex.setEISErrorCode(String.valueOf(rc));
            frex.setEISErrorMessage(replyStr);
            throw frex;
        }
        replyStr = pass(controlSocket,
                         new String(pc.getPassword()));
        rc = getReplyCode(replyStr, m_ftpDesc.getHost());
        logDebug("TestFTPClient::
                         command[PASS] =>" + rc + "<=");
        if ( FTPReply.isPositiveCompletion(rc)) {
            return true;
        }
        if ( !FTPReply.isPositiveIntermediate(rc)) {
            // PASS failed
            logError("Unable to login to server '" +
                         m_ftpDesc.getHost() + "'; " +
                    "FTP command PASS returned
                                 unexpected
                           reply code : " + rc );

            FileResourceException frex = new
FileResourceException(PCExceptionIndex.ERROR_LOGIN);
            frex.setEISErrorCode(String.valueOf(rc));
            frex.setEISErrorMessage(replyStr);
            throw frex;
        }
        //Check ACCT account
        String accountName = "hard-coded";
        replyStr = sendCommand(controlSocket,
                             FTPCommand.ACCT,
                                      accountName);
        rc = getReplyCode(replyStr,
                       m_ftpDesc.getHost());
        logDebug("TestFTPClient::
                      command[ACCT] =>" + rc + "<=");
        if ( !FTPReply.isPositiveCompletion(rc)) {
            // ACCT failed
            logError("Unable to login to
                      server '" + m_ftpDesc.getHost() + "'; " +
                    "FTP command ACCT returned unexpected
                      reply code : " + rc );
```

```
            FileResourceException frex = new
FileResourceException(PCExceptionIndex.ERROR_LOGIN);
            frex.setEISErrorCode(String.valueOf(rc));
            frex.setEISErrorMessage(replyStr);
            throw frex;
        }
        return true;


    }

    private void logDebug(String logData){
         _logger.log(Level.FINE, logData);
    }

    private void logError(String logData){
        _logger.log(Level.SEVERE, logData);
    }
}
```

FtpListResponseParser Interface

The following is the `FTPListResponseParser` Interface, which defines the interface for parsing FTP file listings and converting that information into `FileInfo` instances.

Example - Sample FTPListResponseParser

```
package oracle.tip.adapter.ftp.parsers;
import …

/**
 * FtpListResponseParser defines the interface for
 * parsing FTP file listings
 * and converting that information into FileInfo instances.
 * It also uses a default
 * or a user supplied Timestamp parser instance to parse the
 * file modified time
 */
public interface FtpListResponseParser
{


  /**
   * This parameter can be used
   * to throttle the Ftp Adapter to
   * return the required
   * number of files from the call based on heuristics
   * such as memory available
   * @param pageSize The number of FileInfo
   * instances to be returned
   */
  public void setPageSize(int pageSize);

  /**
   * @return Returns the page size
   */
  public int getPageSize();

  /**
   * This method is used to configure
   * the parser instance
   * being returned.
```

```
 * @param ftpDescriptor
 */
public void configure(IFtpDescriptor ftpDescriptor);

/**
 * Parses an Ftp listing and returns an array
 * of FileInfo
 * @param directory The Ftp directory being listed
 * @param stream The data socket stream to read from
 * @param encoding Server encoding
 * @return The list of FileInfo instances
 * @exception Exception  Any exception in reading
 * the data socket stream.
 */

public FileInfo[] parseListResponse(String directory,
         InputStream stream, String encoding)
                 throws IOException;

/**
 * Parses an Ftp listing and returns an
 * array of FileInfo
 * using the default encoding
 * @param directory The Ftp directory being listed
 * @param stream The data socket stream to read from
 * @return The list of FileInfo instances
 * @exception Exception  Any exception in reading
 * the data socket stream.
 */
public FileInfo[] parseListResponse(String directory,
          InputStream stream) throws IOException;

/**
 * Returns a single FileInfo instance from a single line
 * @param line The line read from the data socket
 */
public FileInfo parseLine(String line);

/**
 * Reads and returns a single line from
 *  the data socket
 * @param reader The reader constructed
 * from the data socket stream
 * @return The line read from the socket stream
 * @exception Exception  Any exception in
 * reading the data socket stream.
 */
public String nextLine(BufferedReader reader)
          throws IOException;

/**
 * Preprocessing hook before the list
 * is returned to the client
 * @param originalList List of lines read
 */
public List preProcess(List originalList);

/**
 * Set the parser for parsing the File timestamps
 * returned from the call to listing
 * @param ftpTimestampParser The Timestamp
```

```
   * parser interface
   */
  public void setTimestampParser(FtpTimestampParser
                  ftpTimestampParser);

  /**
   * Parses the timestamp from listing
   * @exception Exception  Any exception
   * in parsing the time
   */
  public long parseTimestamp(String timestamp)
      throws Exception;
}
```

FtpTimestampParser Interface

`FTPTimestampParser` defines the interface for parsing timestamps from files on an FTP Server. See the example below.

Example - Sample FtpTimestampParser Interface

```
package oracle.tip.adapter.ftp.parsers;

import …;
/**
 * FtpTimestampParser defines
 * the interface for parsing timestamps from
 * files on Ftp Server.
 *
 */

public interface FtpTimestampParser
{
  /**
   * This parameter can be used to configure
   * the timestamp parser implementation
   * @param ftpDescriptor The IFtpDescriptor
   * instance used
   *to configure the timestamp parser
   */
    public void setFtpDescriptor(
                IFtpDescriptor ftpDescriptor);
  /**
   * Returns the timestamp from the file
   * time in listing format
   * @param timestamp File time from the File listing
   * @return timestamp in long
   * @throws Exception
   */
    public long parseTimestamp(String timestamp)
                throws Exception;
}
}
```

ftpmapping Schema

The FTP mapping schema, `ftpmapping`, is shown in the example below.

Example - FTPMapping Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://schemas.oracle.
```

```
                        com/ftpadapter/mapping"
            xmlns:tns="http://schemas.oracle.com
                        /ftpadapter/mapping"
            targetNamespace="http://schemas.oracle.com/
                        ftpadapter/mapping"
            elementFormDefault="qualified"
                    attributeFormDefault=
                            "unqualified">

  <xsd:element name="ftpmapping">
    <xsd:complexType>
      <xsd:sequence>
       <xsd:element ref="globalcommandsconfiguration"
                      minOccurs="0" />
        <xsd:element name="listing" type="tns:ftp-operation-type"
                      minOccurs="0" />
        <xsd:element name="retrieve" type="tns:ftp-operation-type"
                      minOccurs="0"/>
        <xsd:element name="store" type="tns:ftp-operation-type"
                     minOccurs="0"/>
        <xsd:element name="delete"
                    type="tns:ftp-operation-type"
                      minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
       </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>

  <xsd:complexType name="ftp-operation-type">
    <xsd:sequence>
      <xsd:element ref="executecommandsbefore"
                  minOccurs="0"/>
      <xsd:element ref="executecommandsafter"
                   minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="command" type="xsd:string"
             use="optional" />
    <xsd:attribute name="argument"
                              type="xsd:string"
             use="optional" />
    <xsd:attribute name="success" type="xsd:string"
             use="optional" />
    <xsd:attribute name="desc" type="xsd:string"
             use="optional" />
    <xsd:attribute name="commandref"
                  type="xsd:string"
             use="optional" />
  </xsd:complexType>

  <xsd:element name="executecommandsafter">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="ftp-command"
               minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

```
<xsd:element name="executecommandsbefore">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="ftp-command"
               minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<xsd:element name="ftp-command">
   <xsd:complexType>
     <xsd:attribute name="argument"
                     type="xsd:string"
                  use="optional" />
     <xsd:attribute name="success"
                   type="xsd:string"
                  use="optional" />
     <xsd:attribute name="desc"
                   type="xsd:string"
                  use="optional" />
     <xsd:attribute name="command"
                       type="xsd:string"
                  use="optional" />
     <xsd:attribute name="commandref"
                         type="xsd:string"
                  use="optional" />
   </xsd:complexType>
 </xsd:element>

 <xsd:element name="global-ftp-command">
   <xsd:complexType>
     <xsd:attribute name="argument"
          type="xsd:string"
                  use="optional" />
     <xsd:attribute name="success" type="xsd:string"
                  use="required" />
     <xsd:attribute name="desc" type="xsd:string"
                  use="optional" />
     <xsd:attribute name="id" type="xsd:string"
                  use="required" />
     <xsd:attribute name="command" type="xsd:string"
                  use="required" />
   </xsd:complexType>
 </xsd:element>

 <xsd:element name="globalcommandsconfiguration">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="global-ftp-command"
                      minOccurs="0"
                         maxOccurs="unbounded" />
```

Using a Manifest.MF file to Generate a fileftp.jar for Compilation

Create a MANIFEST.MF file with the contents as shown.Be sure to include the Class-path directive. Ensure that each entry in the Class-path directive is separated by a blank space.

Manifest-Version: 1.0

Ant-Version: Apache Ant 1.7.1

Created-By: 20.4-b02 (Sun Microsystems Inc.)

Implementation-Vendor: Oracle

Implementation-Title: Ftp Adapter Extensibility

Implementation-Version: 11.1.1

Product-Name: Ftp Adapter Extensibility Manifest

Product-Version: 12.1.2.0.0

Specification-Version: 11.1.1

Class-Path: `jca-binding-api.jar fileAdapter.jar ftpAdapter.jar ../../../../wlserver/modules/javax.resource_1.6.1.jar ../oracle.soa.fabric_11.1.1/bpm-infra.jar`

To include this `MANIFEST.MF`, do the following:

1. Extract `fileAdapter.jar` and `ftpAdapter.jar` from

   `$fmwhome/soa/soa/connectors/FileAdapter.rar` and `$fmwhome/soa/soa/connectors/FTPAdapter.rar` respectively

2. Copy these files under `$fmwhome/soa/soa/modules/oracle.soa.adapter_11.1.1`.

3. Copy the created `MANIFEST.MF` to the same folder

   ```
   jar cmf fileftp.jar MANIFEST.MF
   ```

4. Use the generated `fileftp.jar` for compilation.

Sample ListParser and TimeParser

The following example shows a sample ListParser and TimeParser.

Example - Sample ListParser and TimeParser

```
package oracle.tip.adapter.ftp.test;

import java.util.regex.Pattern;
import java.util.Map;
import java.util.HashMap;
import java.util.StringTokenizer;

import oracle.tip.adapter.file.FileInfo;
import oracle.tip.adapter.file.FileLogger;
import oracle.tip.adapter.file.LoggerUtil;
import oracle.tip.adapter.ftp.IFtpDescriptor;
import oracle.tip.adapter.ftp.parsers.
                     FtpTimestampParser;
import oracle.tip.adapter.ftp.parsers.
                     FtpListResponseParserImpl;
/**
 * Implementation of FtpListResponseParser
                      for MLSD responses
 * Responses are return as
         - type=file;modify=20110101010101;size=1024;
                               filename
 */
```

```
public class MLSDListResponseParserImpl
  extends FtpListResponseParserImpl
{
  private static final String TYPE = "type";
  private static final String SIZE = "size";
  private static final String MODIFY = "modify";

  public MLSDListResponseParserImpl()
  {
  }
/**
   * Returns a FileInfo instance
   * for a single line.
   * @param line Line from list response
   * @return FileInfo instance
   */
  public FileInfo parseLine(String line)
       throws Exception
  {

    System.out.println
               ("MLSDListResponseParserImpl::parseLine
                called [" + line + "]");
    FileInfo fileInfo = new FileInfo();
    fileInfo.setRaw(line);

    //extract the file name

    String response[] = line.split(" ");
    if(response == null || response.length != 2){
       throw new Exception("Invalid response
                   from ftp server [" + line + "]");
    }

    fileInfo.setFileName(response[1]);
    StringTokenizer st = new
                  StringTokenizer(response[0], ";");
    Map<String, String> properties =
                  new HashMap<String, String>();
    String token = null;
    while(st.hasMoreElements()){
       token = st.nextToken().trim();
       int index = token.indexOf('=');
       if(index == -1){
           throw new Exception
          ("Invalid ftp line since token
                           [" + token + "]
                  does not contain '='");
       }
       String key = token.substring(0, index).trim();
       String value = token.substring(index+ 1,
                  token.length()).trim();
       if(key.length() ==0 || value.length() ==0){
           throw new Exception("Invalid ftp line since
             either key[" + key + "] or value [" + value + "]
                 is invalid");
       }
       properties.put(key, value);
    }
    String type = properties.get(TYPE);
    String modify = properties.get(MODIFY);
```

```
  String size = properties.get(SIZE);
  if("file".equals(type)){
      fileInfo.setFileType(FileInfo.IS_FILE);
  }
  else if("dir".equals(type)){
      fileInfo.setFileType(FileInfo.IS_DIR);
  }
  else {
      fileInfo.setFileType(FileInfo.IS_UNKNOWN);
  }
  try{
      fileInfo.setSize(Long.parseLong(size));
  }
  catch(NumberFormatException e){
  }

  long modifiedTime = 0;
  try{
      modifiedTime = parseTimestamp(modify);
      fileInfo.setTimestamp(modifiedTime);
  }
  catch(Exception e){
      FileLogger.logWarning
          ("Unable to parse timestamp[" + modify + "]",e);
  }
  System.out.println("FileInfo returned
                [" + fileInfo + "]");
  return fileInfo;
}
```

5

Oracle JCA Adapter for Sockets

This chapter describes how to use Oracle JCA Adapter for Sockets (Oracle Socket Adapter), which works with Oracle BPEL Process Manager (Oracle BPEL PM) and Oracle Mediator (Mediator) as an external service.

This chapter includes the following sections:

Introduction to Oracle Socket Adapter

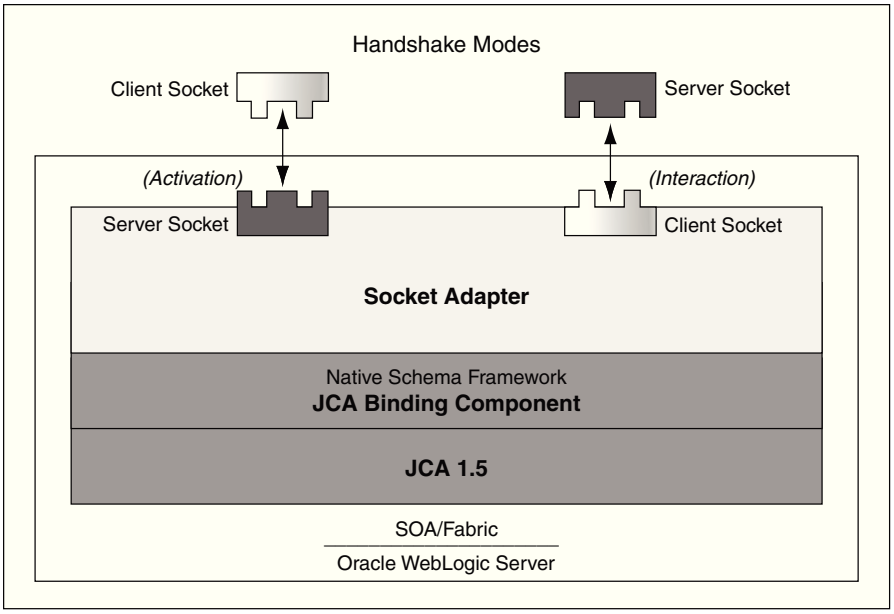
Oracle Socket Adapter is a JCA 1.5 compliant adapter for modeling standard or nonstandard protocols for communication over TCP/IP sockets. You can use an Oracle Socket Adapter to create a client or a server socket, and establish a connection. The data that is transported can be text or binary.

This section includes the following topics:

Oracle Socket Adapter Architecture

Oracle Socket Adapter is based on the JCA 1.5 architecture. JCA provides a standard architecture for integrating heterogeneous enterprise information systems (EIS). The JCA Binding Component of the Oracle Socket Adapter exposes the underlying JCA interactions as services (`WSDL` with JCA binding) for Oracle BPEL PM integration. Figure 5-1 illustrates the architecture of Oracle Socket Adapter. For details about the Oracle JCA Adapter architecture, see Architecture.

Figure 5-1 Oracle Socket Adapter Architecture



Socket Adapter Message Rejection and Resubmission Not Used

Note that the Socket Adapter does not support the message rejection and retry functionality that is applicable to other JCA Adapters.

Usually the client socket application that connects to the server socket performs the error handling when there is an exception from the downstream adapter. Because message rejections in Socket adapter are not supported, thExaltessage re-submission feature is not used in the Socket Adapter.

Oracle Socket Adapter Integration with Mediator

Oracle Socket Adapter is automatically integrated with Mediator. When you create an Oracle Socket Adapter service in JDeveloper Designer, the Adapter Configuration Wizard is started. This wizard enables you to configure the Oracle Socket Adapter. When configuration is complete, a WSDL file of the same name is created in the Application Navigator section of Oracle JDeveloper (JDeveloper). This WSDL file contains the configuration information you specify in the Adapter Configuration Wizard.

The Operation Type page of the Adapter Configuration Wizard prompts you to select an operation to perform. Based on your selection, different Adapter Configuration Wizard pages appear and prompt you for configuration information.

For more information about Oracle JCA Adapter integration with Mediator, see Adapter Integration with Oracle Fusion Middleware.

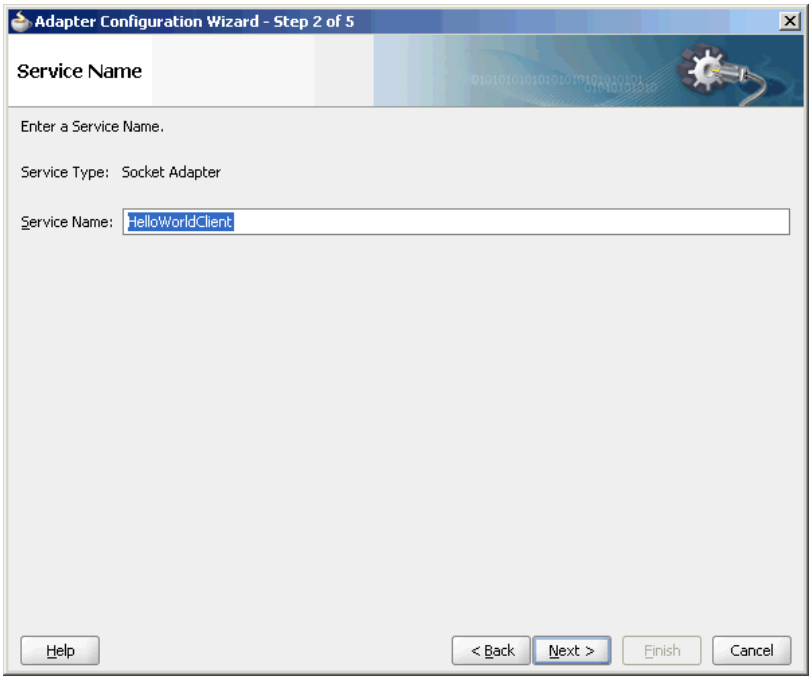
Oracle Socket Adapter Integration with Oracle BPEL PM

Oracle Socket Adapter is automatically integrated with Oracle BPEL PM. When you drag and drop Socket Adapter from the Components window of JDeveloper BPEL Designer, the Adapter Configuration Wizard starts with a Welcome page, as shown in Figure 5-2.

Figure 5-2 The Adapter Configuration Wizard - Welcome Page

This wizard enables you to configure an Oracle Socket Adapter. The Adapter Configuration Wizard then prompts you to enter a service name, as shown in Figure 5-3.

Figure 5-3 The Adapter Configuration Wizard Service Name Page



When configuration is complete, a WSDL file of the same name is created in the Application Navigator section of JDeveloper. This WSDL file contains the configuration information you specify in the Adapter Configuration Wizard.

The Operation Type page of the Adapter Configuration Wizard prompts you to select an operation to perform. Based on your selection, different Adapter Configuration Wizard pages appear and prompt you for configuration information.

For more information about Oracle JCA Adapter integration with Oracle BPEL PM, see Adapter Integration with Oracle Fusion Middleware.

Oracle Socket Adapter Integration with SOA Composite

A composite is an assembly of services, service components (Oracle BPEL PM and Mediator), wires, and references designed and deployed in a single application. The composite processes the information described in the messages. The details of the composite are stored in the `composite.xml` file. For more information on integration of the Oracle Socket Adapter with SOA composite, see Oracle SOA Composite Integration with Adapters.

Oracle Socket Adapter Features

Oracle Socket Adapter enables you to configure a BPEL process or a Mediator service to read and write data over TCP/IP sockets. It includes the following features:

- Allows modeling of standard or nonstandard protocols for communication over TCP/IP sockets
- Supports both inbound and outbound communication
- Allows you to model complex protocol handshakes declaratively, by using XSL
- Allows you the option of plugging in custom Java code to model a protocol handshake
- Provides support for reading and writing native data over sockets as the adapter is integrated with the translator infrastructure (NXSD)
- Supports multiple character encoding

Oracle Socket Adapter Concepts

This section describes the following Oracle Socket Adapter concepts:

- Communication Modes
- Mechanisms for Defining Protocols
- Character Encoding and Byte Order
- Performance Tuning

Communication Modes

Oracle Socket Adapter supports inbound and outbound communication over sockets that can be unidirectional or bidirectional. The communication modes of Oracle Socket Adapter are discussed in the following sections:

- Inbound Synchronous Request/Response
- Outbound Synchronous Request/Response
- Inbound Receive

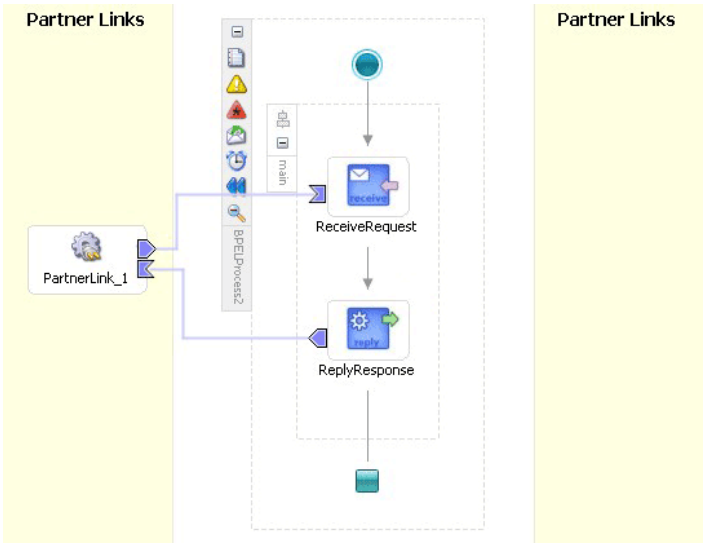
- Outbound Invoke

Inbound Synchronous Request/Response

As part of inbound activation, the Oracle Socket Adapter opens a server socket and waits for incoming connections. The adapter uses the connection to the server socket and reads the request message, which is published to BPEL or Mediator. The Oracle Socket Adapter then uses the same connection to send the response back synchronously.

Figure 5-4 illustrates an inbound synchronous request/response scenario.

Figure 5-4 BPEL Scenario of Inbound Synchronous Request/Response

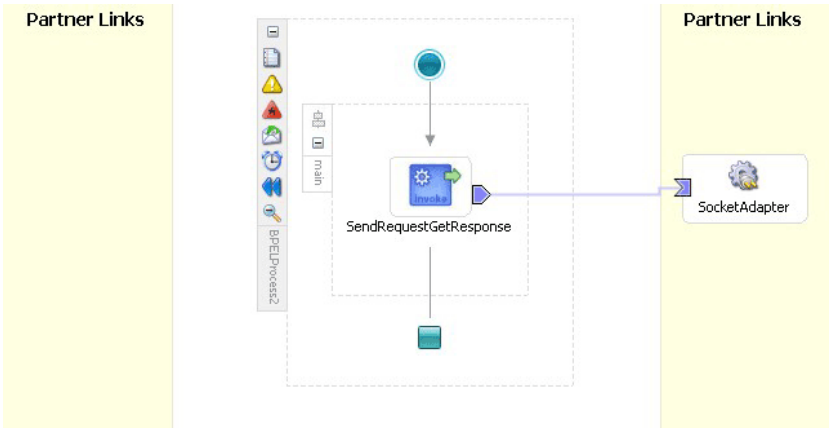


Outbound Synchronous Request/Response

In the case of outbound synchronous request/response, a request comes from BPEL or Mediator. The Oracle Socket Adapter connects to the server socket to send the request message to the server socket on the output stream. The Oracle Socket Adapter then blocks the response from the server socket on the input stream and publishes the response back to BPEL or Mediator.

Figure 5-5 illustrates an outbound synchronous request/response scenario.

Figure 5-5 BPEL Scenario of Outbound Synchronous Request/Response



Inbound Receive

As part of inbound activation, the Oracle Socket Adapter opens a server socket and waits for incoming connections. The adapter uses the connection to the server socket and reads the request message, which is published to BPEL or Mediator. In this scenario, no reply is sent.

Outbound Invoke

In the case of an outbound one way invoke scenario, the request comes from BPEL or Mediator. Oracle Socket Adapter connects to the server socket and sends the request message to the server socket on the output stream without expecting a reply.

Mechanisms for Defining Protocols

Communication protocols or handshakes consist of different discrete steps such as authentication procedures, acknowledgments, and sending or receiving data depending on conditions. Oracle Socket Adapter supports the following mechanisms to define the protocol handshakes.

- Protocol with Handshake Mechanism Using Style Sheet
- Protocol with Handshake Mechanism Using Custom Java Code
- Protocol Without Handshake Mechanism

Protocol with Handshake Mechanism Using Style Sheet

Oracle Socket Adapter can be configured to use a protocol designed with a handshake mechanism, defined using style sheets that use XPath Extension functions exposed by the adapter. This can be granular read and write operation on the socket I/O stream or till the end of the stream. These functions also enable you to use native format constructs for reading and writing data. This handshake mechanism uses XSLT constructs to define operations such as assignments, validations, and control flow.

You can use the XPath Extension functions with the translator infrastructure in the following ways:

- By using StyleReader, which is exposed by the NXSD framework, to read and write from the socket stream using the following methods:

  – `socketRead(nxsdStyle:String, nxsdStyleAttributes:String):String`

    You can use this method to read from the socket input stream.

  – `socketWrite(value:String, nxsdStyle:String, nxsdStyleAttributes:String):String`

    You can use this method to write to the socket output stream.

  The XSLT shown in Figure 5-6 demonstrates the usage of extension functions that use StyleReader.

Figure 5-6 XSLT with Extension Functions That Use StyleReader

```
<?xml version="1.0" encoding="windows-1252" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:ora="http://www.oracle.com/XSL/Transform/java/"
                xmlns:socket="http://www.oracle.com/XSL/Transform
                /java/oracle.tip.adapter.socket.ProtocolTranslator"
                xmlns="http://xmlns.oracle.com/HelloWorld/">
 <!-- Root template -->
 <xsl:template match="/">
  <SynchronousRequestResponseProcessRequest>
   <input>
    <xsl:value-of select="socket:socketRead('terminated', 'terminatedBy=;')"/>
   </input>
  </SynchronousRequestResponseProcessRequest>
  <!-- Copy input value into a temporary variable -->
  <xsl:variable name="temp">
   <xsl:value-of select="input"/>
  </xsl:variable>
  <!-- Concat 'Hello ' to the input string and build the response -->
  <SynchronousRequestResponseProcessResponse>
   <result>
    <xsl:value-of select="concat('Hello ', $temp)"/>
   </result>
  </SynchronousRequestResponseProcessResponse>
 </xsl:template>
</xsl:stylesheet>
```

- By annotating the schema, which defines the input and output variables, using NXSD constructs to read and write from the socket stream using the following methods:

  – `socketReadWithXlation():DocumentFragment`

    You can use this method to read from the socket input stream by using the schema and schema element configured for input.

  – `socketWriteWithXlation(xml:NodeList)`

    You can use this method to write to the socket output stream by using the schema configured for output.

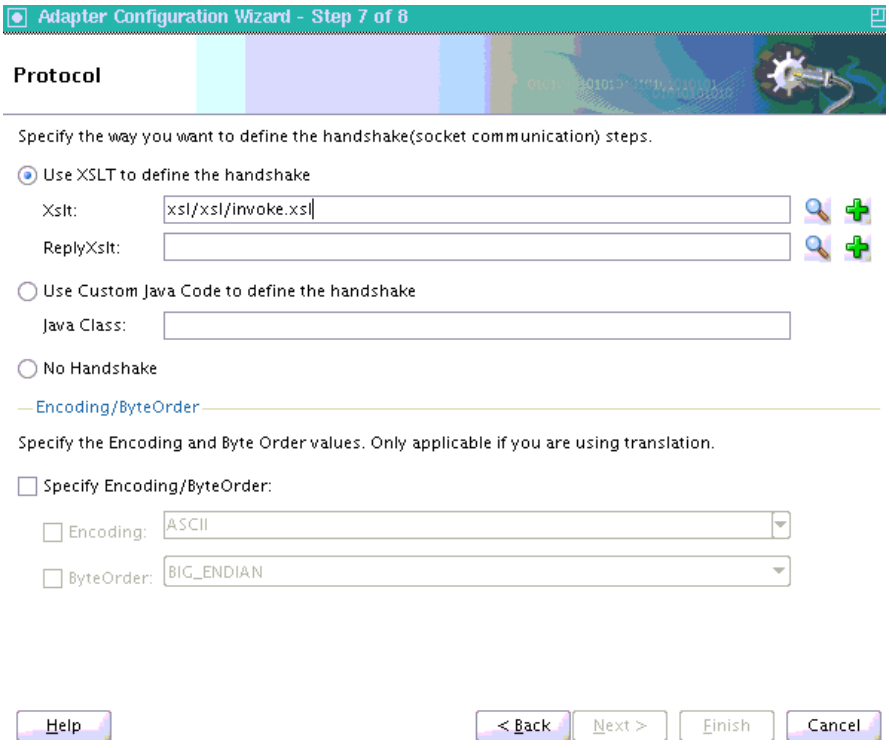
The XSD file shown in Figure 5-7 demonstrates the usage of extension functions by annotating the schema, which defines the input and output variables, using NXSD constructs.

Figure 5-7 XSD with Extension Functions That Do Not Use StyleReader

```
<schema attributeFormDefault="unqualified" elementFormDefault="qualified"
        targetNamespace="http://xmlns.oracle.com/HelloWorldl"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd" nxsd:stream="chars"
        nxsd:version="NXSD">
 <element name="HelloWorldlProcessRequest">
  <complexType>
   <sequence>
    <element name="input" type="string" nxsd:style="fixedLength"
             nxsd:length="15"/>
   </sequence>
  </complexType>
 </element>
 <element name="HelloWorldlProcessResponse">
  <complexType>
   <sequence>
    <element name="result" type="string" nxsd:style="fixedLength"
             nxsd:length="15"/>
   </sequence>
  </complexType>
 </element>
</schema>
```

To define a handshake using style sheet, you must select **Use XSLT to define the handshake** and browse to select the XSL file in the Protocol page, as shown in Figure 5-8.

Figure 5-8 Defining a Protocol with the Handshake Mechanism By Using a Style Sheet



Protocol with Handshake Mechanism Using Custom Java Code

Oracle Socket Adapter can be configured to use a protocol with a customized handshake mechanism, defined by plugging in custom Java code. The custom Java code must implement

`oracle.tip.pc.services.translation.util.ICustomParser`, the ICustomParser interface, provided by Oracle Socket Adapter, which enables custom implementation of handshakes.

Note:

The ICustomParser interface files are in the `bpm-infra.jar` file. This jar file is available in the following directory:

`$SOA_ORACLE_HOME/soa/modules/oracle.soa.fabric_11.1.1`

The following methods must be implemented based on the appropriate communication paradigm:

- `public Element executeOutbound(InputStream in, OutputStream out, Element payLoad) throws Exception;`

  The outbound handshake must implement this method.

  Example:

```
public Element executeOutbound(InputStream in, OutputStream out, Element payLoad)
throws Exception {
    BufferedReader in1 = new BufferedReader(new InputStreamReader(in));
    PrintWriter out1 = new PrintWriter(new OutputStreamWriter(out));

    out1.println(payLoad.getFirstChild().getNodeValue());

    String retVal = in1.readLine();

    StringBuffer strBuf = new StringBuffer();
    strBuf.append("<?xml version='1.0' encoding='" + enc + "' ?>"
            + "<out xmlns='http://xmlns.oracle.com/EchoServer/'>");
        strBuf.append(retVal + "</out>");

    DOMParser parser = new DOMParser();
    parser.setValidationMode(DOMParser.NONVALIDATING);
        Element elem = (Element) parser.getDocument().getElementsByTagName(
            "out").item(0);

    return elem;
}
```

- `public Element executeInboundRequest(InputStream in) throws Exception;`

  The inbound request must implement this method.

  Example:

```
public Element executeInboundRequest(InputStream in) throws Exception {
    BufferedReader in1 = new BufferedReader(new InputStreamReader(in));

    String input = in1.readLine();

    StringBuffer strBuf = new StringBuffer();
    strBuf.append("<?xml version='1.0' encoding='" + enc + "' ?>"
            + "<EchoClientProcessRequest xmlns='http://xmlns.oracle.com/
EchoClient'>");

        strBuf.append("<input>" + input + "</input></EchoClientProcessRequest>");
```

```
    DOMParser parser = new DOMParser();
    parser.setValidationMode(DOMParser.NONVALIDATING);
    parser.parse(new InputSource(new StringReader(strBuf.toString())));
    Element elem = (Element) parser.getDocument().getElementsByTagName(
            "EchoClientProcessRequest").item(0);

    return elem;
}
```

- `public void executeInboundReply(Element payLoad, OutputStream out) throws Exception;`

  The inbound reply must implement this method.

  Example:

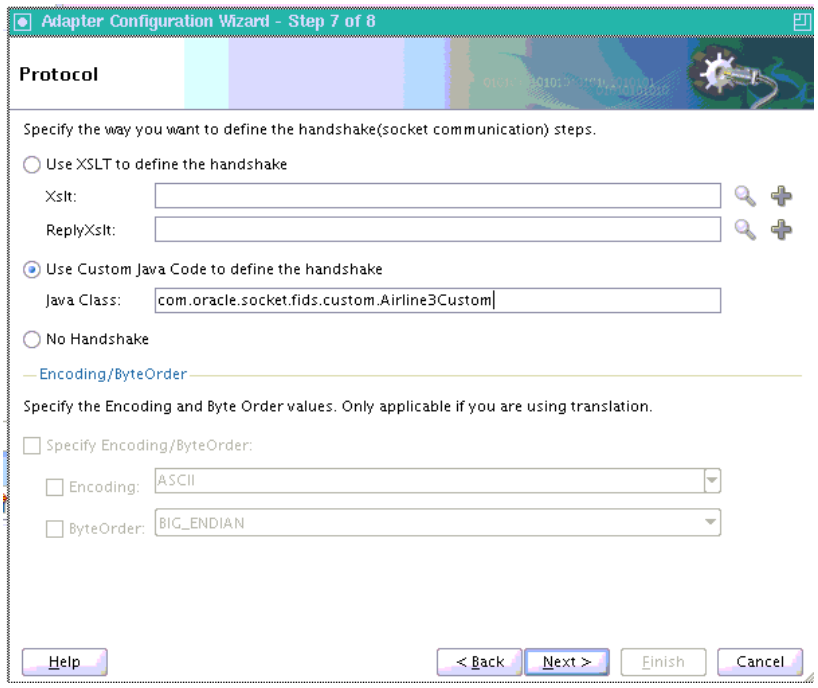
```
public void executeInboundReply(Element payLoad, OutputStream out) throws
Exception {
    PrintWriter out1 = new PrintWriter(new OutputStreamWriter(out));

    NodeList list = payLoad.getChildNodes();
    String retVal = null;
    for(int i = 0; i < list.getLength(); i++) {
        Node node = list.item(i);
        NodeList list1 = node.getChildNodes();
        for(int j = 0; j < list1.getLength(); j++) {
            Node node1 = list1.item(j);
            if(node1.getNodeType() == Node.TEXT_NODE) {
                    retVal = node1.getNodeValue();
            }
        }
    }
    out1.println(retVal);
    out1.flush();
}
```

Note:

`in` is the handle to the socket input stream and `out` is the handle to the socket output stream.

To use a custom Java code to define a handshake, you must select **Use Custom Java Code to define the handshake** and specify the Java class implementing the handshake in the **Java Class** field, as shown in Figure 5-9.

Figure 5-9 Defining a Protocol with Handshake Mechanism By Using Custom Java Code

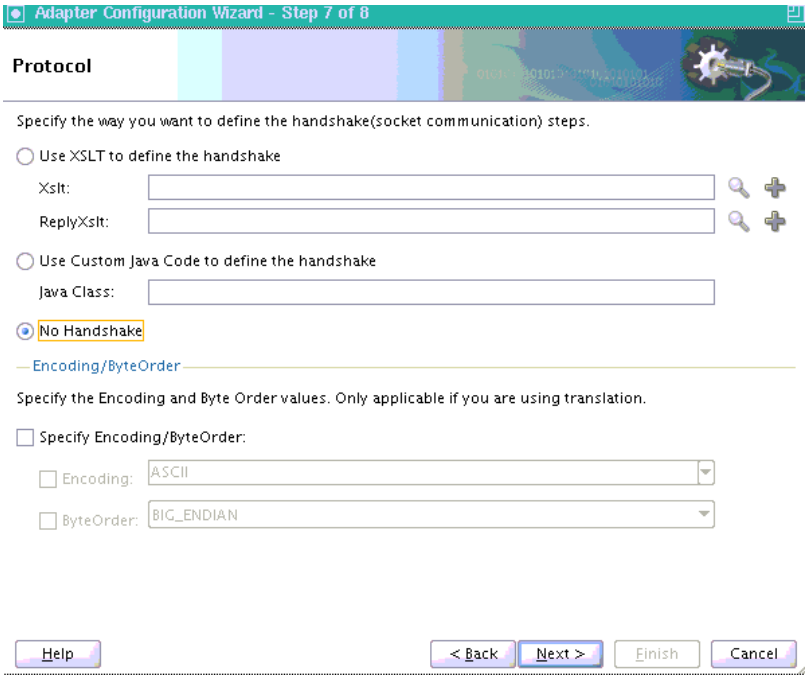


Protocol Without Handshake Mechanism

Oracle Socket Adapter can be configured to use protocols that do not require handshakes involving translation to and from the socket I/O stream.

To use a protocol that does not require a handshake, you must select **No Handshake** in the Protocol page, as shown in Figure 5-10.

Figure 5-10 Defining a Protocol without a Handshake Mechanism



Character Encoding and Byte Order

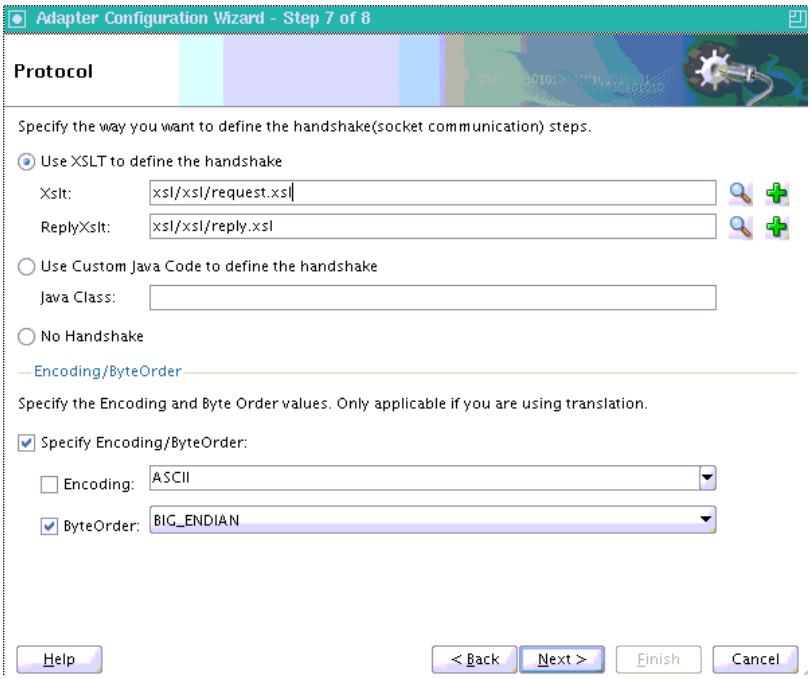
The Encoding property represents the character encoding in which native data is stored, and the ByteOrder property is the byte order of the native data, which is either `BIG_ENDIAN` or `LITTLE_ENDIAN`.

Character encoding and byte order can be specified in the schema file (NXSD), using the Native Format Builder wizard. You can also specify the encoding and the byte order to be used, by using the Adapter Configuration Wizard. When encoding and byte order are not specified, the default values are `US-ASCII` and `BIG_ENDIAN`.

To specify the encoding and byte order values, which are applicable only if you are using translation, you must perform the following steps in the Protocol page of the Adapter Configuration Wizard:

1. In the Encoding/ByteOrder section of the Protocol page, select the **Specify Encoding/Byte Order** option, as shown in Figure 5-11.

Figure 5-11 The Adapter Configuration Wizard - Protocol Page



2. Perform one of these tasks to set the encoding:

   a. To use the encoding specified in the schema file, leave Encoding unchecked.

   b. To specify the encoding using the Adapter Configuration Wizard, select **Encoding**, and then select an encoding type from the Encoding list.

Note:

If you select Encoding, then the encoding type specified using the Adapter Configuration Wizard takes precedence over the encoding type specified in the NXSD file.

---

3. Perform one of these tasks to set the byte order:

a. To use the byte order specified in the schema file, select **Use Byte Order Value from the schema**.

b. To specify the byte order using the Adapter Configuration Wizard, select **ByteOrder**, and then select a byte order from the ByteOrder list.

4. Click **Finish**. Once you click Finish, the Configuration Wizard displays a page that indicates that you have finished configuring the Socket Adapter.

Performance Tuning

The Oracle Socket Adapter supports performance tuning features, including:

- Configuring Connection Pooling

For more information, see Oracle JCA Adapter Tuning Guide and Oracle JCA Adapter Properties.

Configuring Oracle Socket Adapter Connection Pooling

One way to optimize Oracle Socket Adapter performance is by using a Connection Pool. You can use a connection pool while the socket server you are connecting to does not close the socket with each interaction. A connection pool lets you use a socket connection repeatedly, avoiding the overhead of creating a new socket for each interaction. You must configure the connection pool for the Oracle Socket Adapter using the Oracle WebLogic Server console.

Note:

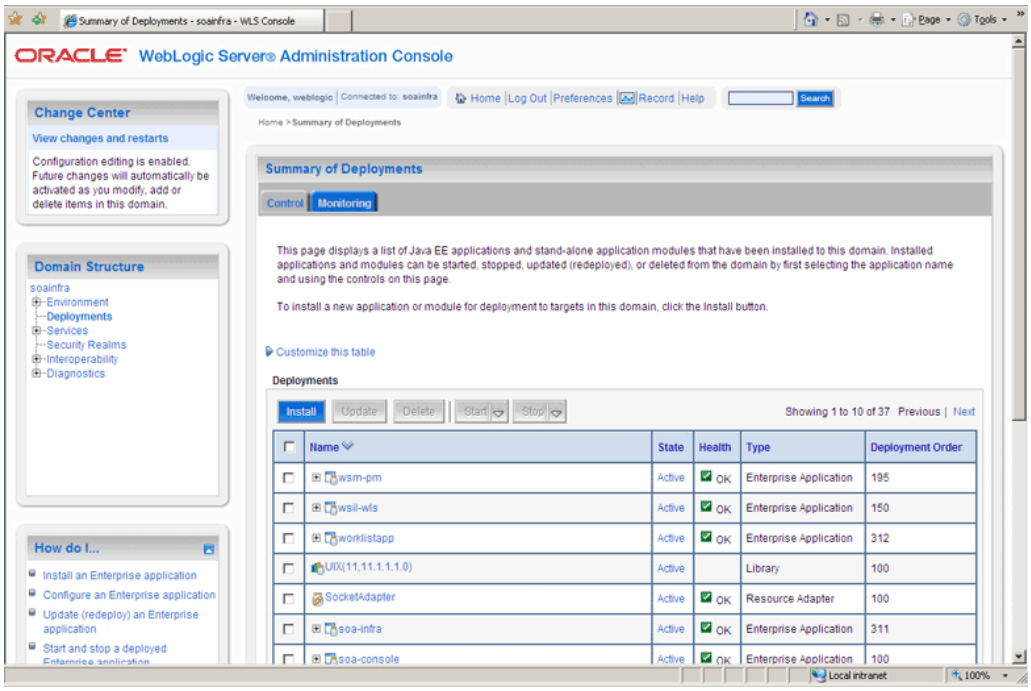
The Connection Pool feature is applicable to outbound interactions only.

How to Configure Oracle Socket Adapter Connection Pooling

1. Log into your Oracle WebLogic Server console. To access the console navigate to `http://servername:portnumber/console`.

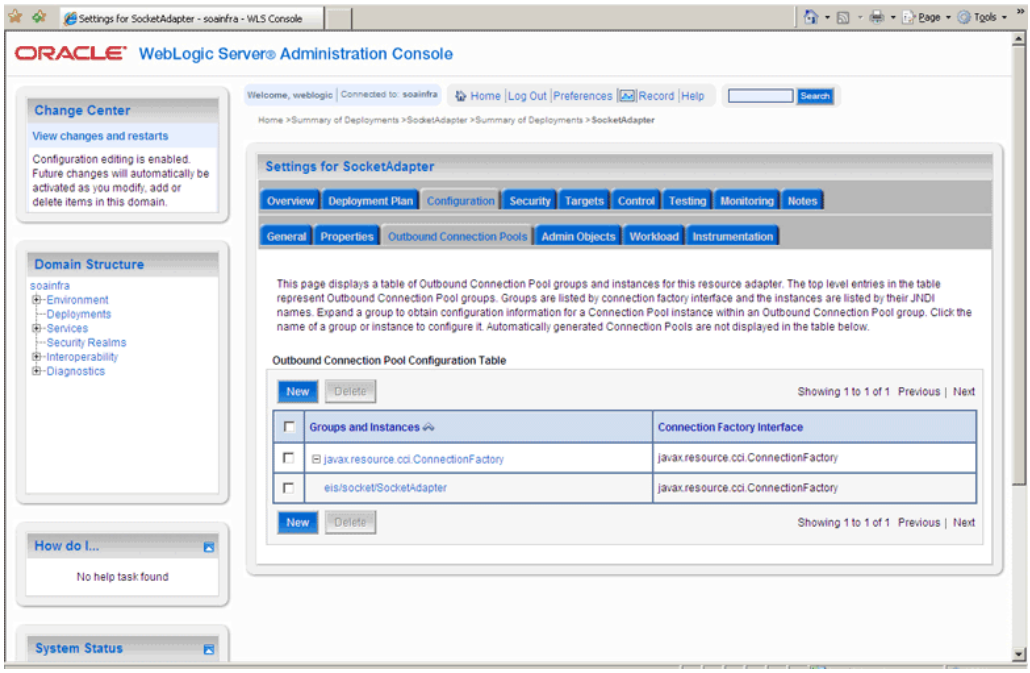
2. Click **Deployments** in the left pane for Domain Structure. The Summary of Deployments page is displayed.

Figure 5-12 Oracle WebLogic Server Console - Summary of Deployments Page



3. Click **SocketAdapter**. The Settings for SocketAdapter page is displayed.
4. Click the **Configuration** tab.
5. Click the **Outbound Connection Pools** tab, and expand **javax.resource.cci.ConnectionFactory** to see the configured connection factories, as shown in Figure 5-13:

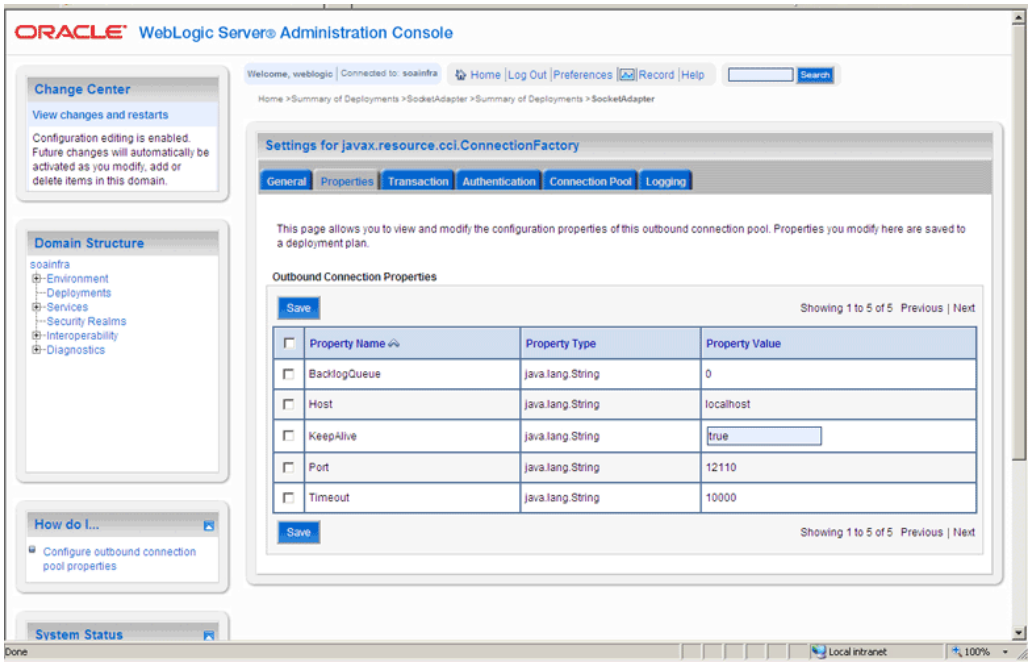
Figure 5-13 Oracle WebLogic Server Console - Settings for SocketAdapter Page



6. Click **eis/socket/SocketAdapter**. The Settings for javax.resource.cci.ConnectionFactory page is displayed.

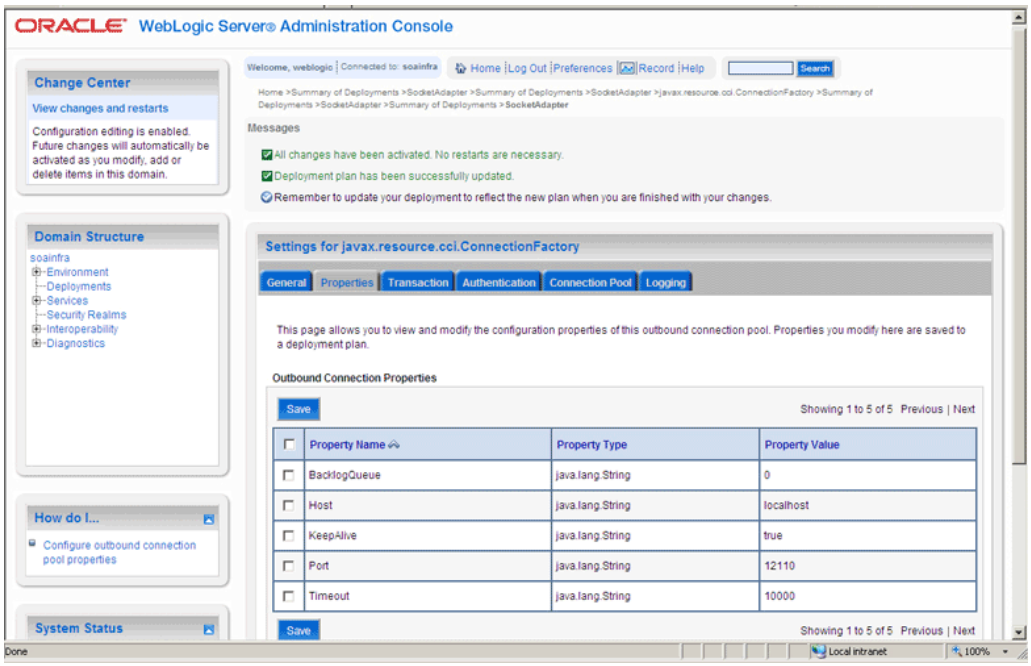
7. Set the `KeepAlive` connection factory property to `true`, as shown in Figure 5-14. The connection pool feature for the Oracle Socket Adapter is enabled.

Figure 5-14 Oracle WebLogic Server Console - Settings for javax.resource.cci.Connectionfactory Page



8. Click **Save**. The Settings for javax.resource.cci.ConnectionFactory page is displayed with the message, `Deployment plan has been successfully updated`, as shown in Figure 5-15.

Figure 5-15 Oracle WebLogic Server Console - Settings for javax.resource.cci.Connectionfactory Page



Note:

You can modify connection pool parameters by using the **Connection Pool** tab of Oracle WebLogic Server Administration Console.

Configuring Oracle Socket Adapter

The following tasks are required for configuring Oracle Socket Adapter:

- Modifying the weblogic-ra.xml File
- Modeling a Handshake
- Designing an XSL File Using the XSL Mapper Tool
- Specifying a TCP Port in a Configuration Plan For an Oracle Socket Adapter

Modifying the weblogic-ra.xml File

To configure Oracle Socket Adapter, you must specify the value of the properties listed in Table 5-1 in the `weblogic-ra.xml` file. You can update these properties from the Oracle WebLogic Server Administration Console. For more information, see Adding an Adapter Connection Factory.

Table 5-1 Oracle Socket Adapter Configuration Properties

Property	Description
Host	In case of outbound interaction, the system name on which the socket server is running, to which you want to connect. In case of inbound interaction, it is always `localhost`.

Table 5-1 (Cont.) Oracle Socket Adapter Configuration Properties

Property	Description
Port	In case of outbound interaction, it is the port number on which a socket server is running, to which an adapter connects. In case of inbound interaction, it is the port number on which the socket adapter listens for incoming connections.
Timeout	With this value set to a nonzero timeout interval, a `read()` call on the InputStream associated with this socket blocks for only this amount of time. If the timeout interval expires, then a `java.net.SocketTimeoutException` is raised though the socket is still valid. The option must be enabled before entering the blocking operation to have effect. The timeout interval must be greater than `0`. A timeout interval of `0` is interpreted as an infinite timeout. The value is in milliseconds.
KeepAlive	Applicable only in case of outbound interactions. Should be set to `true` to use connection pool feature.
BacklogQueue	Applicable in case of inbound interactions. This value indicates the maximum queue length for incoming connection indications (a request to connect). If a connection indication arrives when the queue is full, then the connection is refused.

Note:

There is a change in behavior in which the ServerSocket is created when you upgrade from Oracle Release 11g to Release 12g. Because of this, remote clients might not able to connect to the ServerSocket when the `hostname` is configured as *localhost*. As a workaround, the `localhost` should be changed to *hostname*.

The following is a sample `weblogic-ra.xml` file:

```
<wls:connection-instance>
                <wls:description>Socket Adapter</wls:description>
                <wls:jndi-name>eis/socket/SocketAdapter</wls:jndi-name>
                <wls:connection-properties>
                    <wls:pool-params>
                        <wls:initial-capacity>0</wls:initial-capacity>
                        <wls:max-capacity>200</wls:max-capacity>
                        <wls:capacity-increment>5</wls:capacity-increment>
                        <wls:shrinking-enabled>true</wls:shrinking-enabled>

<wls:shrink-frequency-seconds>60</wls:shrink-frequency-seconds>

<wls:connection-creation-retry-frequency-seconds>2</wls:connection-creation-retry
-frequency-seconds>

<wls:connection-reserve-timeout-seconds>5</wls:connection-reserve-timeout-seconds>

<wls:match-connections-supported>true</wls:match-connections-supported>
                        <wls:use-first-available>true</wls:use-first-available>
                    </wls:pool-params>

<wls:transaction-support>NoTransaction</wls:transaction-support>

<wls:reauthentication-support>true</wls:reauthentication-support>
                    <wls:properties>
                        <wls:property>
```

```
                    <wls:name>Host</wls:name>
                    <wls:value>localhost</wls:value>
                </wls:property>
                <wls:property>
                    <wls:name>Port</wls:name>
                    <wls:value>12110</wls:value>
                </wls:property>
                <wls:property>
                    <wls:name>Timeout</wls:name>
                    <wls:value>10000</wls:value>
                </wls:property>
                <wls:property>
                    <wls:name>BacklogQueue</wls:name>
                    <wls:value>0</wls:value>
                </wls:property>
                <wls:property>
                    <wls:name>KeepAlive</wls:name>
                    <wls:value>True</wls:value>
                </wls:property>
            </wls:properties>
            <wls:res-auth>Application</wls:res-auth>
        </wls:connection-properties>
    </wls:connection-instance>
```

Note:

To set up connection pooling, you must set the `KeepAlive` property to `true`.

Modeling a Handshake

A handshake may be required to negotiate a connection with a client or a server socket.

Modeling an Outbound Handshake

The outbound XSLT uses an input corresponding to the invoked message. The outbound XSLT writes to the socket output stream by using extension functions. The output is dummy for unidirectional or a response for bidirectional communication.

The following example demonstrates the modeling of a Synchronous Request/ Response communication paradigm:

```
<xsl:stylesheet
...
xmlns:socket="http://www.oracle.com/XSL/Transform/java/
oracle.tip.adapter.socket.ProtocolTranslator" />
xmlns:request="http://www.TragetNameSpace.com/Request" >

  <xsl:template match="/">

    <!-- Write the entire content of "books" element using translator -->
    <xsl:variable name="username" select="socket:socketWriteWithXlation(.)" />

    <!-- Read the stream using translator -->
    <xsl:copy-of select="socket:socketReadWithXlation()" />

  </xsl:template>
</xsl:stylesheet>
```

Modeling an Inbound Handshake

The inbound XSLT uses a dummy input, reads the socket input stream through extension functions, and constructs the XML record to be published.

The following example demonstrates a handshake in which the client sends across a user identification terminated by a comma (,) and a password terminated by a semicolon (;) for validation, and then sends the message payload:

```
<xsl:stylesheet
...
xmlns:socket="http://www.oracle.com/XSL/Transform/java/
oracle.tip.adapter.socket.ProtocolTranslator" />
  <xsl:template match="/">
    <!-- Read the user name -->
    <xsl:variable name="username"
select="socket:socketRead('terminated','terminatedBy=,')" />
    <!-- Read password if user name is correct -->
    <xsl:if test="normalize-space($username)='user'">
      <xsl:variable name="password"
select="socket:socketRead('terminated','terminatedBy=;')" />
      <!-- If password is correct proceed to read the payload using translator
-->
      <xsl:if test="normalize-space($password)='password'">
        <!--  Send an OK -->
        <xsl:variable name="ack1" select="socket:socketWrite('001','','')" />

        <output>
<!--  Wait for the payload -->
          <xsl:copy-of select="socket:socketReadWithXlation()" />
        </output>

      </xsl:if>
      <!--  Send an error -->
      <xsl:else><xsl:variable name="ack2"
select="socket:socketWrite('000','','')" /></xsl:else>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

Designing an XSL File Using the XSL Mapper Tool

You can design an XSL file by using the XSL mapper tool for Oracle Socket Adapter. The following sections describe the procedure for designing XSL for different communication scenarios:

- Designing XSL for Inbound Synchronous Request/Reply
- Designing XSL for Outbound Synchronous Request/Reply

Designing XSL for Inbound Synchronous Request/Reply

This section describes the procedure for designing XSL for an inbound synchronous request/reply scenario by using the XSL mapper tool:

Note:

To perform this use case, you require the following files from the `artifacts.zip` file contained in the `Adapters-101SocketAdapterHelloWorld` sample:

- `artifacts/schemas/HelloWorld.xsd`

You can access the `Adapters-101SocketAdapterHelloWorld` sample on the Oracle SOA Sample Code site.

Copy this file to the `HelloWorldComposite\xsd` folder under the `HelloWorldComposite` project.

Design an SOA Composite

To design an SOA composite, perform the steps described in Designing the SOA Composite.

Note:

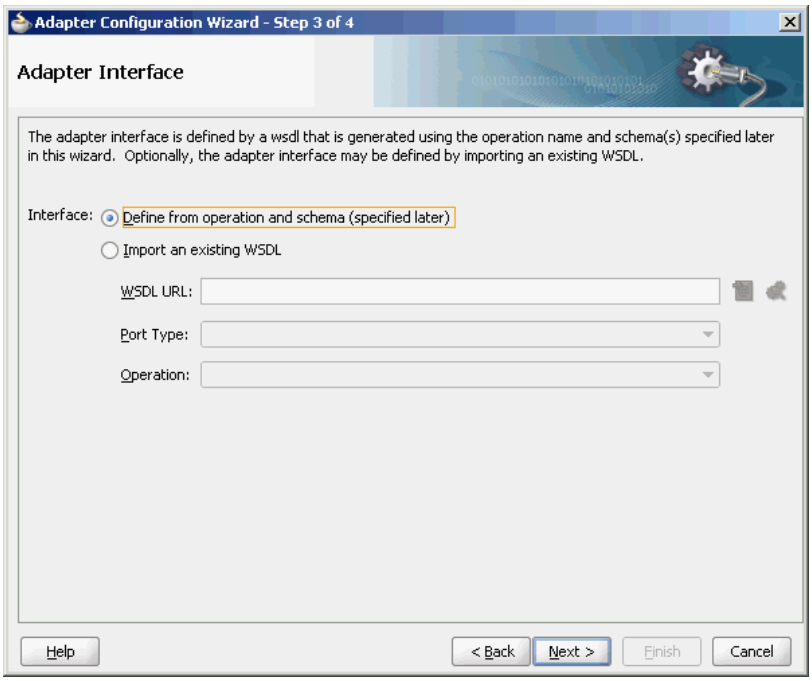
The steps provided in Designing the SOA Composite are applicable to a composite with Oracle BPEL PM. Alternatively, you can create a composite with Mediator.

Create an Inbound Oracle Socket Adapter

To create an inbound Oracle Socket Adapter service, perform the following steps:

1. Drag and drop **Socket Adapter** from the Components Palette to the Exposed Services swim lane. The Welcome page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter the service name, `HelloWorld` in the **Service Name** field and then click **Next**. The Adapter Interface page is displayed.
4. Select **Define from operation and schema (specified later)**, as shown in the Figure 5-16, and click **Next**. The Operation page is displayed.

Figure 5-16 The Adapter Configuration Wizard - Adapter Interface Page



5. Select **Inbound Synchronous Request/Reply** as the Operation Type and then click **Next**. The Socket Connection page is displayed.

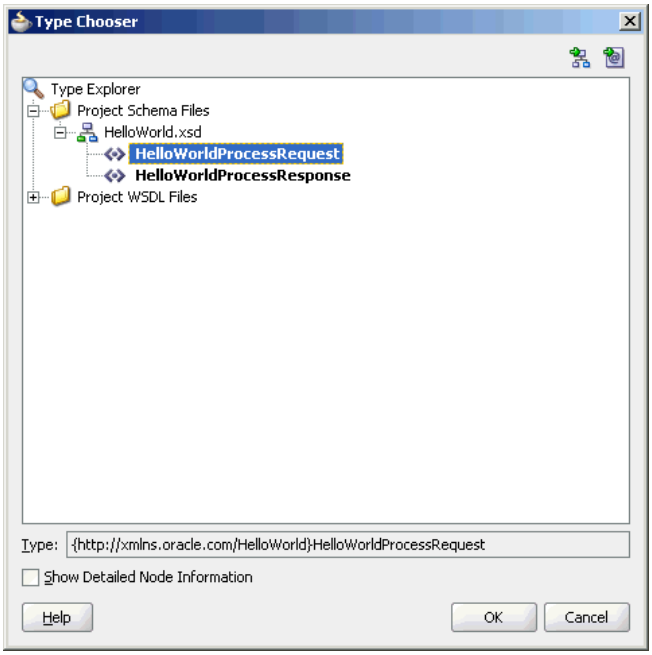
6. Enter `eis/socket/InboundSocketAdapter` in the **Socket Connection JNDI Name** field, as shown in Figure 5-17, and click **Next**. The Messages page is displayed.

Figure 5-17 The Adapter Configuration Wizard Socket Connection Page

7. Click **Browse For Schema File** that appears at the end of the URL field in the Request Message Schema box. The Type Chooser dialog is displayed.

8. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessRequest**, as shown in Figure 5-18.

Figure 5-18 The Type Chooser Dialog



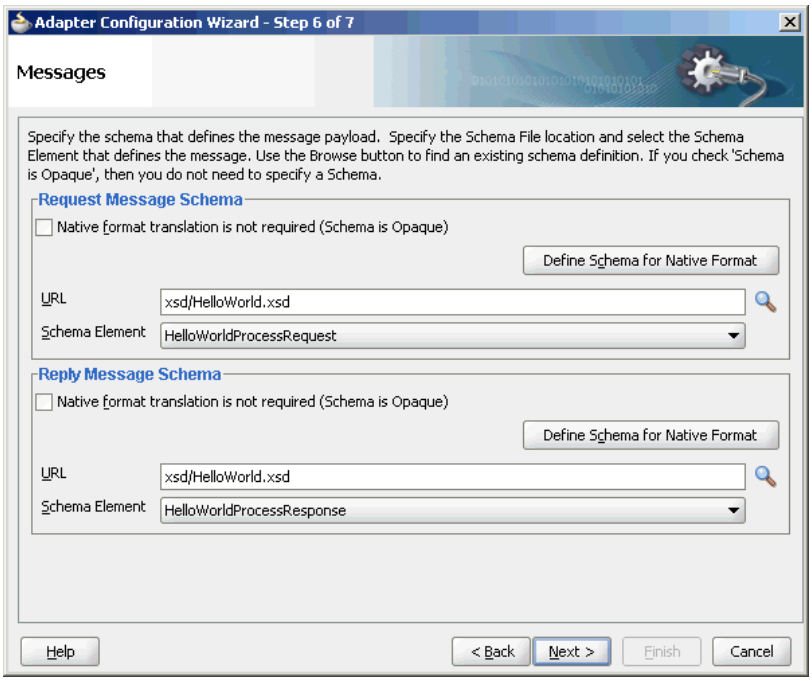
9. Click **OK**. The URL field in the Messages page is populated with the **HelloWorld.xsd** file.

10. Click **Browse For Schema File** that appears at the end of the URL field in the Reply Message Schema box. The Type Chooser dialog is displayed.

11. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessResponse**.

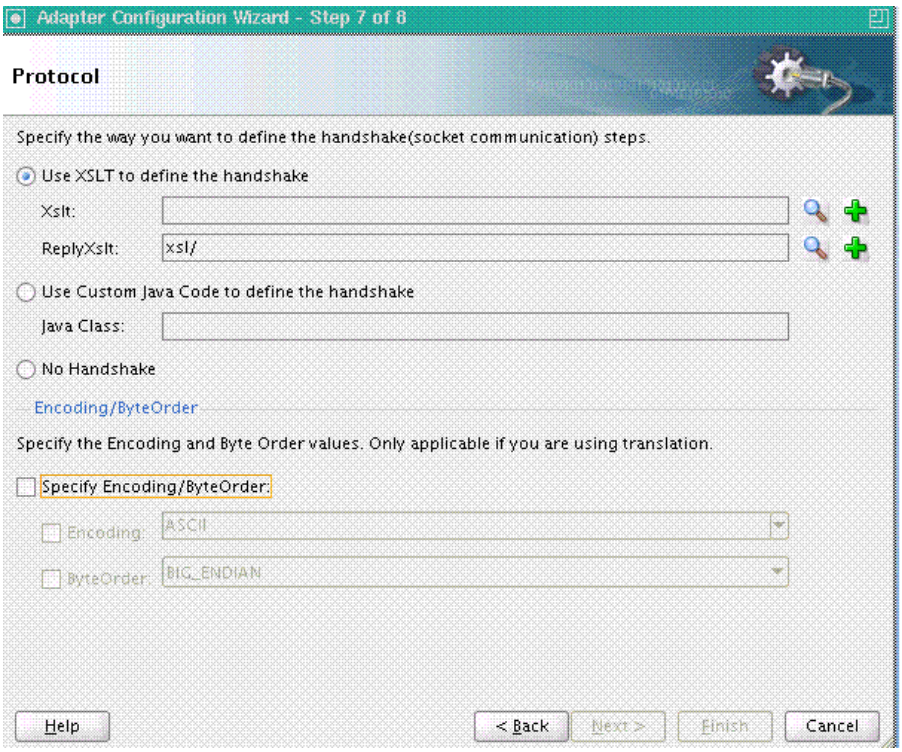
12. Click **OK**. The URL fields in the Messages page are populated with the **HelloWorld.xsd** files, as shown in Figure 5-19.

Figure 5-19 The Adapter Configuration Wizard File Messages Page



13. Click **Next**. The Protocol page is displayed, as shown in Figure 5-20.

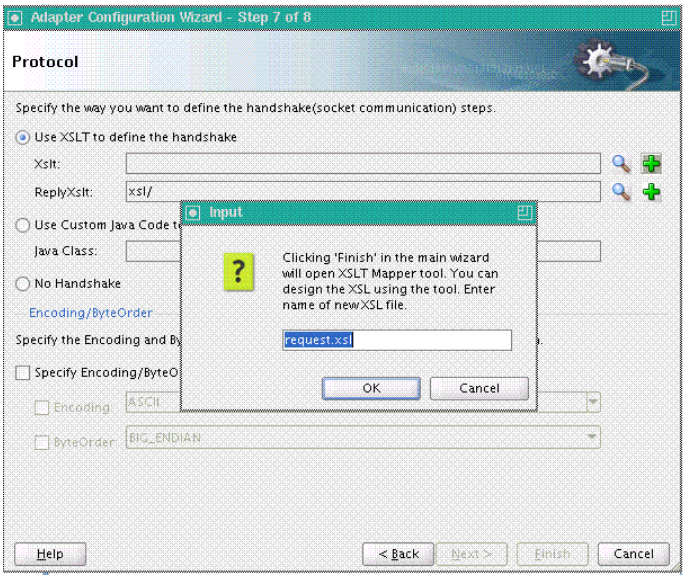
Figure 5-20 The Adapter Configuration Wizard - Protocol Page



14. Select **Use XSLT to define the handshake**.

15. Click the **create new xsl file** icon that appears at the end of the XSLT field. The Input dialog appears, as shown in Figure 5-21.

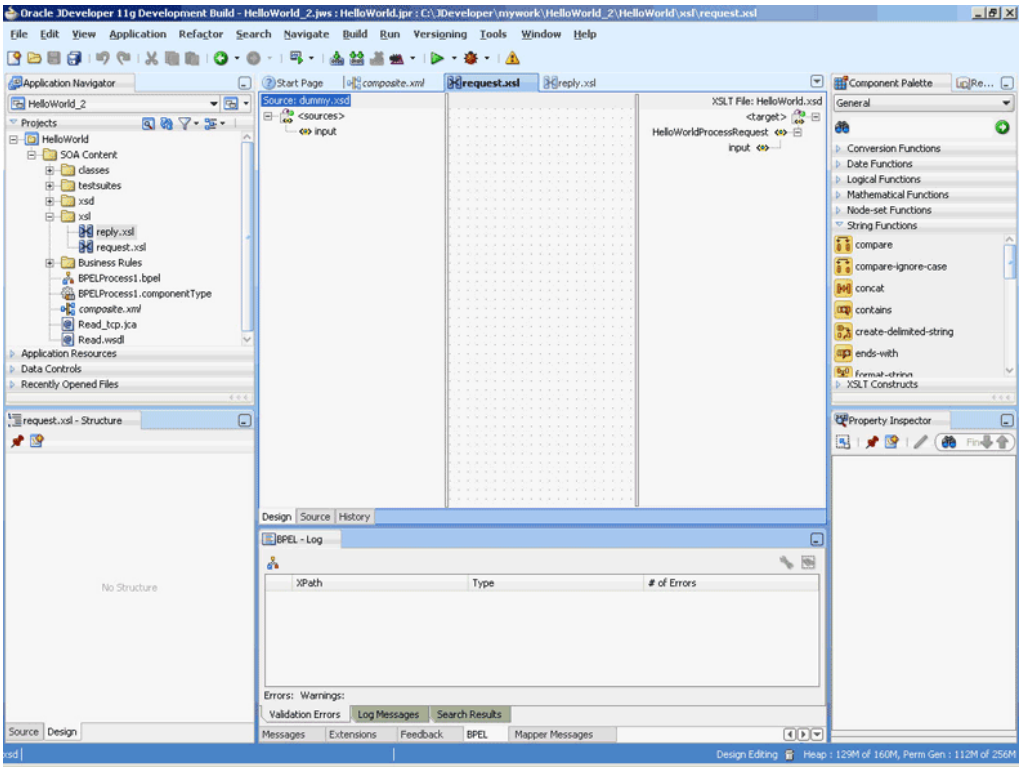
Figure 5-21 The input Dialog of the Protocol Page



16. Use the default value, **request.xsl**, as the name of the XSL file, as shown in Figure 5-21 and click **OK**.
17. Click the **create new xsl file** icon that appears at the end of the ReplyXslt field. The Input dialog appears.
18. Use the default value, **reply.xsl**, as the name of the XSL file, and click **OK**.
19. Click **Finish**. The request.xsl and the reply.xsl files are created.

    Figure 5-22 shows the request.xsl page.

Figure 5-22 The JDeveloper - request.xsl Page



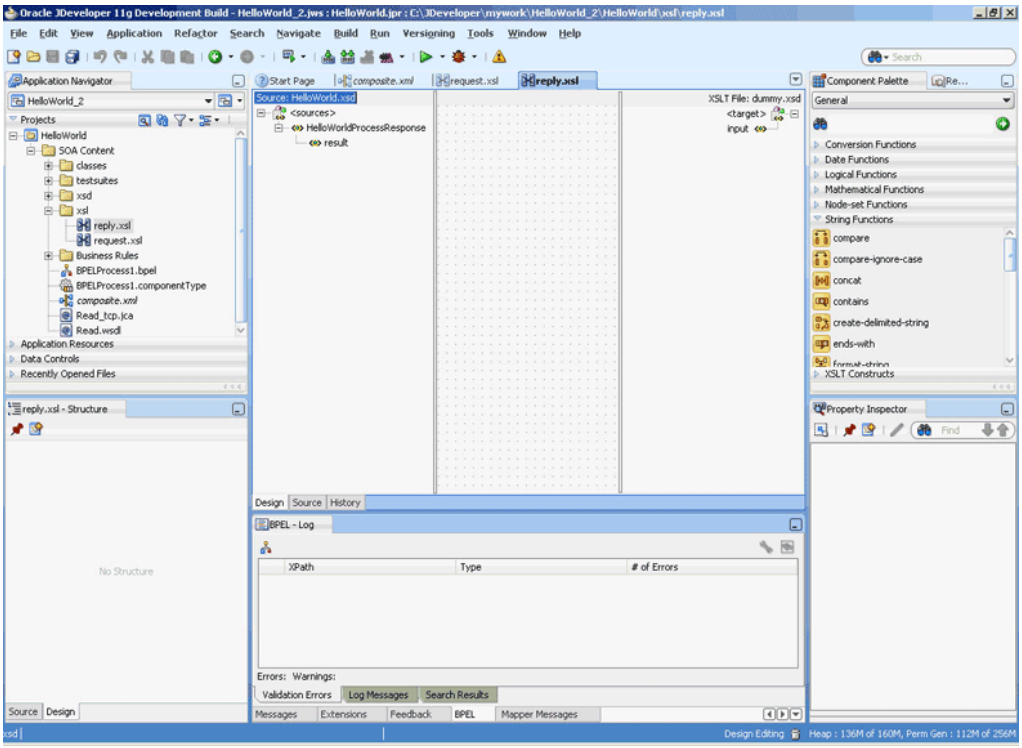
Note:

A `dummy.xsd` file appears in the left Source pane of the `request.xsl` page, which is used as the source for the XSL mapper tool.

In an inbound request scenario, Oracle Socket Adapter reads native data that is received by the socket and converts it to an XML format. That is, on the source side there is no XML file. Because the XSLT mapper always requires source and target XSD files, a dummy XSD file appears in the mapper tool.

Figure 5-23 shows the `reply.xsl` page.

Figure 5-23 The JDeveloper - reply.xsl Page



Note:

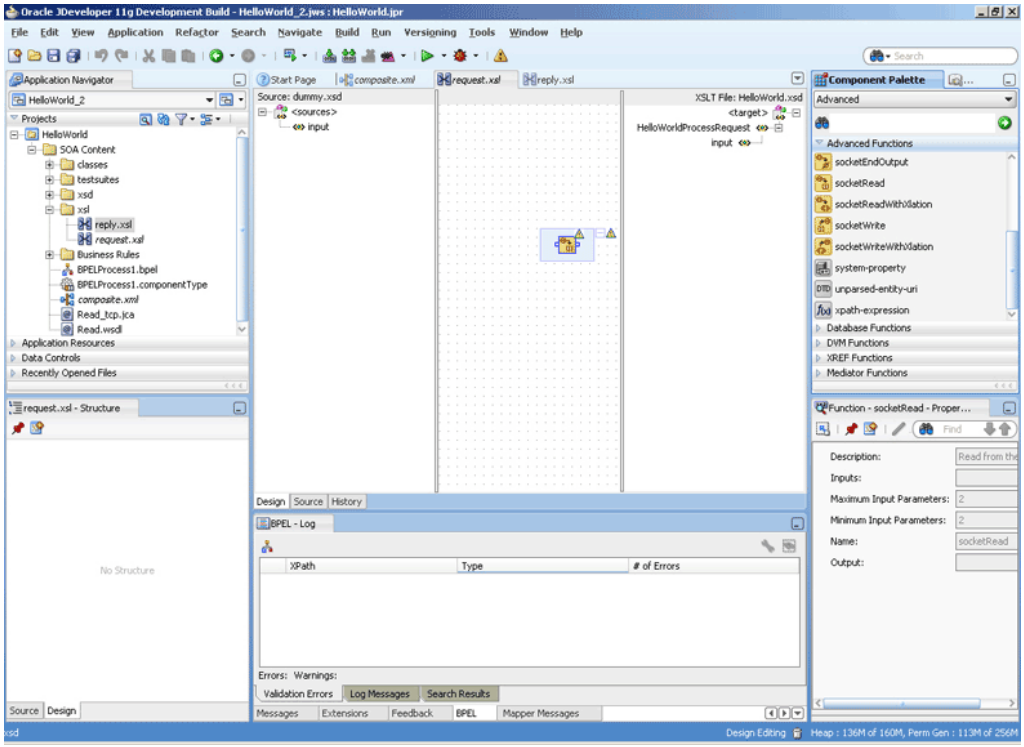
A `dummy.xsd` file appears in the right target pane of the `reply.xsl` page. This `dummy.xsd` file is used as the target for the XSL mapper tool.

---

20. Define the request part of the inbound synchronous request/reply operation as follows:

a. In the request.xsl page, drag and drop **socketRead** from the Advanced Functions list of the Components Palette to the middle pane, as shown in Figure 5-24.

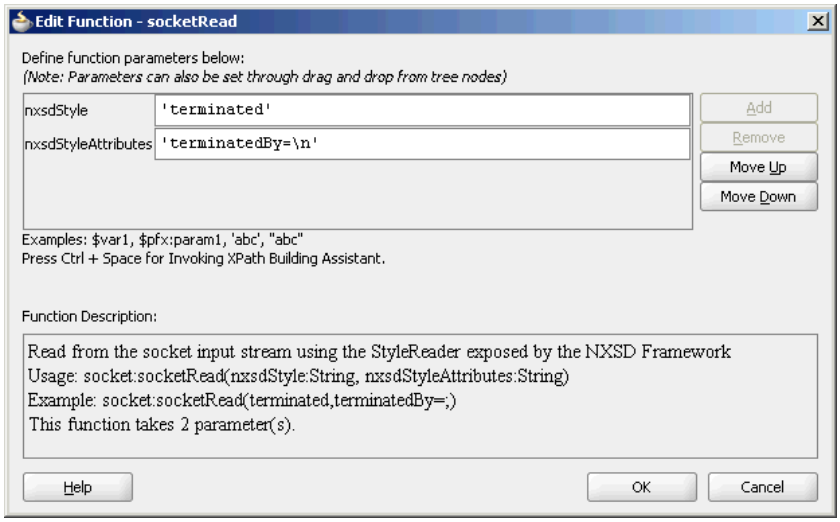
Figure 5-24 The JDeveloper - request.xsl Page



b. Double-click the **socketRead** advanced function. The Edit Function - socketRead dialog appears.

c. Enter the function parameters in the **nxsdStyle** and **nxsdStyleAttributes** fields, as shown in Figure 5-25.

Figure 5-25 The Edit Function - socketRead Dialog



Note:

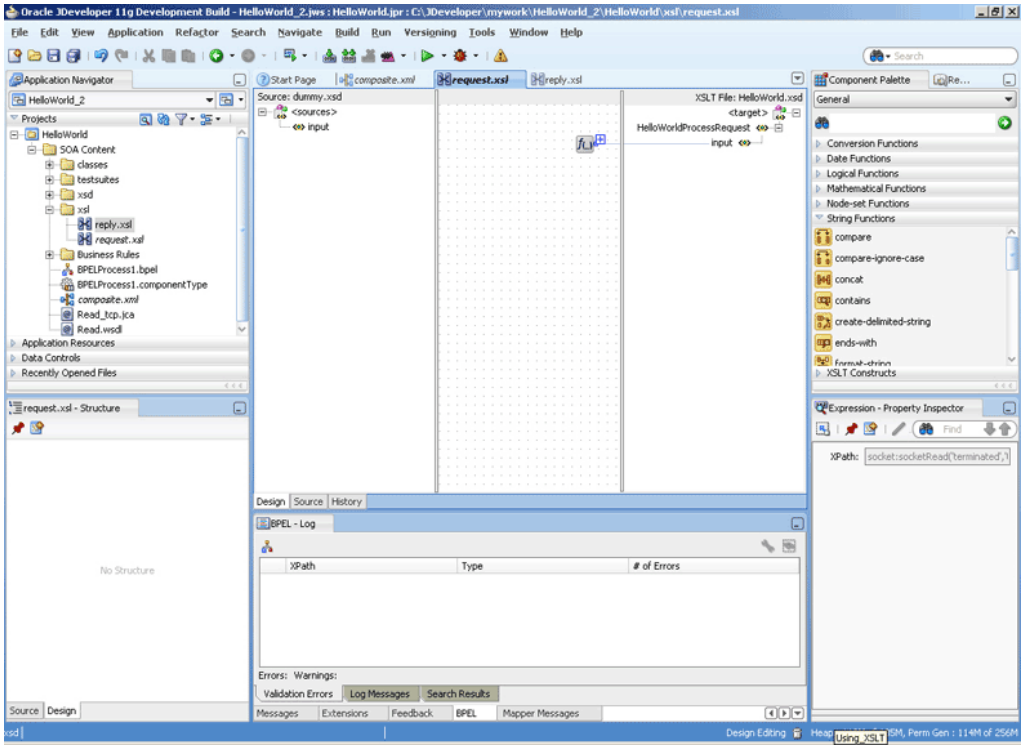
The `socketRead` function reads from the socket input stream by using the `StyleReader` exposed by the NXSD framework.

---

d. Click **OK**. The request.xsl (XSL mapper tool) page is displayed.

e. Link the `sockRead` function in the middle pane to the target `input` node on the right pane. The request.xsl (XSL mapper tool) with the XSL mapping is displayed, as shown in Figure 5-26.

Figure 5-26 The JDeveloper - request.xsl Page



21. Define the reply part of the inbound synchronous request/reply operation as follows:

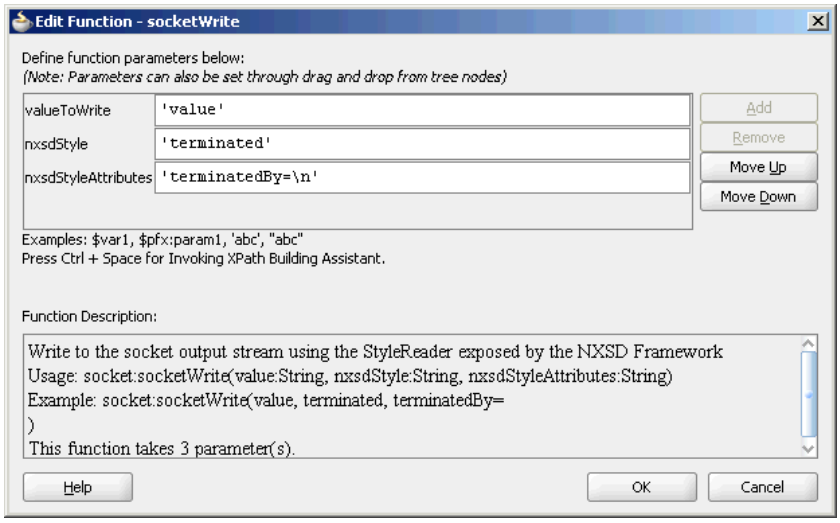
a. From the Components window list, select **Advanced**, and then select **Advanced Functions**. A list of advanced functions are displayed.

b. In the reply.xsl page, drag and drop **socketWrite** from the Advanced Functions list of the Components window to the middle pane.

c. Double-click the **socketWrite** advanced function. The Edit Function - socketWrite dialog appears.

d. Enter the function parameters in the **valueToWrite**, **nxsdStyle**, and **nxsdStyleAttributes** fields, as shown in Figure 5-27.

Figure 5-27 The Edit Function - socketWrite Dialog



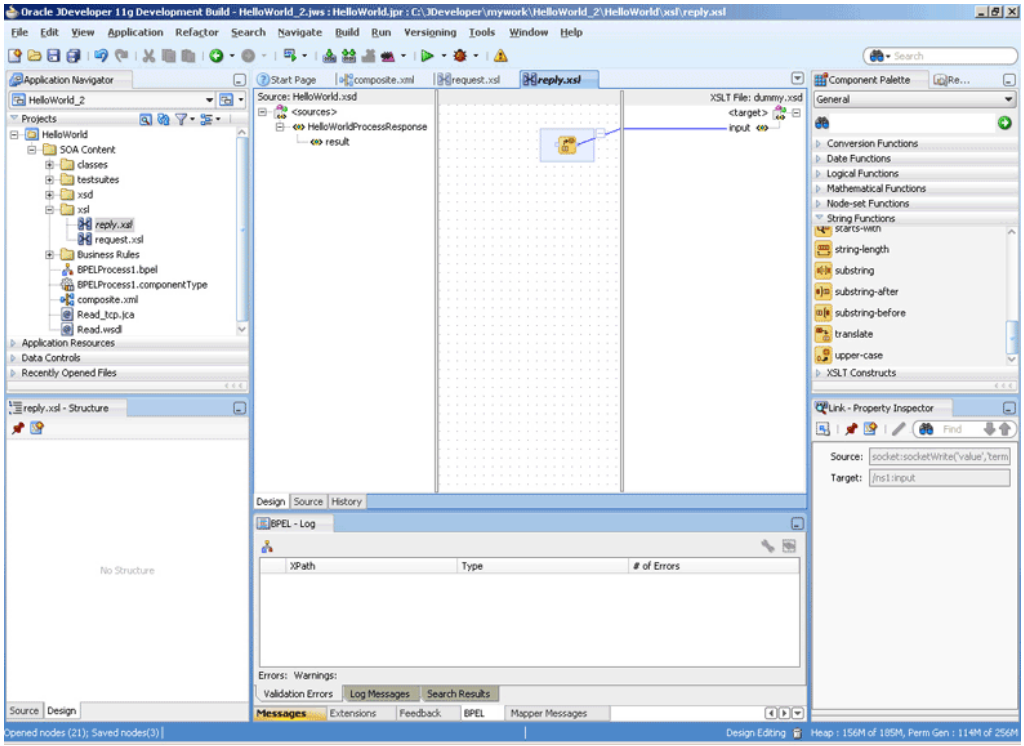
Note:

The `socketWrite` function writes to the socket output stream by using the `StyleReader` exposed by the NXSD framework.

e. Click **OK**. The reply.xsl (XSL mapper tool) page is displayed.

f. Link the `sockWrite` function in the middle pane to the target `input` node on the right pane. The reply.xsl (XSL mapper tool) with the XSL mapping is displayed, as shown in Figure 5-28.

Figure 5-28 The JDeveloper - reply.xsl Page



22. Click **File**, **Save All**. The request.xsl and reply.xsl files for the inbound Oracle Socket Adapter are created.

Designing XSL for Outbound Synchronous Request/Reply

This section describes the procedure for designing XSL for an outbound synchronous request/reply scenario by using the XSL mapper tool:

Note:

To perform this use case, you require the following files from the `artifacts.zip` file contained in the `Adapters-101SocketAdapterHelloWorld` sample:

- `artifacts/schemas/HellowWorld.xsd`

You can access the `Adapters-101SocketAdapterHelloWorld` sample on the Oracle SOA Sample Code site.

Copy the `HelloWorld.xsd` file to `HelloWorldComposite\xsd` under the `HelloWorldComposite` project:

Design an SOA Composite

To design an SOA composite, perform the steps described in Designing the SOA Composite..

Note:

The steps provided in Designing the SOA Composite are applicable to a composite with Oracle BPEL PM. Alternatively, you can create a composite with Mediator.

Create an Outbound Oracle Socket Adapter

To create an outbound Oracle Socket Adapter reference, perform the following steps:

1. Drag and drop **Socket Adapter** from the Components Palette to the External References swim lane. The Welcome page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter the service name, `HelloWorld` in the **Service Name** field and then click **Next**. The Adapter Interface page is displayed.
4. Select **Define from operation and schema (specified later)**, as shown in the Figure 5-16 and click **Next**. The Operation page is displayed.
5. Select **Outbound Synchronous Request/Reply** as the Operation Type and then click **Next**. The Socket Connection page is displayed.
6. Enter `eis/socket/OutboundSocketAdapter` in the **Socket Connection JNDI Name** field and click **Next**. The Messages page is displayed.
7. Click **Browse For Schema File** that appears at the end of the URL field in the Request Message Schema box. The Type Chooser dialog is displayed.

8. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessRequest**, as shown in Figure 5-18.

9. Click **OK**. The URL field in the Messages page is populated with the `HelloWorld.xsd` file.

10. Click **Browse For Schema File** that appears at the end of the URL field in the Reply Message Schema box. The Type Chooser dialog is displayed.

11. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessResponse**.

12. Click **OK**. The URL fields in the Messages page are populated with the `HelloWorld.xsd` files, as shown in Figure 5-19.

13. Click **Next**. The Protocol page is displayed.

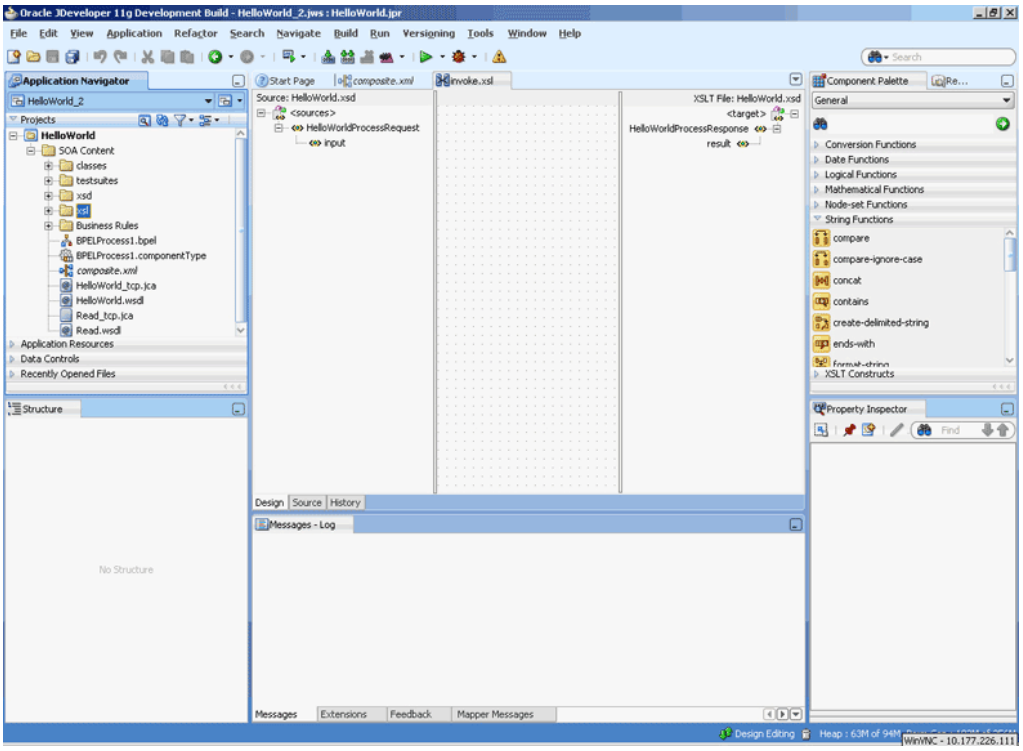
14. Select **Use XSLT to define the handshake**.

15. Click the **create new xsl file** icon that appears at the end of the Xslt field. The Input dialog appears.

16. Use the default value, `invoke.xsl`, as the name of the XSL file and click **OK**.

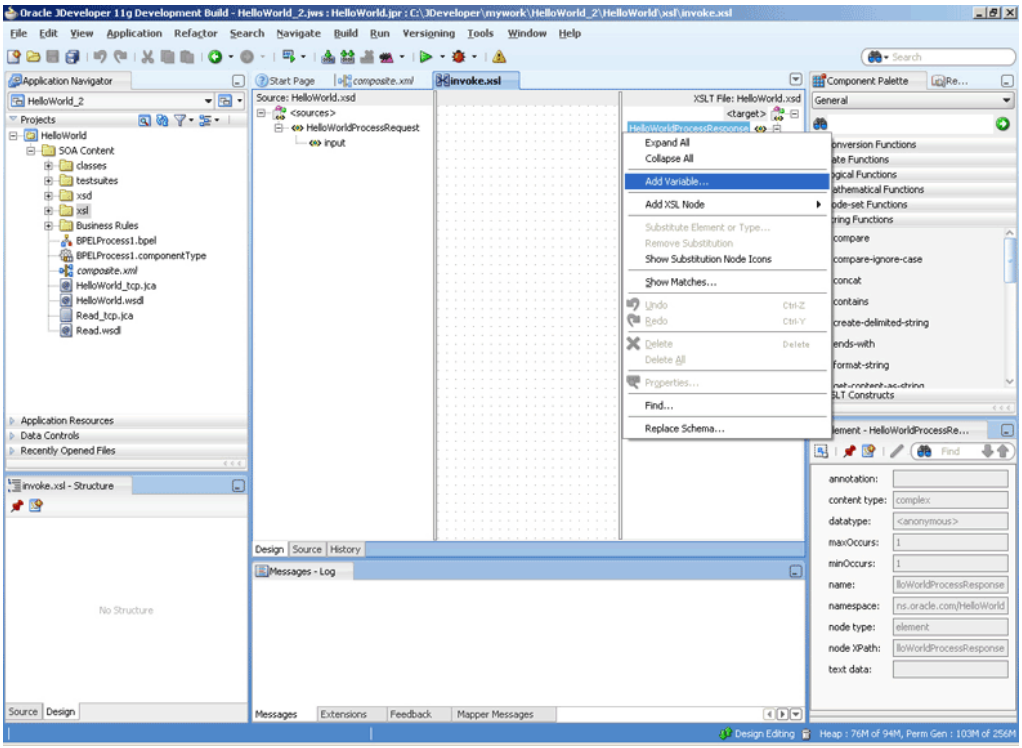
17. Click **Finish**. The invoke.xsl file appears in the XSL mapper tool, as shown in Figure 5-29.

Figure 5-29 The JDeveloper - invoke.xsl Page



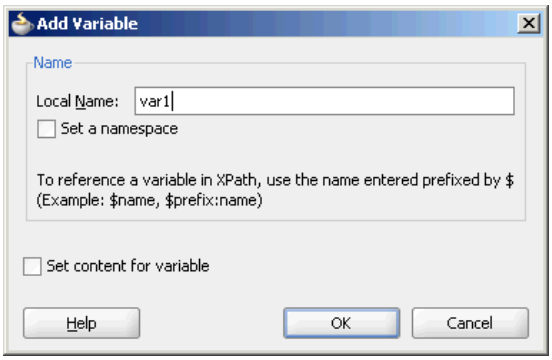
18. Right-click the **HelloWorldProcessResponse** element on the target side. A menu is displayed, as shown in Figure 5-30.

Figure 5-30 The JDeveloper - invoke.xsl Page



19. Click **Add Variable...**. The **Add Variable** dialog is displayed, as shown in Figure 5-31.

Figure 5-31 The Add Variable Dialog



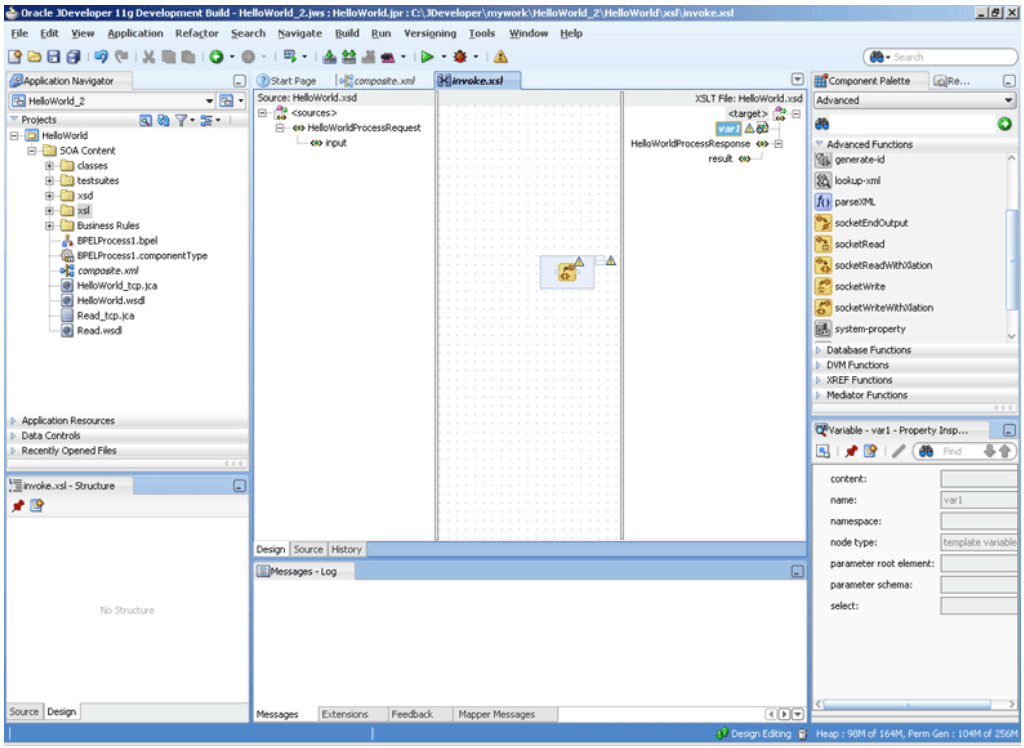
20. Enter `var1` in the **Local Name** field, and click **OK**. The var1 variable is added to the target pane of the XSL mapper tool.

21. From the Components window list, select **Advanced**; then, select **Advanced Functions**. A list of advanced functions is displayed.

22. Define the request part of the outbound synchronous request/reply operation, to write the data to the socket server, as follows:

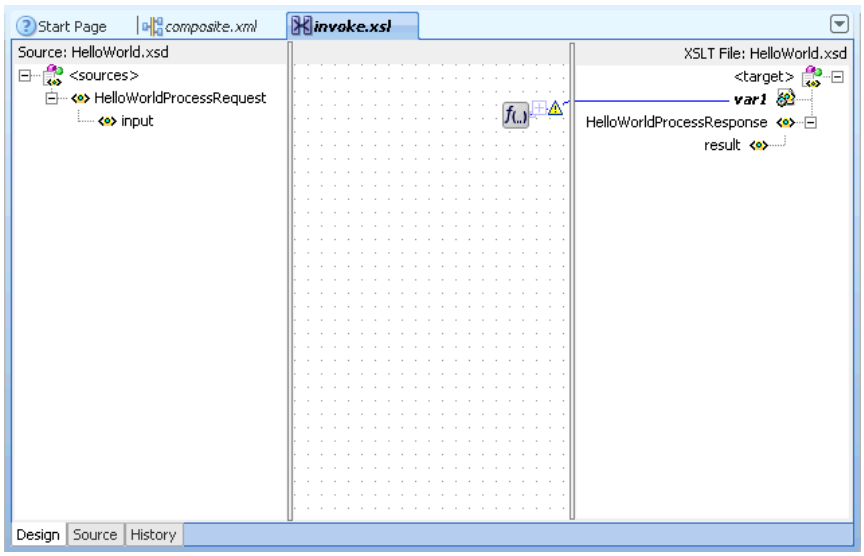
    a. Drag and drop **socketWriteWithXlation** from the Advanced Functions list of the Components window to the middle pane, as shown in Figure 5-32.

Figure 5-32 The JDeveloper - invoke.xsl Page



b. Drag the **var1** node to the socketWriteWithXlation function. A link is created, as shown in Figure 5-33.

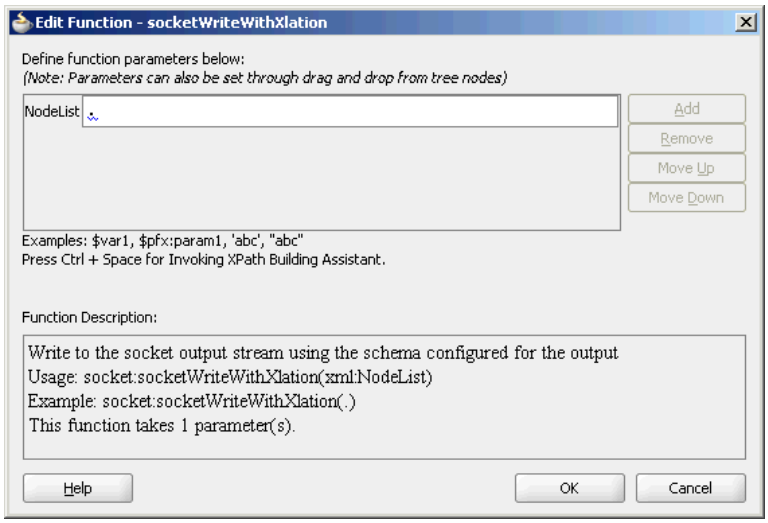
Figure 5-33 The JDeveloper - invoke.xsl Page



c. Double-click the **socketWriteWithXlation** advanced function. The Edit Function - socketWriteWithXlation dialog appears.

d. Enter a dot (.) in the **NodeList** field, as shown in Figure 5-34.

Figure 5-34 The Edit Function - socketWriteWithXlation Dialog



Note:

The `socketWriteWithXlation` function writes to the socket output stream using the schema configured for the output.

The dot (.) specified in the NodeList field signifies writing the HelloWorldProcessRequest to the top level node.

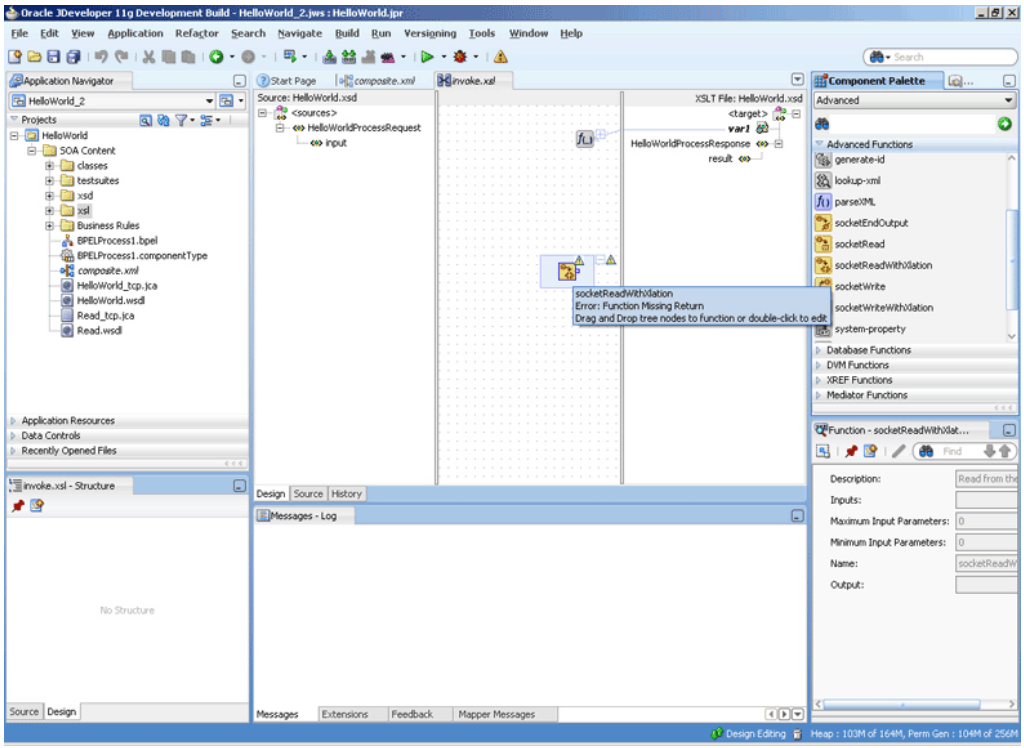
e. Click **OK**. A Warning dialog appears.

f. Click **Yes**. The invoke.xsl page is displayed. The request part of the Synchronous Request/Reply operation is defined.

23. Define the reply part of the outbound synchronous request/reply operation as follows:

a. Drag and drop **socketReadWithXlation** from the Advanced Functions list of the Components window to the middle pane, as shown in Figure 5-35.

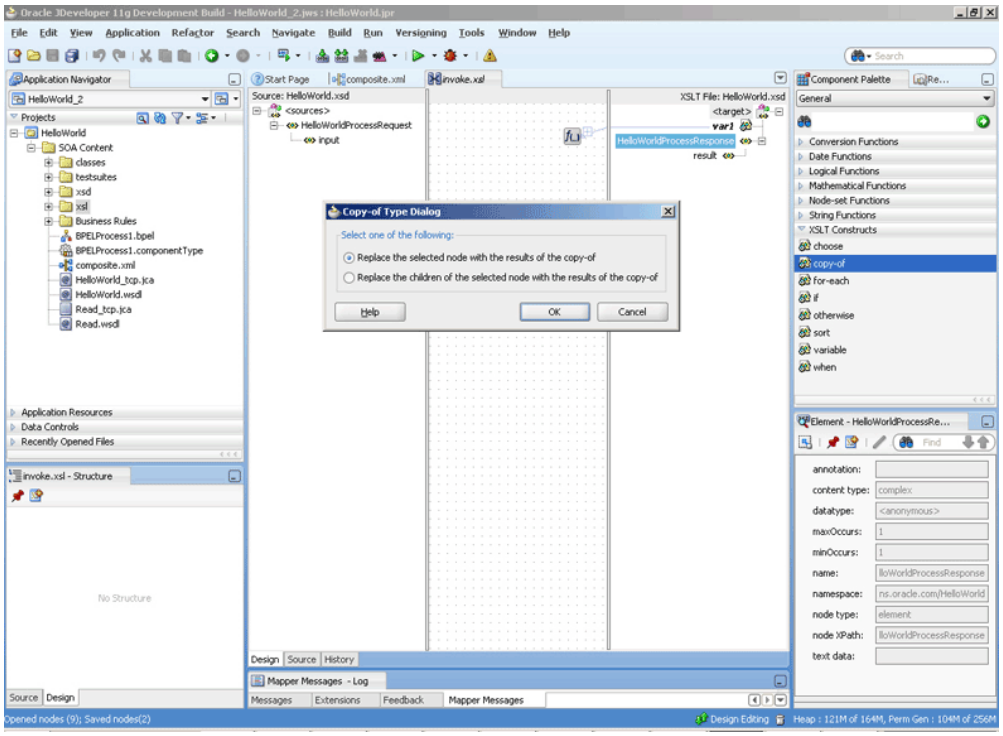
Figure 5-35 The JDeveloper - invoke.xsl Page



b. From the Components window list, select **General**; then, select **XSLT Constructs**. A list of XSLT constructs is displayed.

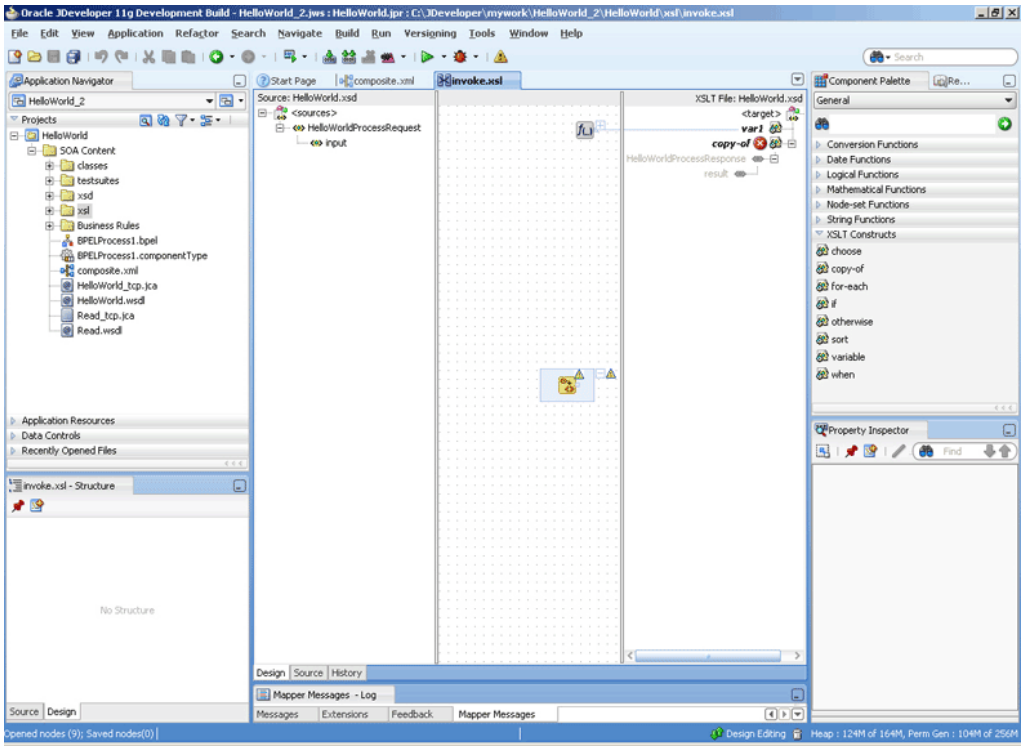
c. Drag **copy-of** from the Components window to HelloWorldProcessResponse in the target pane. The Copy-of Type Dialog appears, as shown in Figure 5-36.

Figure 5-36 The JDeveloper - invoke.xsl Page with Copy-of Type Dialog



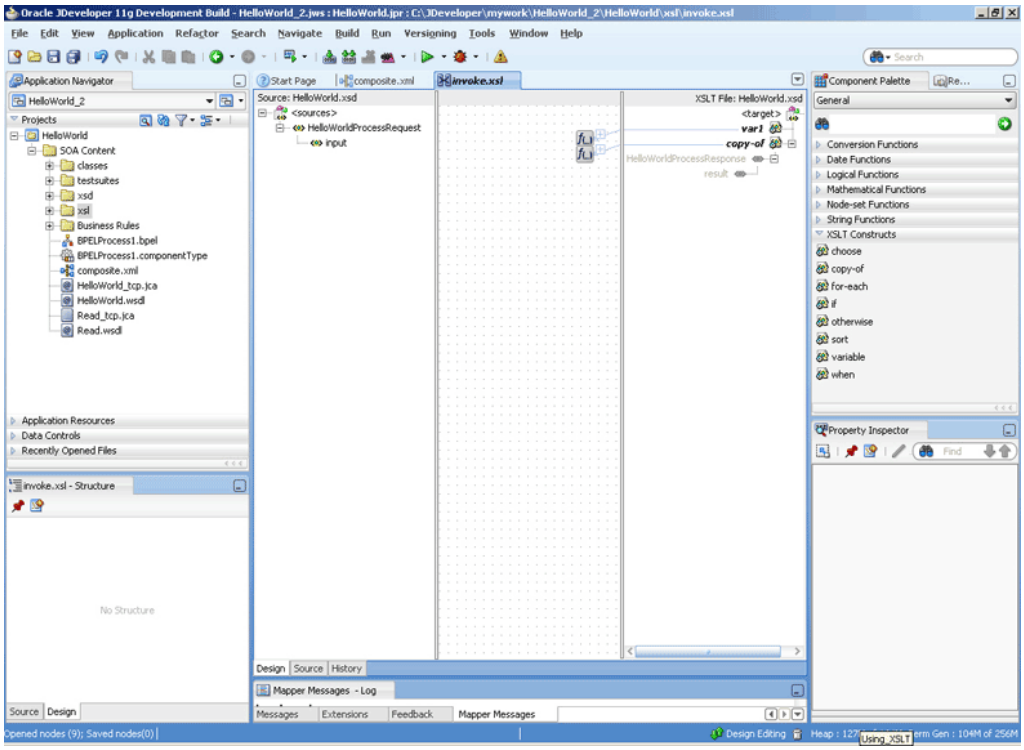
d. Click **OK**. The invoke.xsl (XSL mapper tool) page is displayed with the copy-of XSLT construct added to the target pane, as shown in Figure 5-37.

Figure 5-37 The JDeveloper - invoke.xsl Page



e. Drag the **copy-of** XSLT construct to the socketReadWithXlation function. A link is created, as shown in Figure 5-38.

Figure 5-38 The JDeveloper - invoke.xsl Page



24. Click **File**, **Save All**. The Outbound Synchronous Request/Reply handshake is defined.

Specifying a TCP Port in a Configuration Plan For an Oracle Socket Adapter

To specify a TCP port in a configuration plan for an Oracle Socket Adapter, perform the following steps (where <service-name> is Service name):

1. Add a port property to your <service-name>_tcp.jca file:

```
<property name="Port" value="Port"/>
```

2. Add the following code to your configuration plan XML file:

```
<service_name=<XXXXXX">
 <property>
  <property name="Port">
   <replace>2222</replace>
 </property>
 <binding type'"jca"/>
</service>
```

3. Add the port property to your .xml file under the service element and specify a default value, in this example, 1111

```
<service name="XXXXX" ui:wsdlLocation
  <interface.wsdl.interface="..."/>
  <binding.jca config="XXXX_tcp.jca"/>
    <property name="Port" type=xs:string" many="false"
```

```
override="may">1111</property>
</service>
```

4. Deploy your composite with the configuration plan.

When deployed, the Oracle Socket Adapter listens on port 2222, as provided in the configuration plan.

If you deploy the composite without a configuration plan or if the configuration plan does not override the Port property, then the Oracle Socket Adapter listens on the socket that the composite.xml file's default Port property specifies (in this example, port 1111).

Java Script Support

The Socket Adapter currently uses XSLT and a custom Java plug-in as a way of defining complex handshakes for both inbound and outbound transactions.

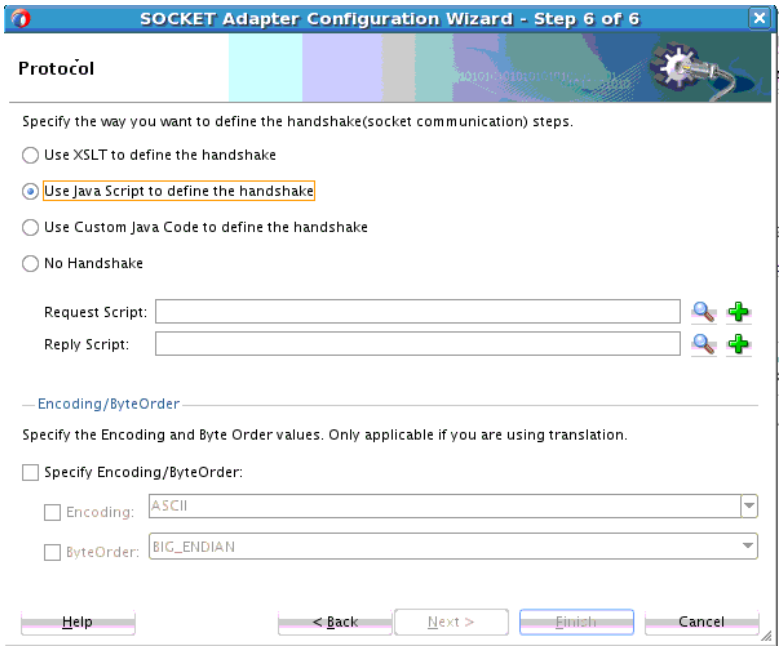
However, there is now an additional way to define handshakes for these transactions, through Javascript support. This feature enables you to write your own Javascripts that can be called to define simple and request-reply handshakes for both inbound and outbound transactions.

Using the Socket Adapter Configuration Wizard to Define Scripts to Use

To define handshakes for transactions, using Javascripts you have prepared:

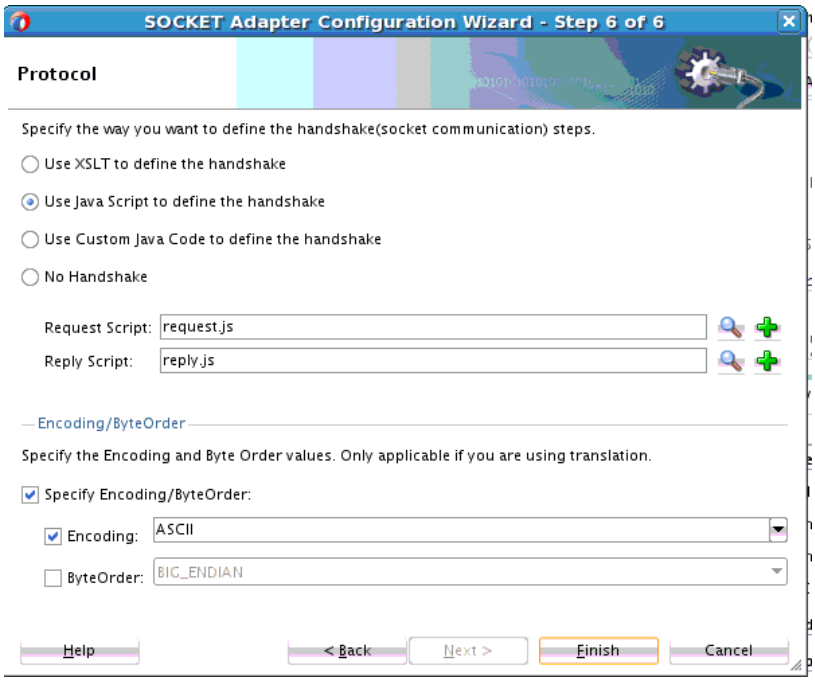
1. On the Protocol Screen, choose Use Javascript to Define the Handshake

Figure 5-39 The Socket Adapter Configuration Wizard with Javascript Chosen



2. Click **Finish**. On the refreshed Protocol Screen, add or browse to a location where your scripts are defined. In Figure 5-40, a request and reply script are provided. The rest of the screen is for translation purposes.

Figure 5-40 The Socket Adapter Configuration Wizard Protocol Screen with Additional Choices for Handshake Definition



3. Click **Next** or **Finish**. You have completed configuring the Socket Adapter with Javascripts that help provide socket communication.

Reporting

Socket Adapter XSLT and Javascript use cases define a communication protocol to communicate with the client/server sockets; typically this communication does not involve much data transfer, compared to that seen in NXSD-based usecases. Because there is not much value in capturing the message size in these use cases, the message statistics are not captured when you deploy a Socket Adapter and enable the snapshots report options provided in the Fusion Middleware Control.

Note that Adapter Reports statistics are collected only for NXSD and XSD translation. These statistics are not supported for XSLT, Javascript and custom protocols.

Sample Script

The following Java Script is a sample Java Script for use with this feature.

```
importPackage(java.io);
importPackage(java.lang);
importPackage(javax.xml.parsers);

println("=====> Under Script Execution");
var inputStream = InputStreamKey;
var outputStream = OutputStreamKey;
var requestPayloadElement = RequestPayloadElementKey;
var ReplyDOMResult = ReplyDOMResultKey;

var isr = new InputStreamReader(inputStream);
var osw = new OutputStreamWriter(outputStream);
var br = new BufferedReader(isr);

println("=====> starting handshake");
```

```
var reply = doHandshake();

createDOM(reply);

function writeToServer(request) {
    println("=====> Under writeToServer");
    var bw = new BufferedWriter(osw);
    bw.write(request);
    bw.newLine();
    bw.flush();
}

function doHandshake() {
    println("=====> Under extractPayload ");
    var nodeList = requestPayloadElement.getChildNodes();
    var length = nodeList.getLength();
    println("=====> Length :" +  length);
        //var i = 0;
        var name = "";
        var empId = "";
        for(var i=0 ; i < length; i++){
            println("===> Name" + i + "=" + nodeList.item(i).getNodeName()); // +
"; Value=" + nodeList.item(i).getFirstChild().getNodeValue());
            if(nodeList.item(i).getNodeName().indexOf("Name")!= -1){
                name = nodeList.item(i).getFirstChild().getNodeValue();
            }
            else if(nodeList.item(i).getNodeName().indexOf("EmpId")!= -1){
                empId = nodeList.item(i).getFirstChild().getNodeValue();
            }
        }
    /*println("=====> name: "+ nodeList.item(0).getNodeName());
    name = nodeList.item(0).getFirstChild().getNodeValue();


    println("=====> name: "+nodeList.item(1).getNodeName());
    empId = nodeList.item(1).getFirstChild().getNodeValue();*/

    println("=====> name :"+name);
    println("=====> empId :"+empId);

        var xmlPayload = "Hello World";/*"<ns1:Inbound-Element xmlns:ns1=\"http://
xmlns.oracle.com/pcbpel/demoSchema/csv\"><ns1:Name>" + name + "</ns1:Name>" +
                        "<ns1:EmpId>" + empId +"</ns1:EmpId></ns1:Inbound-
Element>";*/
        println("=====> xmlPayload :"+xmlPayload);
    writeToServer(name);
    writeToServer(empId);
    osw.close();
    var code = readFromServer();
    return code;
}

function readFromServer() {
    println("=====> Under readFromServer ");
    var str = ""; //= br.readLine();
    var tmpStr = "";
    while (true){
    tmpStr = br.readLine();
    if(tmpStr == -1 ||  tmpStr == null)
        break;
```

```
        println("tmpStr=" + tmpStr);
        str += tmpStr;
    }

    println("=====> Reply from Server :"+str);
    return  str;
}

function createDOM(code){
    println("=====> Under createDOM "); ;
        var fRootNode = null;
        var fDocument = null;
        var fTempNode = null;
        var NAMESPACE = "http://xmlns.oracle.com/pcbpel/demoSchema/csv";
        var ROOT_NODE = "Result";
        var dbf = DocumentBuilderFactory.newInstance();
        var db = dbf.newDocumentBuilder();
        fDocument = db.newDocument();

        fRootNode = fDocument.createElementNS(NAMESPACE, ROOT_NODE);
        fRootNode.appendChild(fDocument.createTextNode(code));

        ReplyDOMResult.setNode(fRootNode);
}
```

Socket Adapter NIO Support

The current socket adapter uses the thread-per-connection approach for inbound connections. The thread-per-connection approach uses an exclusive worker thread for each connection. Within the handling loop, a worker thread waits for new incoming data, processes the request, returns the response data. This behavior results in the blocking of the thread until an interaction completes.

However, in a scenario where a high number of concurrent users are using an inbound Socket Adapter, there is a wait for a thread once the operating system/thread pool limit reached. A large number of threads have their own memory footprints, which can cause "Out Of Memory" exceptions in certain cases.

The socket adapter now uses (New I/O) NIO APIs. The APIs of NIO have been designed to provide access to the low-level I/O operations of modern operating systems

NIO APIs reduce the number of threads to user-configurable values. This enables the Socket Adapter to achieve better performance and throughput in scenarios, where the system is loaded.

Note that the Socket Adapter supports NIO APIs only for the NXSD mode of translation. The reason for not supporting this in other modes is that NIO assumes an inherent request-reply model. a scenario where a request is always followed by a reply.

In other modes of translation, a request might not always followed by a reply. For example, a scenario where a Socket Server sends a welcome message once it receives a client connection.

Or, there could be a scenario where there are multiple handshakes (multiple reads followed by writes). In such scenario, the Socket Adapter would not know whether full data for the first read has arrived or not.

The Adapter provides support for configuring the number of nio processor threads for NIO events that are processing. The default value for this property is 1.

If you want to specify a value for the property `NioProcessorThreadCount` other then the default value of 1, you must manually add the property `NioProcessorThreadCount` to the appropriate JCA file(s) . In the following example, the value for `NioProcessorThreadCount` is set to 2.

Example - Manually Setting the Value for the Property NioProcessorThreadCount in the jca File

```
<adapter-config name="inboundRequestReplyService" adapter="Socket Adapter"
wsdlLocation="inboundRequestReplyService.wsdl"
                xmlns="http://platform.integration.oracle/blocks/adapter/fw/
metadata">
  <connection-factory location="eis/socket/SocketAdapter" adapterRef=""/>
  <endpoint-activation portType="InboundRequestReply_ptt"
operation="InboundRequestReply"
     UITransmissionPrimitive="InboundRequestReply">
    <activation-spec className=
              "oracle.tip.adapter.socket.SocketActivationSpec">
      <property name="TransMode" value="NXSD"/>
      <property name="NioProcessorThreadCount" value="2"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

If you want to use the older implementation where a thread count is created per interaction, use the following in your JCA file:

Example - Sample jca File for Specifying Thread Count per Interaction

```
<adapter-config name="inboundRequestReplyService"
      adapter="Socket Adapter"
                 wsdlLocation="inboundRequestReplyService.wsdl"
                  xmlns="http://platform.integration.oracle/blocks/adapter/fw/
metadata">
     <connection-factory location="eis/socket/SocketAdapter" adapterRef=""/>
     <endpoint-activation portType="InboundRequestReply_ptt"
       operation="InboundRequestReply"
            UITransmissionPrimitive="InboundRequestReply">
      <activation-spec className=
                 "oracle.tip.adapter.socket.SocketActivationSpec">
         <property name="TransMode" value="NXSD"/>
         <property name="SupportNIO" value="false"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

SSL Support for the Socket Adapter

SSL means Secure Socket Layer and is a Cryptographic Protocol that provides Communication Security over the Internet. It provides asymmetric cryptography for Key exchange and symmetric encryption for privacy, and encrypts a segment of network Connection at the Transport Layer.

The Oracle Platform Security Service Keystore Service, or OPSS Keystore Service enables you to manage keys and certificates for SSL, message security, encryption, and related tasks. You use the Keystore Service to create and maintain keystores that contain keys, certificates, and other artifacts.

If you do not provide `TrustStoreName` and `TrustStoreStripeName` when you use KSS (then Domain level Trust store, that is, system/trust, is used.)

SSL Support within the Socket Adapter

SSL Support within the Socket Adapter is required when you want to send or receive data over a Secure Socket connection. This adds a layer of security protection over the underlying network transport protocol, and includes:

- Integrity Protection
- Authentication
- Confidentiality (Privacy Protection)

SSL is supported only in non NIO scenarios.

SSL is not supported in Request-Reply scenarios with NXSD mode of translation. The translator (for NXSD or for opaque payloads) looks for the end-of-stream before translating from native format to XML SSL The Socket adapter API does not support closing of streams, hence the Socket adapter with NXSD translation is not supported for request-reply usecases.

Table 5-2 provides the Socket Adapter SSL Connection Factory properties, which includes a list of the SSL Connection Factory Properties when you configure SSL in KSS (Oracle PSS) mode.

Table 5-2 SSL Connection Factory Properties when Configured in KSS Mode

Value	Description
`SslEnable`	Specifies if Connection Factory is SSL
`KeyStorePassword`	Password for the key store
`TrustStorePassword`	Password for trust store
`SslProtocol`	Version of SSL protocol (SSLv3)
`KeyStoreProviderName`	Key store provider
`KeyStoreType`	Type of key store (=KSS)
`KeyStoreAlgorithm`	Algorithm for key manager Factory
`TrustStoreProviderName`	Trust store provider
TrustStoreType	Type of trust store (=KSS)
TrustStoreAlgorithm	Algorithm for key manager Factory
`NeedClientAuthentication`	Specifies whether the client authentication is required (Inbound).
`KeyStoreStripeName`	Name of the stripe within which key store is created.
`KeyStoreStripeName`	Name of the key store under the given stripe.
`TrustStoreName`	Name of the trust store under given stripe.
`TrustStoreStripeName`	Name of the stripe within which trust store is created.

Oracle Socket Adapter Use Cases

This section includes the following Oracle Socket Adapter use cases:

- Hello World
- Flight Information Display System

Oracle Socket Adapter Hello World

This use case is a simple HelloWorld use case which demonstrates the synchronous inbound request/response and synchronous outbound request/response modes of communication using Oracle Socket Adapter. The HelloWorld business process takes an input string from the Oracle Socket Adapter inbound service and publishes the message to the BPEL process.

The BPEL process invokes the Oracle Socket Adapter outbound service (a simple HelloWorld Server, which adds a prefix?Hello? to the input string and returns it) and returns the received string using a synchronous reply.

This use case includes the following sections:

- Prerequisites
- Designing the SOA Composite
- Creating the Inbound Service
- Creating the Outbound Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using the Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console)

Prerequisites

To perform this use case, you require the following files from the `artifacts.zip` file contained in the `Adapters-101SocketAdapterHelloWorld` sample:

- `artifacts/schemas/HelloWorld.xsd`
- `artifacts/xsl/request.xsl`
- `artifacts/xsl/reply.xsl`
- `artifacts/xsl/invoke.xsl`

You can access the `Adapters-101SocketAdapterHelloWorld` sample on the Oracle SOA Sample Code site.

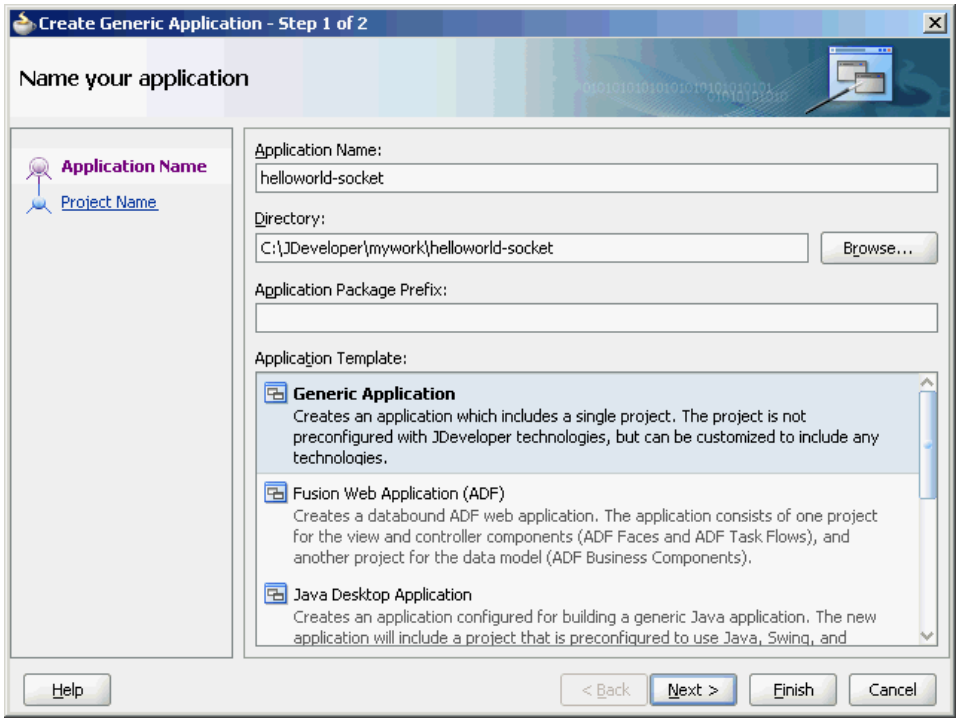
Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.

2. Enter `helloworld-socket` in the **Application Name** field, as shown in Figure 5-41, and then click **Next**. The Name your project page is displayed.

Figure 5-41 The Create SOA Application Dialog

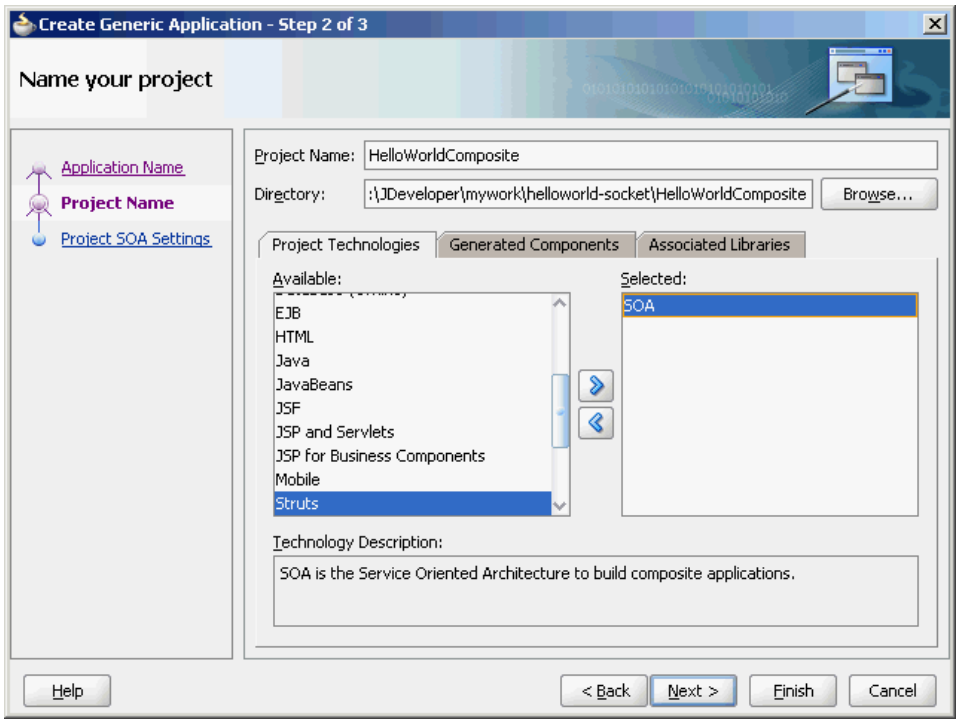


3. Click **OK**. The Name Your Project dialog is displayed.

4. Enter `HelloWorldComposite` in the **Project Name** field, and then select **SOA** under Project Technologies and move it to the **Selected** box by clicking the right-arrow, as shown in Figure 5-42.

   The HelloWorld application and the HelloWorldComposite project appear in the Application Navigator.

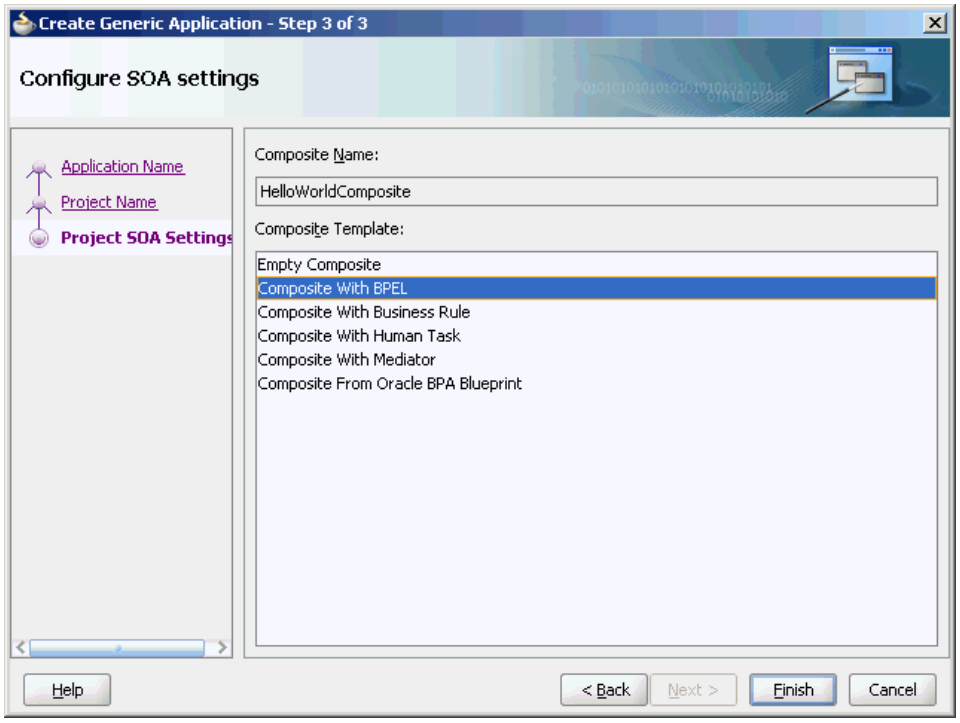
Figure 5-42 The Create Project Dialog



5. Click **Next**. The Configure SOA settings dialog appears.

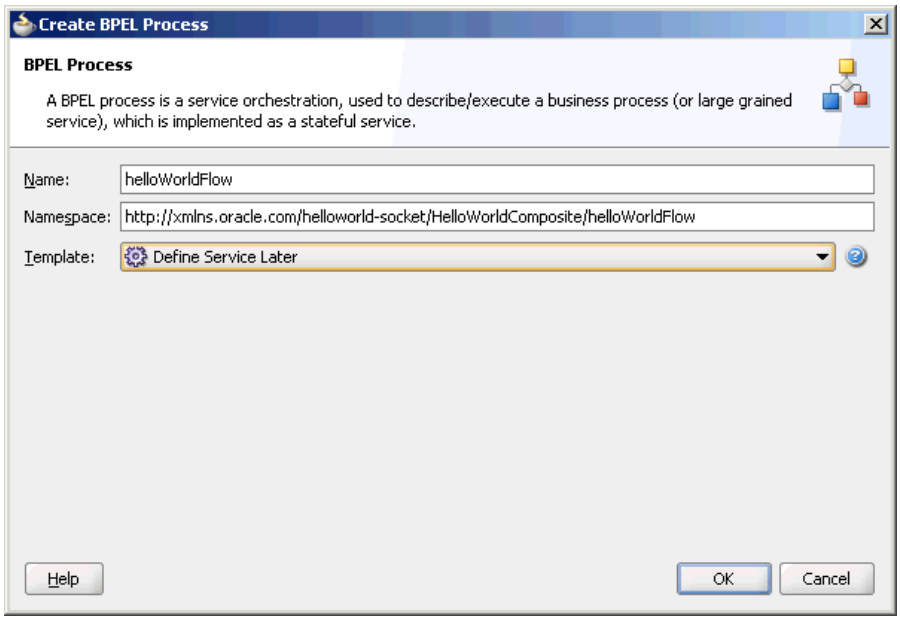
6. Select **Composite With BPEL** in the Composite Template box, as shown in Figure 5-43, and click **Finish**. The **Create BPEL Process** dialog is displayed.

Figure 5-43 The Configure SOA Settings Dialog



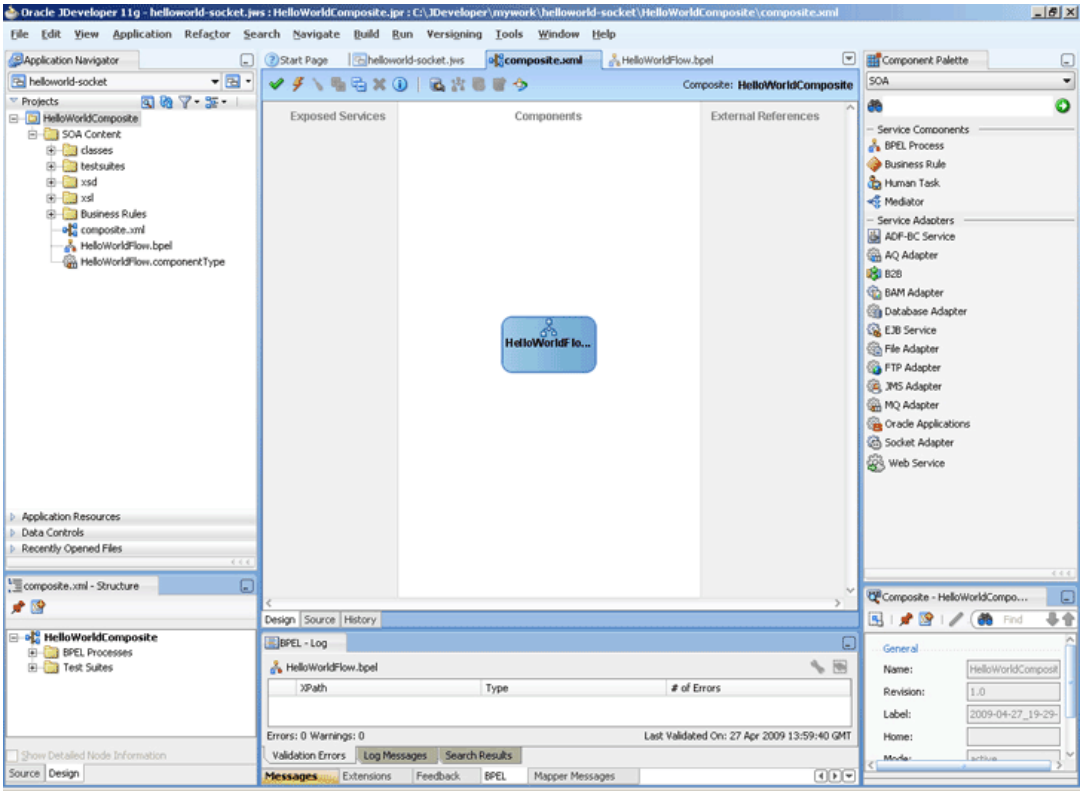
7. Enter `HelloWorldFlow` in the **Name** field and select **Define Service Later** from the Template box, as shown in Figure 5-44.

Figure 5-44 The Create BPEL Process Dialog



8. Click **OK**. The `HelloWorld` application and the `HelloWorldComposite` project appear in the design area, as shown in Figure 5-45.

Figure 5-45 The JDeveloper - composite.xml



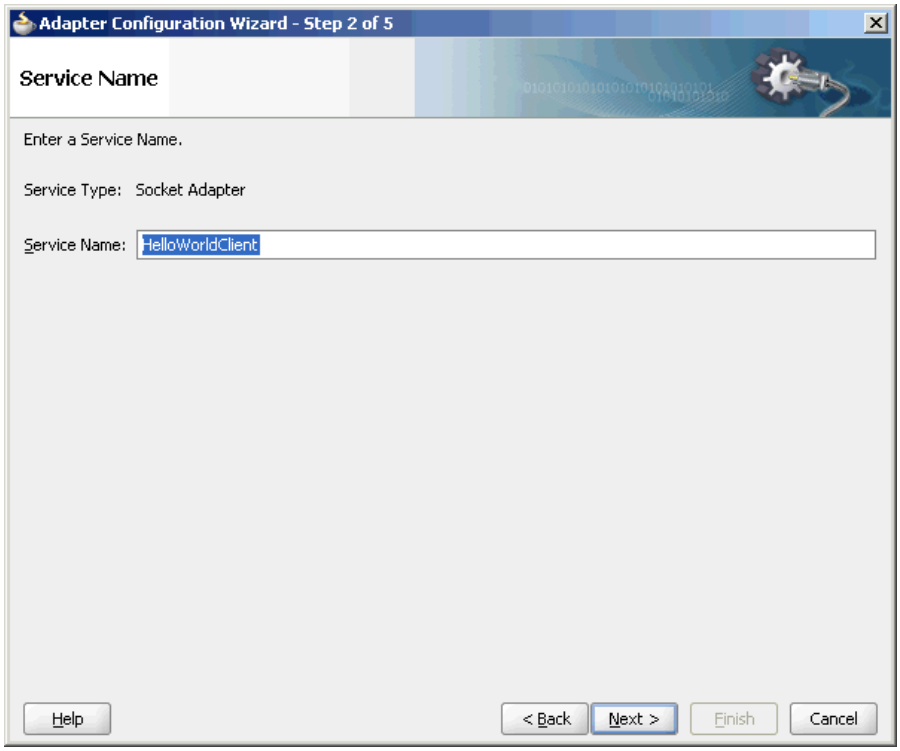
9. Copy the `HelloWorld.xsd` file to the xsd directory in your project (see Prerequisites for the location of this file).
10. Copy the `request.xsl`, `reply.xsl`, and `invoke.xsl` files to the xsl directory in your project (see Prerequisites for the location of these files).

Creating the Inbound Oracle Socket Adapter Service

Perform the following steps to create an inbound Oracle Socket Adapter service:

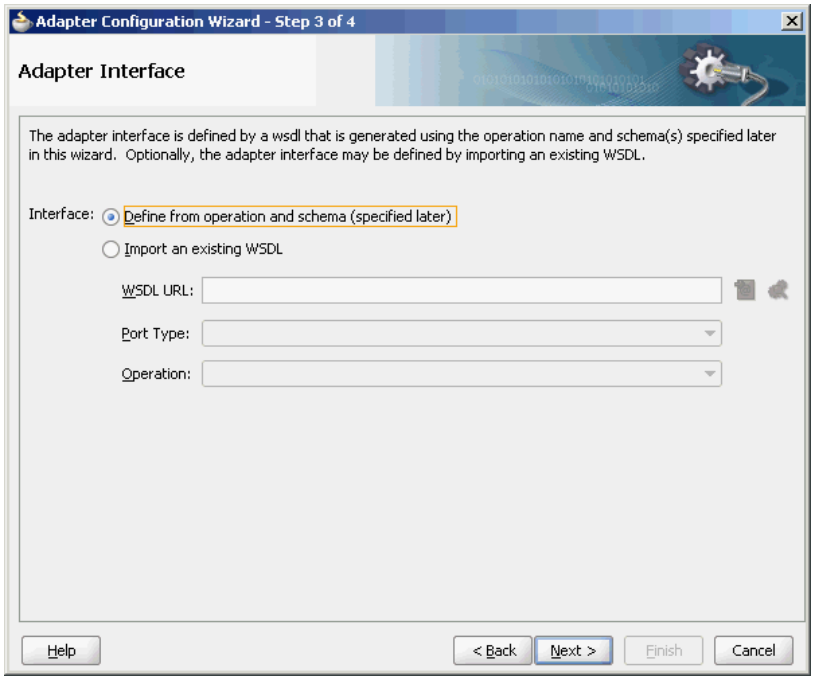
1. Drag and drop Socket Adapter from Components to the Exposed Services swim lane. The Welcome page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `HelloWorldClient` in the **Service Name** field, as shown in Figure 5-46.

Figure 5-46 The Adapter Configuration Wizard Service Name Page



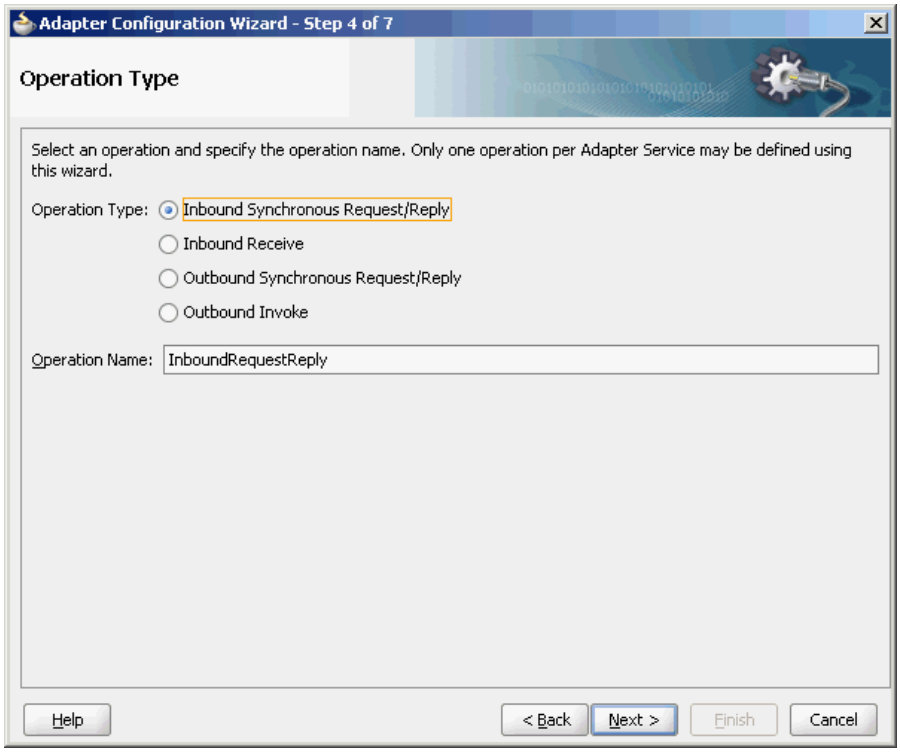
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, as shown in the Figure 5-47 and click **Next**. The **Operation** page is displayed.

Figure 5-47 The Adapter Configuration Wizard - Adapter Interface Page



6. Select **Inbound Synchronous Request/Reply** as the Operation Type, as shown in Figure 5-48.

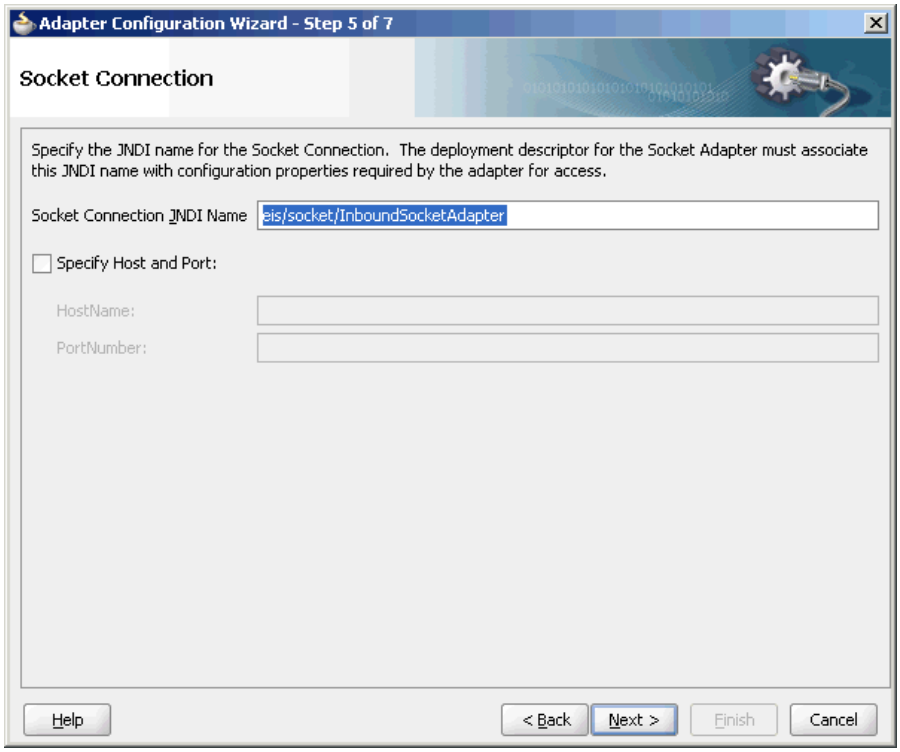
Figure 5-48 The Adapter Configuration Wizard Operation Page



7. Click **Next**. The **Socket Connection** page is displayed.

8. Enter `eis/socket/InboundSocketAdapter` in the **Socket Connection JNDI Name** field, as shown in Figure 5-49, and click **Next**. The **Messages** page is displayed.

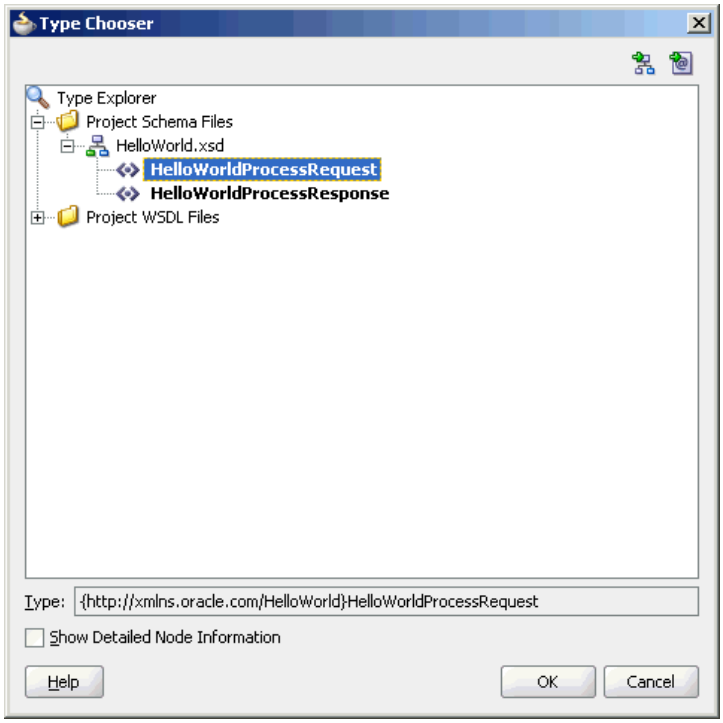
Figure 5-49 The Adapter Configuration Wizard Socket Connection Page



9. Click **Browse For Schema File** that appears at the end of the URL field in the Request Message Schema box. The **Type Chooser** dialog is displayed.

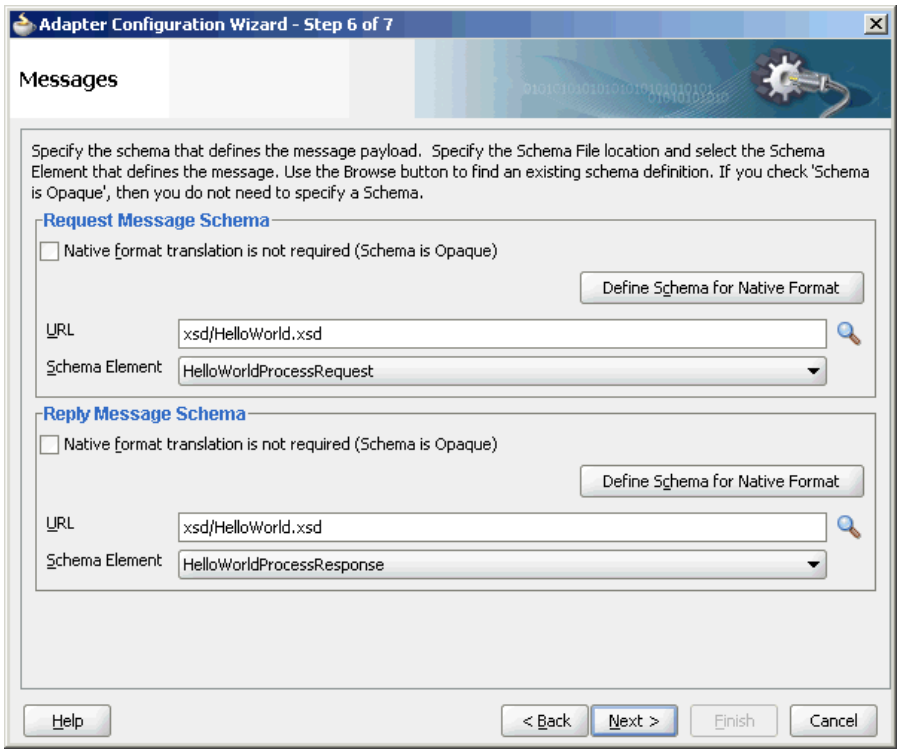
10. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessRequest**, as shown in Figure 5-50.

Figure 5-50 The Type Chooser Dialog



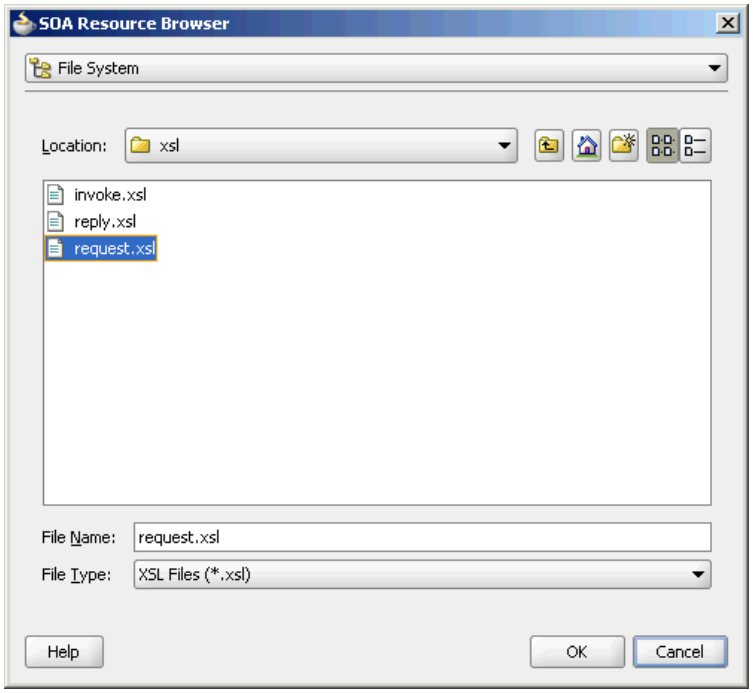
11. Click **OK**. The URL field in the **Messages** page is populated with the `HelloWorld.xsd` file.
12. Click **Browse For Schema File** that appears at the end of the URL field in the **Reply Message Schema** box. The **Type Chooser** dialog is displayed.
13. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessRequest**.
14. Click **OK**. The URL fields in the **Messages** page are populated with the `HelloWorld.xsd` files, as shown in Figure 5-51.

Figure 5-51 The Adapter Configuration Wizard File Messages Page



15. Click **Next**. The **Protocol** page is displayed.
16. Select **Use XSLT to define the handshake**.
17. Click **Browse to select the XSL file** that appears at the end of the Xslt field. The **SOA Resource Browser** dialog is displayed.
18. Select **request.xsl** as the file name, as shown in Figure 5-52, and click **OK**. The Xslt field is populated.

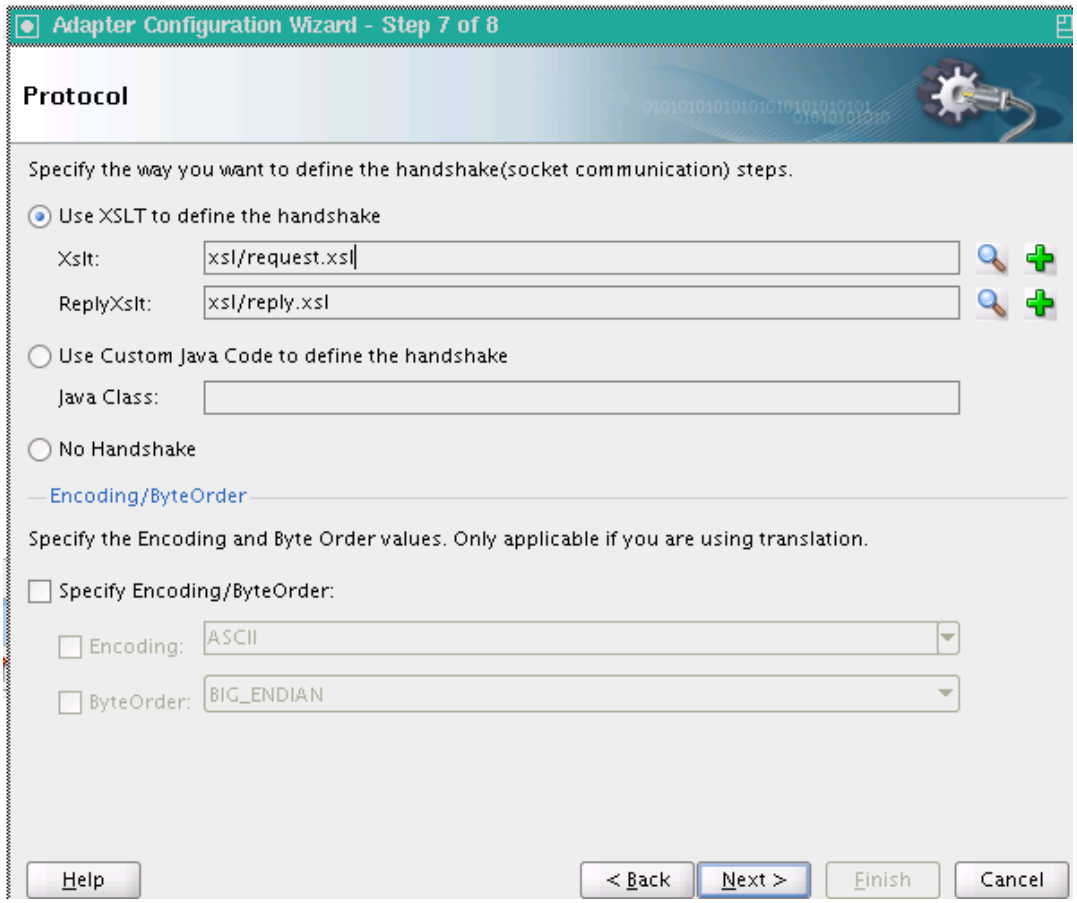
Figure 5-52 The SOA Resource Browser Dialog



19. Click **Browse to select the XSL file** that appears at the end of the ReplyXslt field. The SOA Resource Browser dialog is displayed.

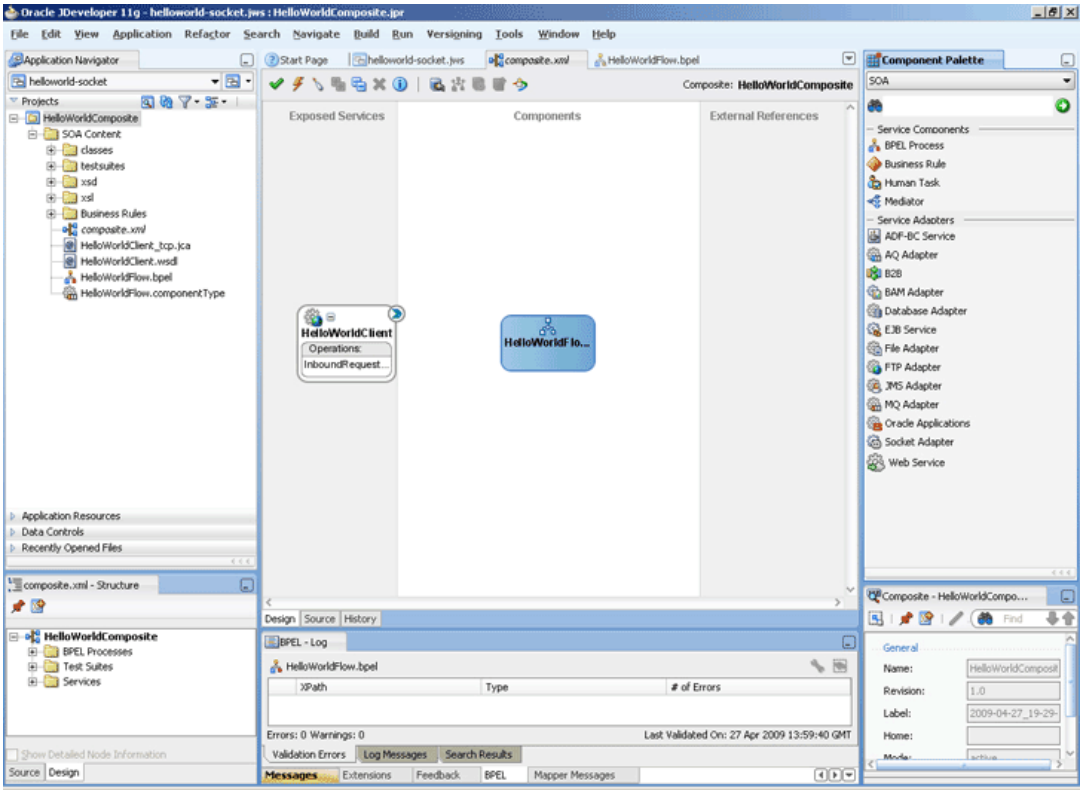
20. Select **reply.xsl** as the file name and click **OK**. The Xslt field is populated, as shown in Figure 5-53.

Figure 5-53 The Adapter Configuration Wizard Protocol Page



21. Click **Finish**. The composite.xml page appears, as shown in Figure 5-54.

Figure 5-54 The JDeveloper - composite.xml Page

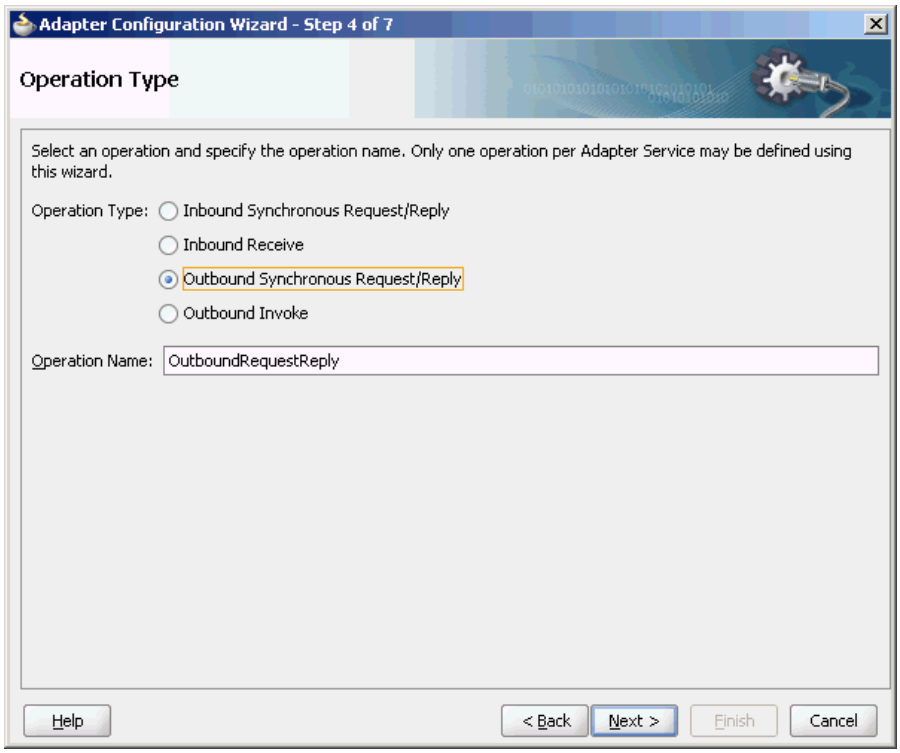


Creating the Outbound Oracle Socket Adapter Service

Perform the following steps to create an outbound Oracle Socket Adapter service:

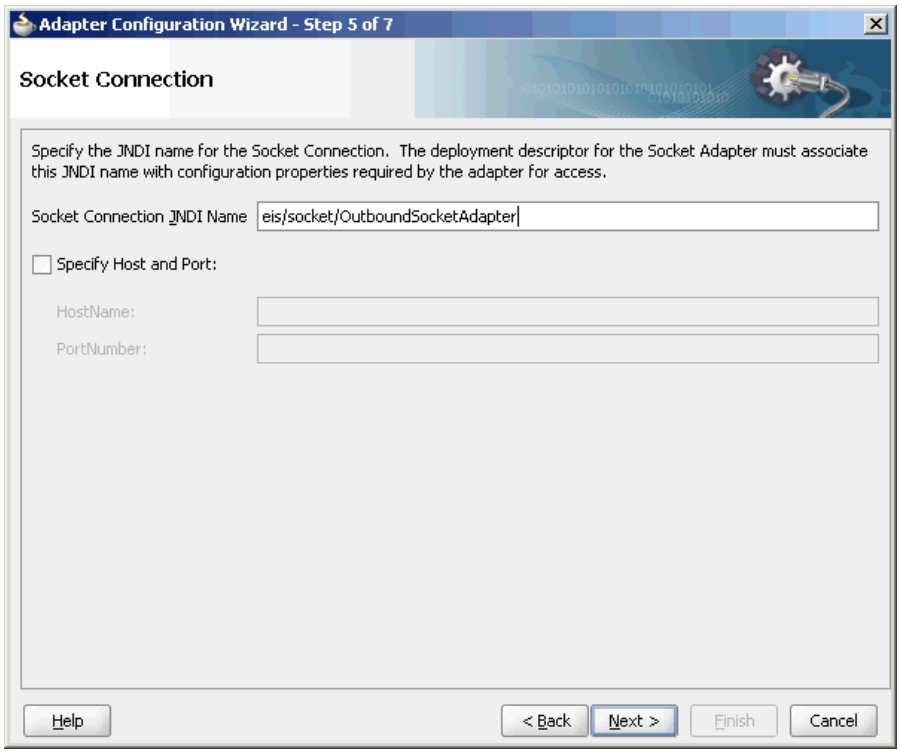
1. Drag and drop **Socket Adapter** from the Components window to the External References swim lane. The **Welcome** page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `HelloWorldServer` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The **Operation** page is displayed.
6. Select **Outbound Synchronous Request/Reply** as the operation type, as shown in Figure 5-55.

Figure 5-55 The Adapter Configuration Wizard Operation Type Page



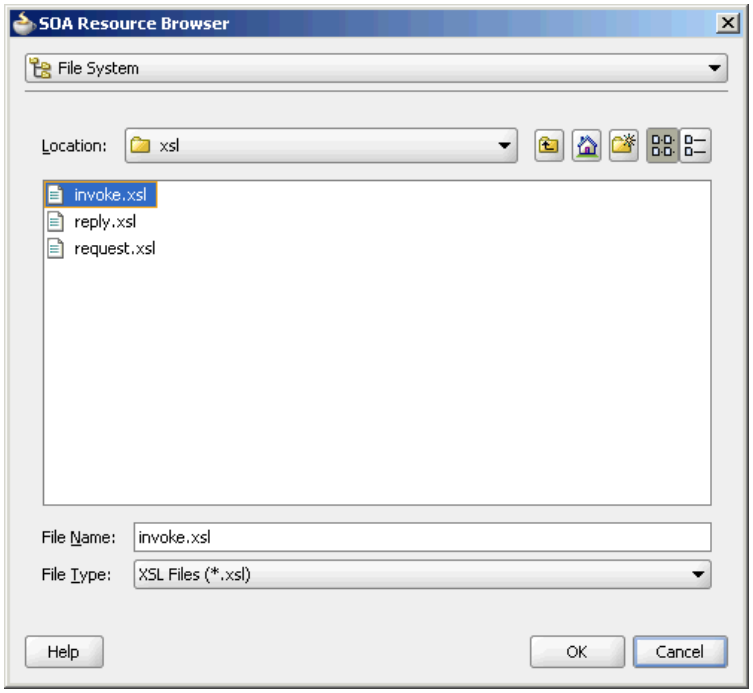
7. Click **Next**. The Socket Connection page is displayed.
8. Enter `eis/socket/OutboundSocketAdapter` in the **Socket Connection JNDI Name** field, as shown in Figure 5-56, and click **Next**. The **Messages** page is displayed.

Figure 5-56 The Adapter Configuration Wizard Socket Connection Page



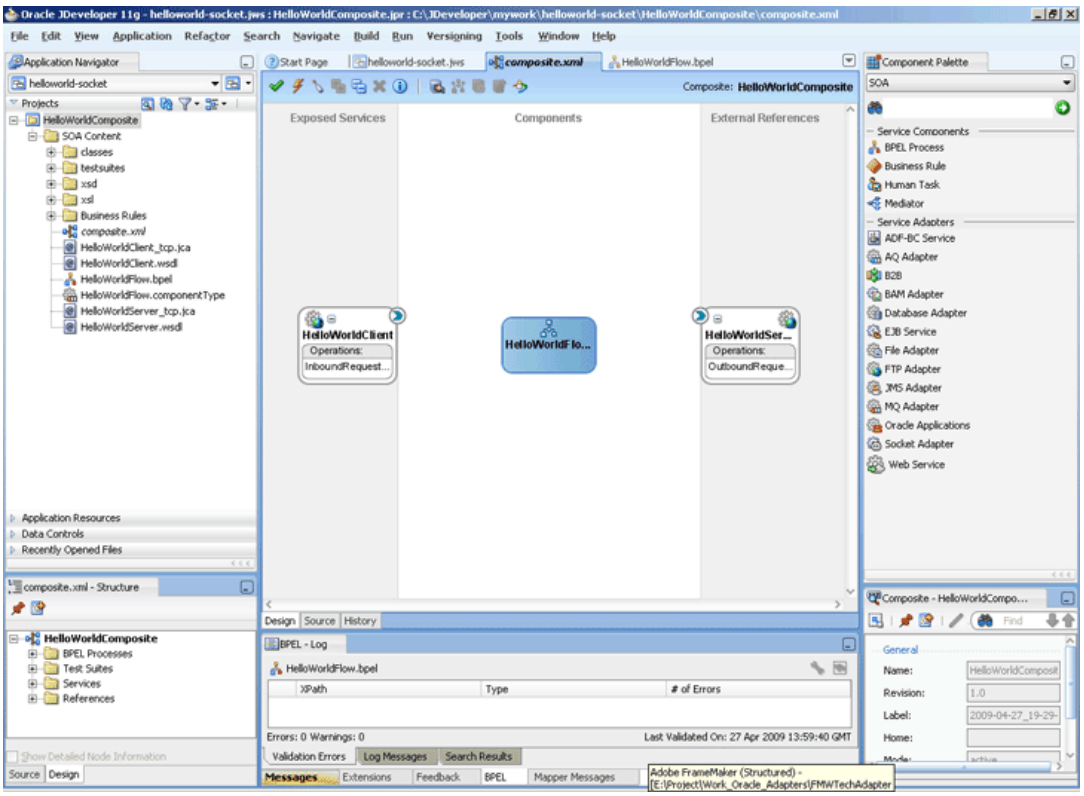
9. Click **Browse For Schema File** that appears at the end of the URL field in the **Request Message Schema** box. The **Type Chooser** dialog is displayed.
10. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessRequest**, as shown in Figure 5-50.
11. Click **OK**. The URL field in the **Messages** page is populated with the `HelloWorld.xsd` file.
12. Click **Browse For Schema File** that appears at the end of the URL field in the **Reply Message Schema** box. The **Type Chooser** dialog is displayed.
13. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessResponse**.
14. Click **OK**. The URL fields in the Messages page are populated with the `HelloWorld.xsd` files, as shown in Figure 5-51.
15. Click **Next**. The Protocol page is displayed.
16. Select **Use XSLT to define the handshake**.
17. Click **Browse to select the XSL file** that appears at the end of the Xslt field. The **SOA Resource Browser** dialog is displayed.
18. Select **invoke.xsl** as the file name, as shown in Figure 5-57, and click **OK**. The Xslt field is populated.

Figure 5-57 The SOA Resource Browser Dialog



19. Click **Finish**. The **composite.xml** page appears, as shown in Figure 5-58.

Figure 5-58 The JDeveloper - composite.xml Page



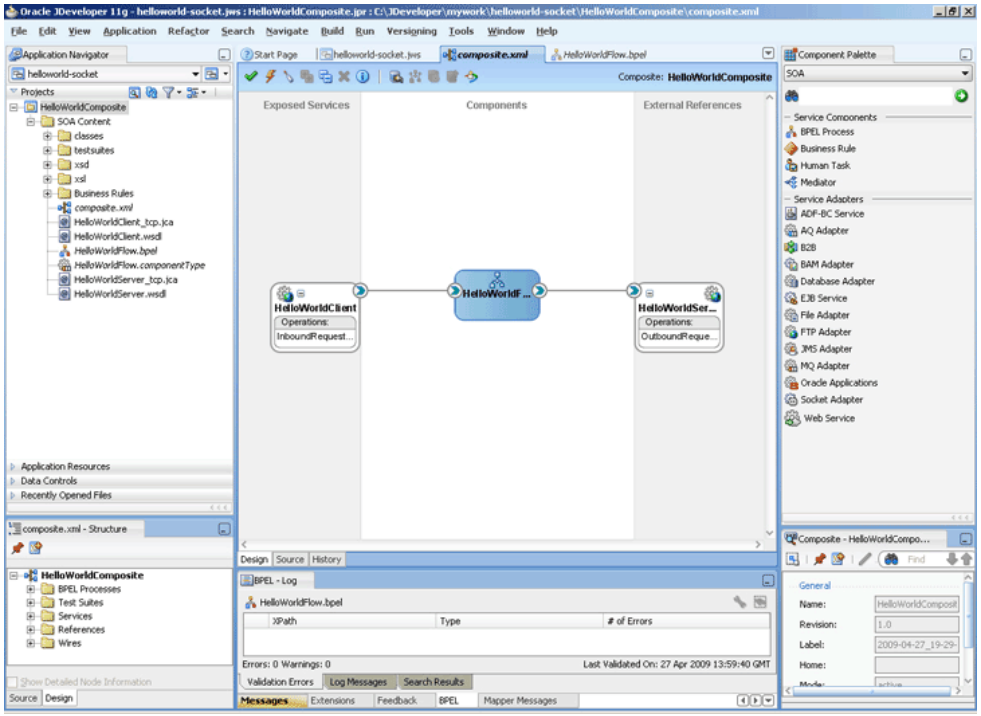
Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the HelloWorldClient in the Exposed Services area to the drop zone that appears as a green triangle in the HelloWorldFlow BPEL process in the **Components** area.
2. Drag the small triangle in the HelloWorldFlow BPEL process in the Components area to the drop zone that appears as a green triangle in the HelloWorldServer in the **External References** area.

   The JDeveloper composite.xml appears, as shown in Figure 5-59.

Figure 5-59 The JDeveloper - composite.xml

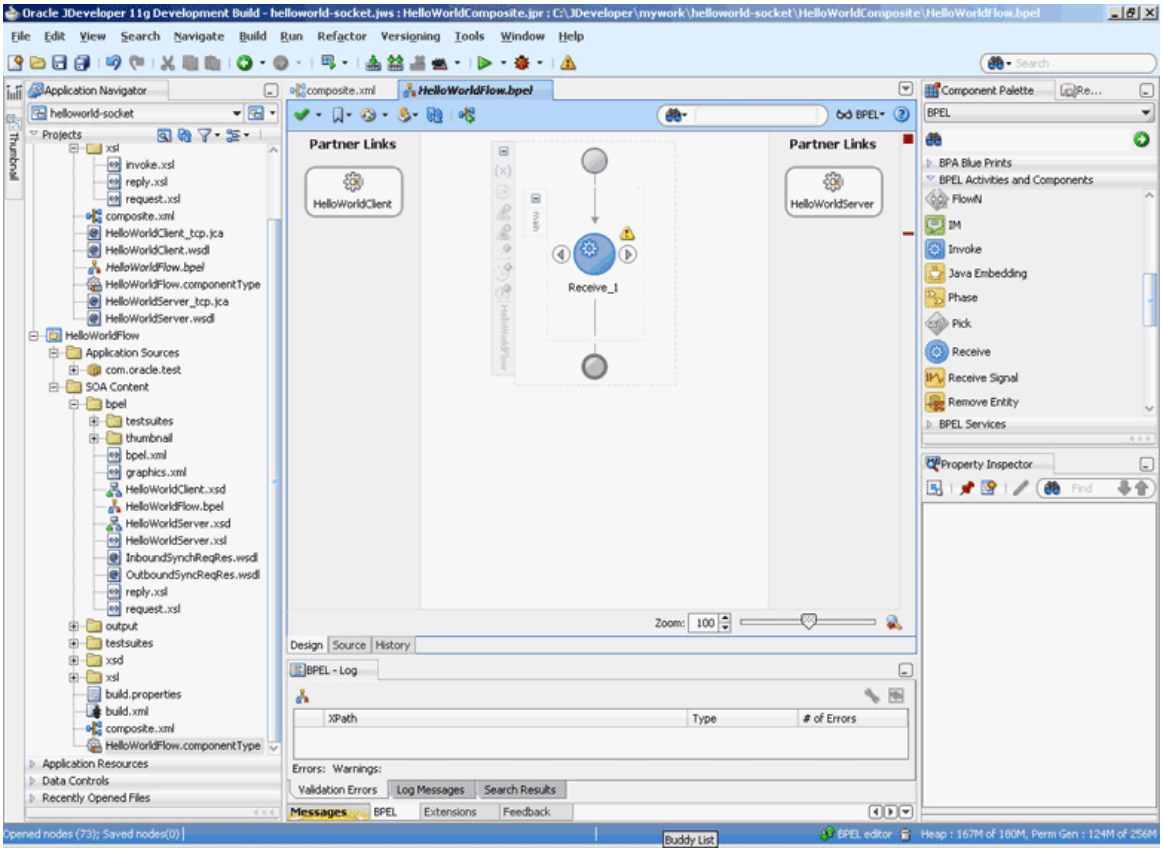


3. Click **File**, **Save All**.

Add a Receive Activity

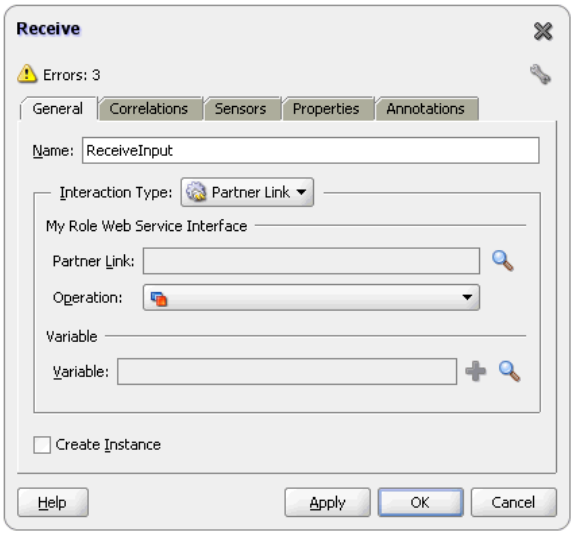
1. Double-click **HelloWorldFlow**. The **BPELHelloWorld.bpel** page is displayed.
2. Drag and drop a **Receive** activity from the Components window to the design area, as shown in Figure 5-60.

Figure 5-60 The JDeveloper - HelloWorldFlow.bpel



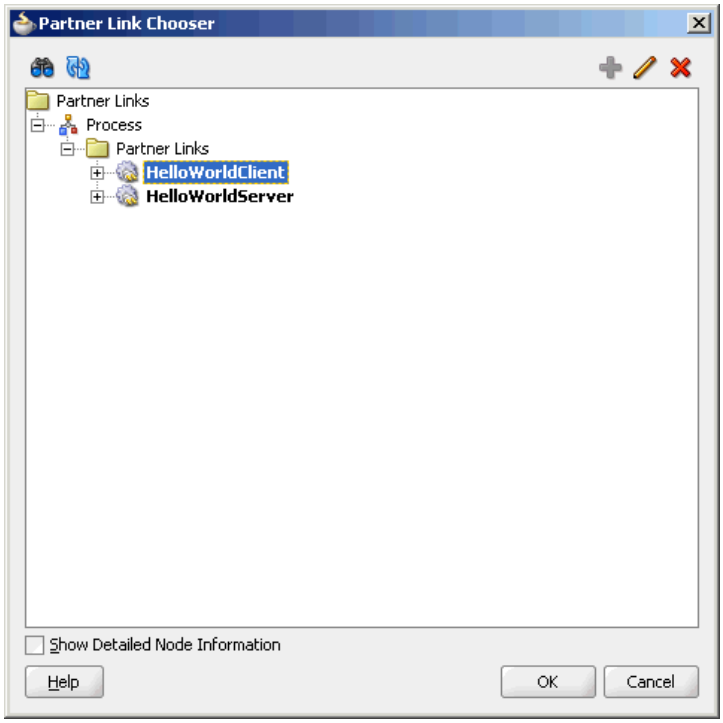
3. Double-click the **Receive** activity. The Receive dialog is displayed.
4. Enter `ReceiveInput` in the **Name** field, as shown in Figure 5-61.

Figure 5-61 The Receive Dialog



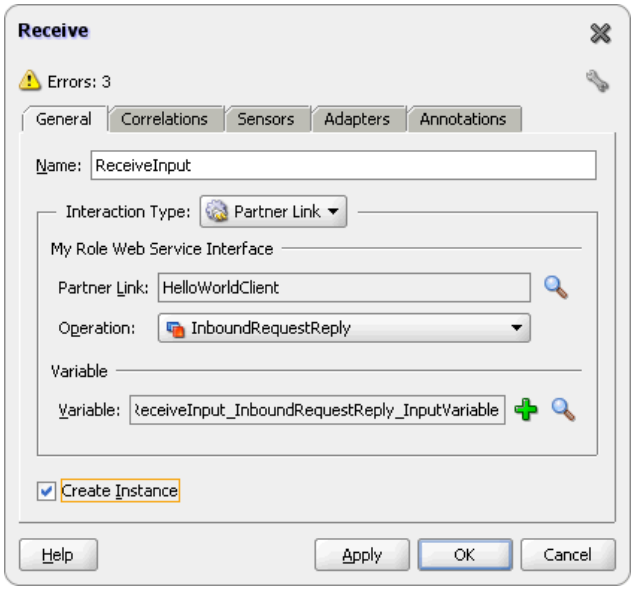
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
6. Select **HelloWorldClient**, as shown in Figure 5-62, and click **OK**.

Figure 5-62 The Partner Link Chooser Dialog



7. Click the **Auto-Create Variable** icon to the right of the Variable field in the **Receive** dialog, as shown in Figure 5-63. The **Create Variable** dialog is displayed.

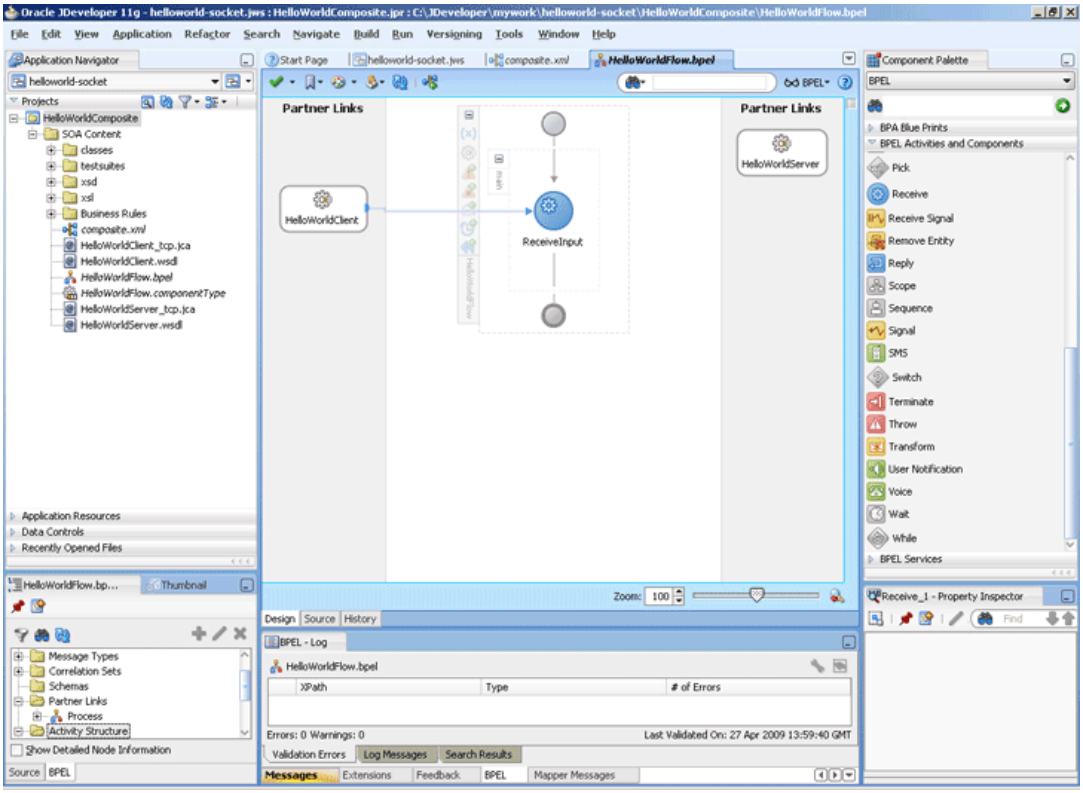
Figure 5-63 The Receive Dialog



8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.

9. Check **Create Instance**, and click **OK**. The **JDeveloper HelloWorldFlow.bpel** page appears, as shown in Figure 5-64.

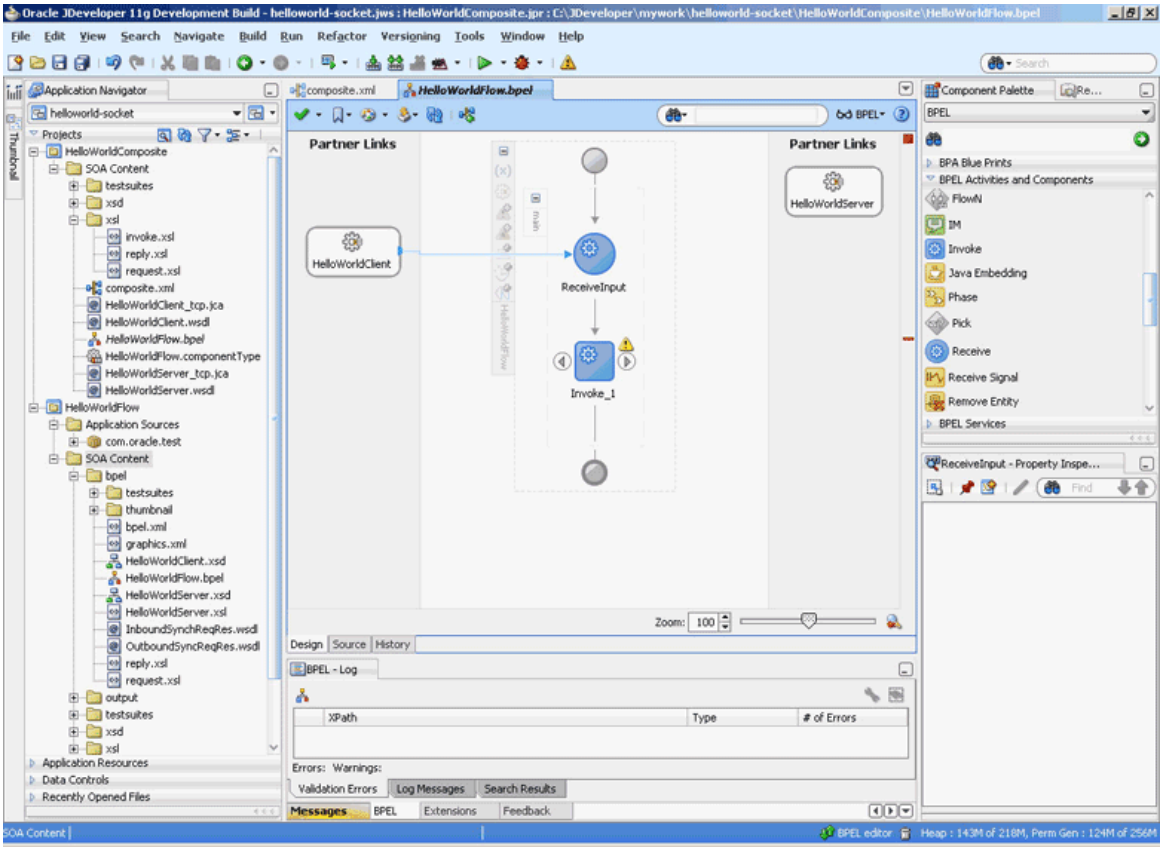
Figure 5-64 The JDeveloper - HelloWorldFlow.bpel



Add an Invoke Activity

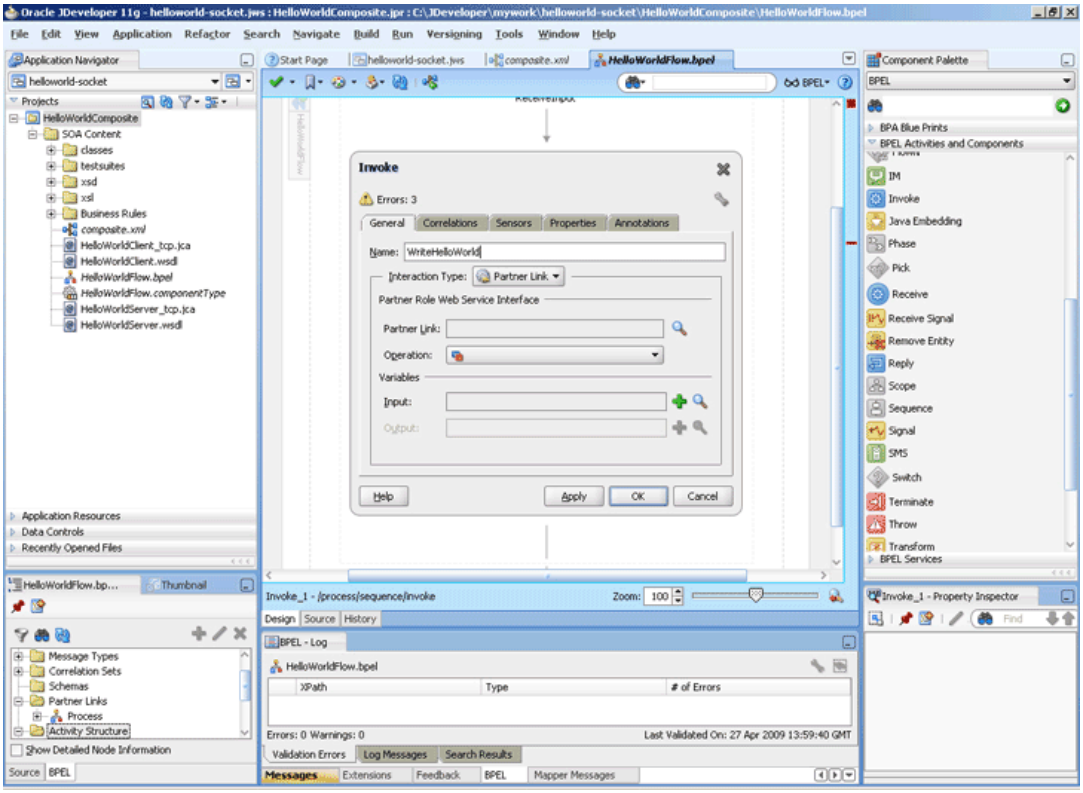
1. Drag and drop an **Invoke** activity after the ReceiveInput activity from the Components window to the design area, as shown in Figure 5-65.

Figure 5-65 The JDeveloper - HelloWorldFlow.bpel



2. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.
3. Enter `WriteHelloWorld` in the **Name** field, as shown in Figure 5-66.

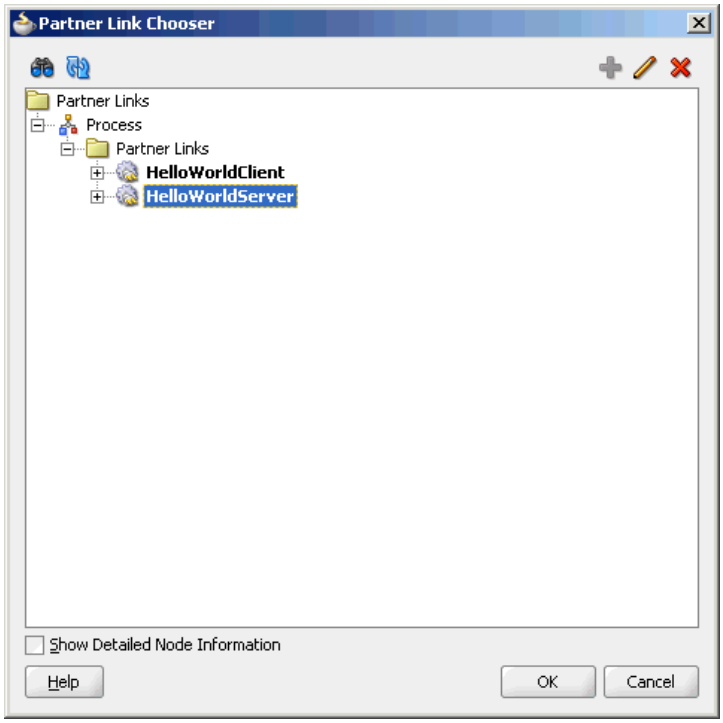
Figure 5-66 The JDeveloper - HelloWorldFlow.bpel



4. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.

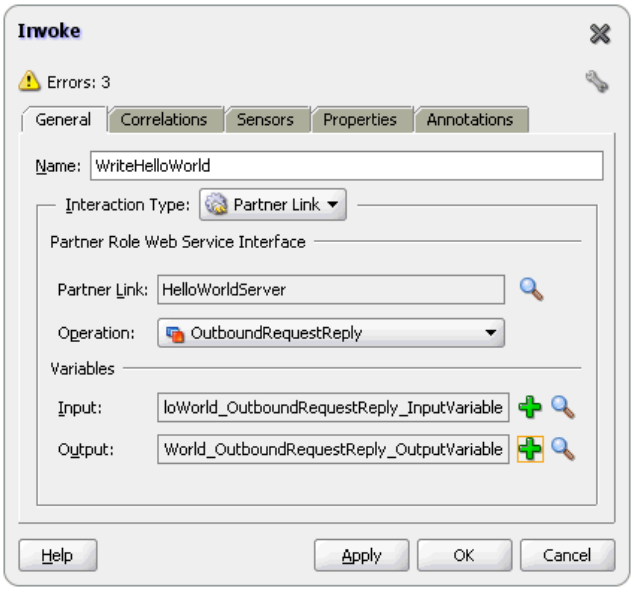
5. Select **HelloWorldServer**, as shown in Figure 5-67, and click **OK**.

Figure 5-67 The Partner Link Chooser Dialog



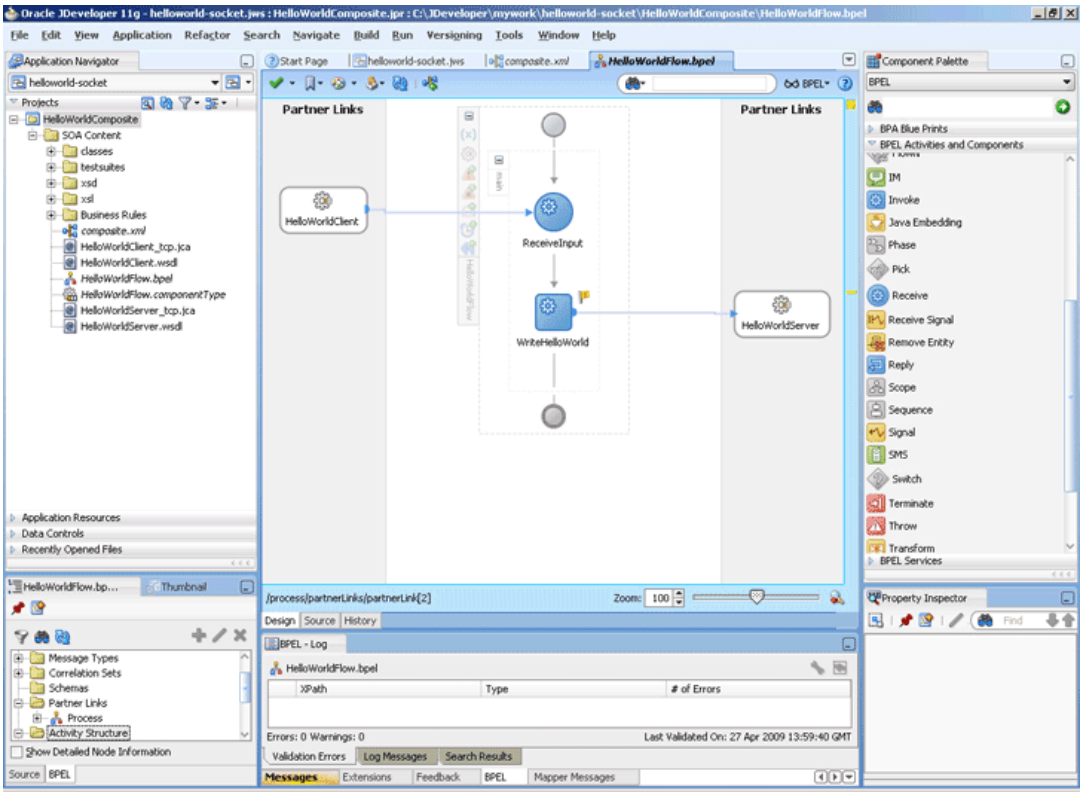
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
8. Repeat the same for selecting the output variable. The **Invoke** dialog is displayed, as shown in Figure 5-68.

Figure 5-68 The Invoke Dialog



9. Click **OK**. The **JDeveloper HelloWorldFlow.bpel** page appears, as shown in Figure 5-69.

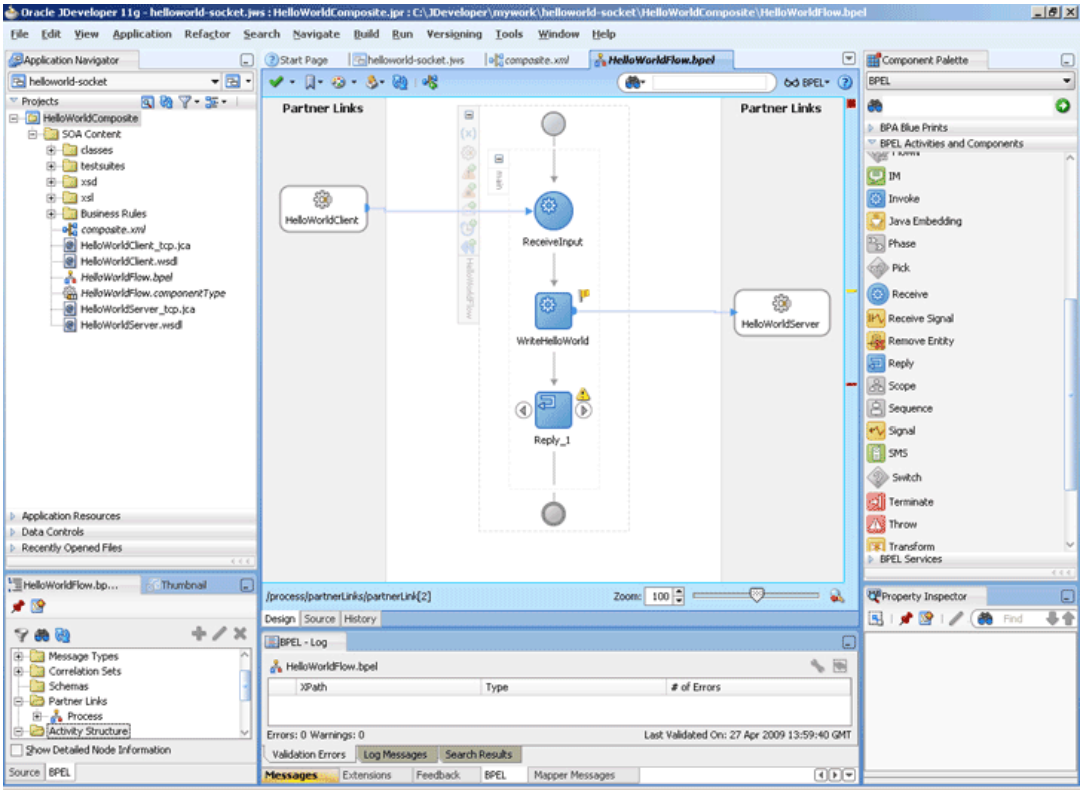
Figure 5-69 The JDeveloper - HelloWorldFlow.bpel



Add a Reply Activity

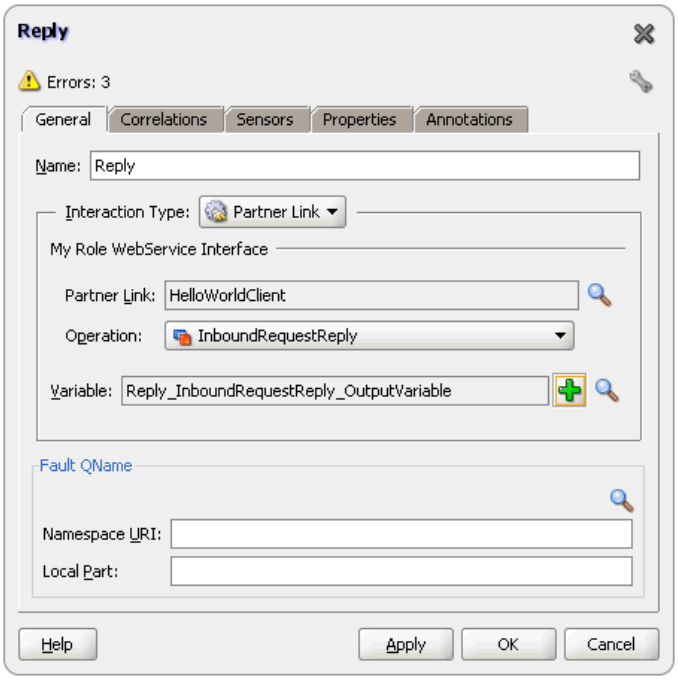
1. Drag and drop a **Reply** activity from the Components window to the design area, as shown in Figure 5-70.

Figure 5-70 The JDeveloper - HelloWorldFlow.bpel



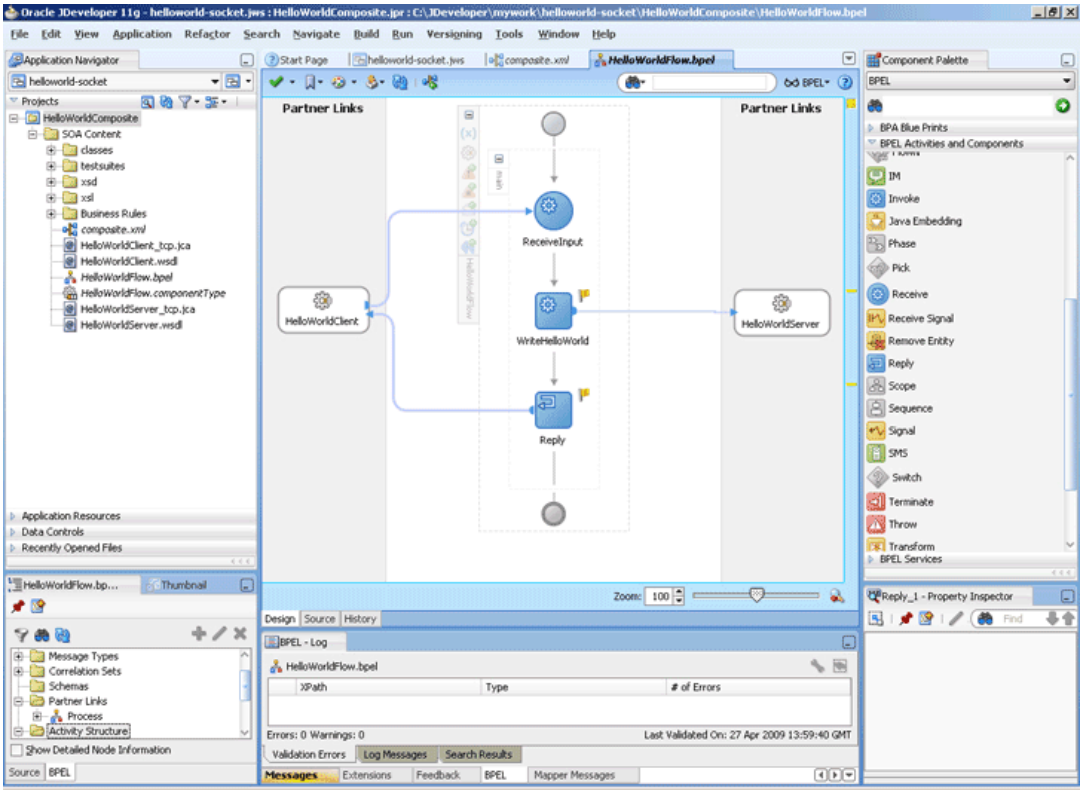
2. Double-click the **Reply** activity. The Reply dialog is displayed.
3. Enter `Reply` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **HelloWorldClient**, as shown in Figure 5-62, and click **OK**.
6. Click the **Auto-Create Variable** icon to the right of the Variable field in the Reply dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name, as shown in Figure 5-71.

Figure 5-71 The Reply Dialog



8. Click **OK**. The JDeveloper **HelloWorldFlow.bpel** page appears, as shown in Figure 5-72.

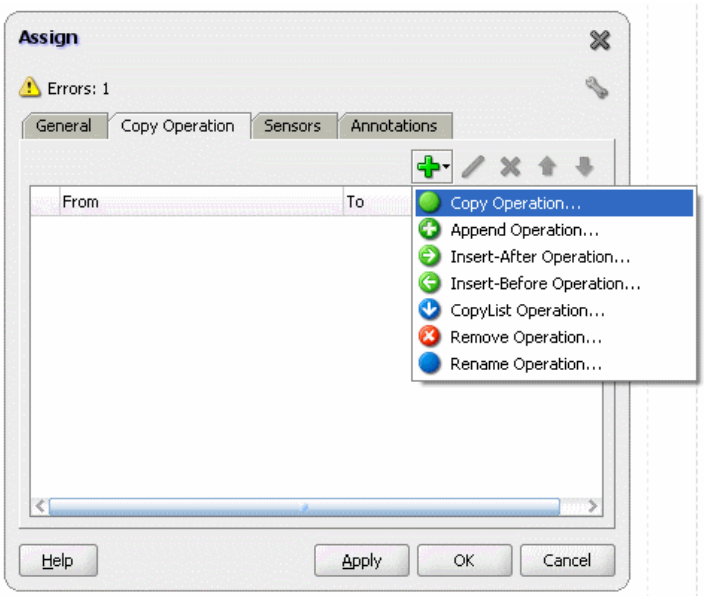
Figure 5-72 The JDeveloper - HelloWorldFlow.bpel



Add Assign Activities

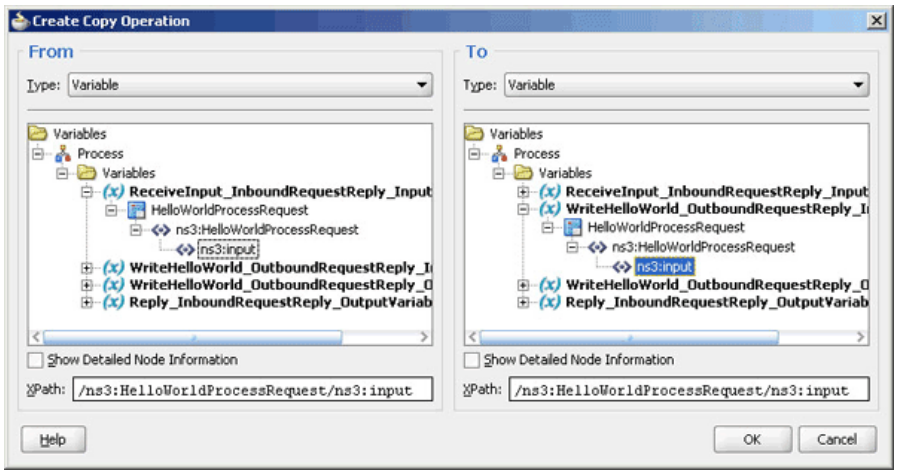
1. Drag and drop an **Assign** activity from the Components window in between the Receive and Invoke activities in the design area.
2. Double-click the **Assign** activity. The **Assign** dialog is displayed.
3. Click the **Copy Operation** tab. The **Assign** dialog is displayed, as shown in Figure 5-73.

Figure 5-73 The Assign Dialog - Copy Operation Tab



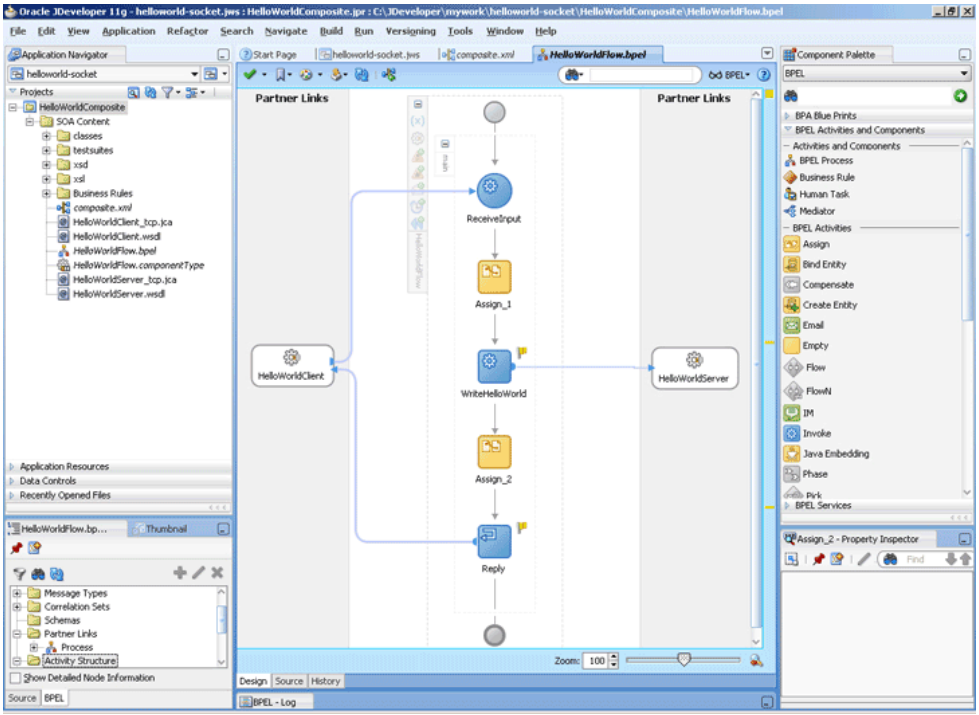
4. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
5. In the left pane, under the ReceiveInput_InboundRequestReply_InputVariable variable select, **ns3:input**.
6. In the right pane, under the WriteHelloWorld_OutboundRequestReply_InputVariable variable select, **ns3:input**, as shown in Figure 5-74.

Figure 5-74 The Create Copy Operation Dialog



7. Click **OK**. The **Assign** dialog is displayed.
8. Click **OK**. The JDeveloper **HelloWorldFlow.bpel** page is displayed.
9. Add another Assign activity in between the Invoke and the Reply activities.
10. Double-click the Assign activity.
11. Click the **Copy Operation** tab, and select **Copy Operation**.
12. In the left pane, select **ns3:result** under WriteHelloWorld_OutboundRequestReply_OutputVariable.
13. In the right pane, select **ns3:result** under Reply_InboundRequestReply_OutputVariable and click **OK**.
14. Click **OK**, the JDeveloper **HelloWorldFlow.bpel** page is displayed, as shown in Figure 5-75.

Figure 5-75 The JDeveloper - HelloWorldFlow.bpel



15. Click **File**, **Save All**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from .

Monitoring Using the Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console)

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the **Recent Instances** area.
3. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
4. Click the Instance ID that you noted in Step 2. The **Flow Trace** page is displayed.
5. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
6. Expand a payload node to view payload details.
7. Click the **Flow** tab to view the process flow. Additionally, click an activity (such as invoke, receive) to view the details of an activity.

Flight Information Display System

The flight information display system use case demonstrates the various modes of defining handshakes by using Oracle Socket Adapter.A flight information display server (FIDS) is started by an FIDS client requesting information on flight status for flights originating from a particular source, JFK, or SFO. The FIDS, in turn, invokes flight data requests for three airlines, Airline1, Airline 2, and Airline 3. The FIDS then collates the information received and replies to the FIDS client by using the HTTP protocol.

This use case includes the following sections:

- Prerequisites
- Designing the SOA Composite
- Creating the Inbound Service
- Creating Outbound Services
- Wiring Services and Activities
- Deploying withJDeveloper
- Monitoring Using the Fusion Middleware Control Console

Prerequisites

To perform this use case, you require the following files from the `artifacts.zip` file contained in the `Adapters-102SocketAdapterFlightInformationDisplaySystem` sample:

- `artifacts/schemas/Airline1.xsd`

- `artifacts/schemas/Airline2.xsd`
- `artifacts/schemas/Airline3.xsd`
- `artifacts/schemas/FIDS.xsd`
- `artifacts/xsl/request.xsl`
- `artifacts/xsl/reply.xsl`
- `artifacts/xsl/invoke.xsl`

To obtain the `Adapters-102SocketAdapterFlightInformationDisplaySystem` sample, access the Oracle SOA Sample Code site.

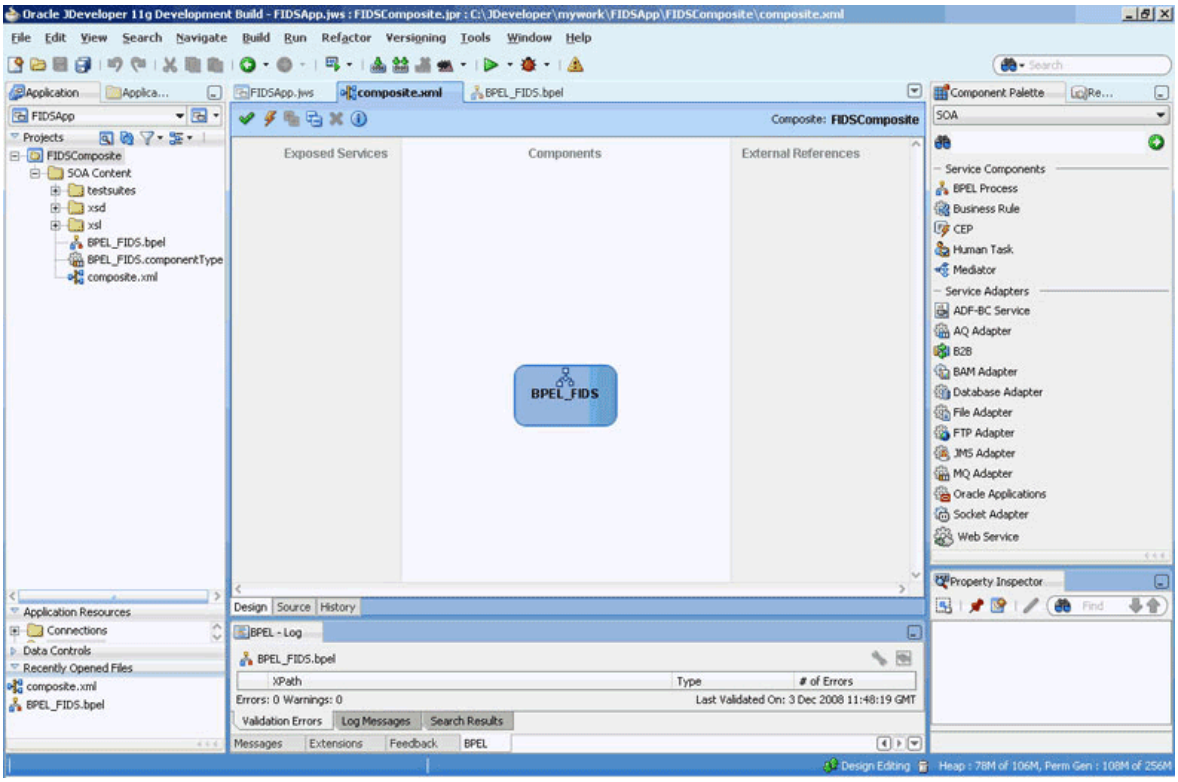
Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following steps:

1. In the Application Navigator of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `FIDSApp` in the **Application Name** field, and then click **Next**. The **Name Your Project** page is displayed.
3. Click **OK**. The **Name Your Project** dialog is displayed.
4. Enter `FIDSComposite` in the **Project Name** field, and then select **SOA** under Project Technologies and move it to the **Selected** box by clicking the right-arrow.

   The FIDSApp application and the FIDSComposite project appear in the Application Navigator.
5. Click **Next**. The **Configure SOA Settings** dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process dialog is displayed.
7. Enter `BPEL_FIDS` in the **Name** field and select **Define Service Later** from the Template box.
8. Click **OK**. The FIDSApp application and the FIDSComposite project appear in the design area, as shown in Figure 5-76.

Figure 5-76 The JDeveloper - composite.xml



9. Copy the `Airline1.xsd`, `Airline2.xsd`, `Airline3.xsd`, and `FIDS.xsd` files to `FIDSComposite\xsd` under the project `FIDSComposite` (see Prerequisites for the location of these files).

10. Copy `invoke.xsl`, `request.xsl`, and `reply.xsl` to `FIDSComposite\xsl` under the project `FIDSComposite` (see Prerequisites for the location of these files).

Creating the Inbound Oracle Socket Adapter Service

Perform the following steps to create an inbound Oracle Socket Adapter service that would be used to expose the FIDSApp application:

1. Drag and drop **Socket Adapter** from the Components to the Exposed Services swim lane. The **Welcome** page of the Adapter Configuration Wizard is displayed.

2. Click **Next**. The **Service Name** page is displayed.

3. Enter `FIDS` in the **Service Name** field.

4. Click **Next**. The **Adapter Interface** page is displayed.

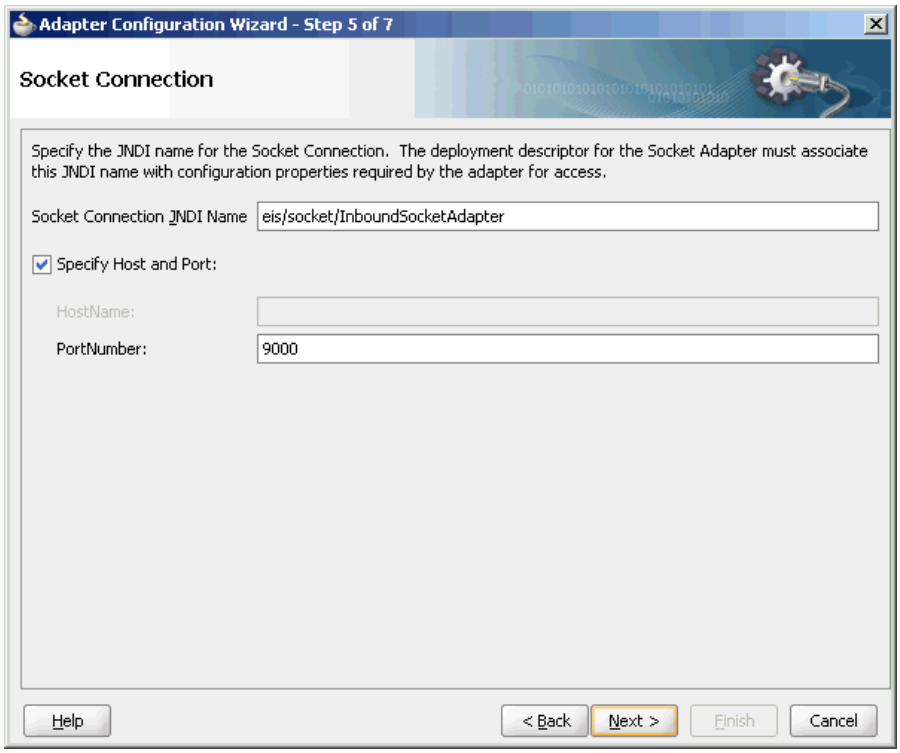
5. Select **Define from operation and schema (specified later)**, and click **Next**. The **Operation** page is displayed.

6. Select **Inbound Synchronous Request/Reply** as the operation type.

7. Click **Next**. The **Socket Connection** page is displayed.

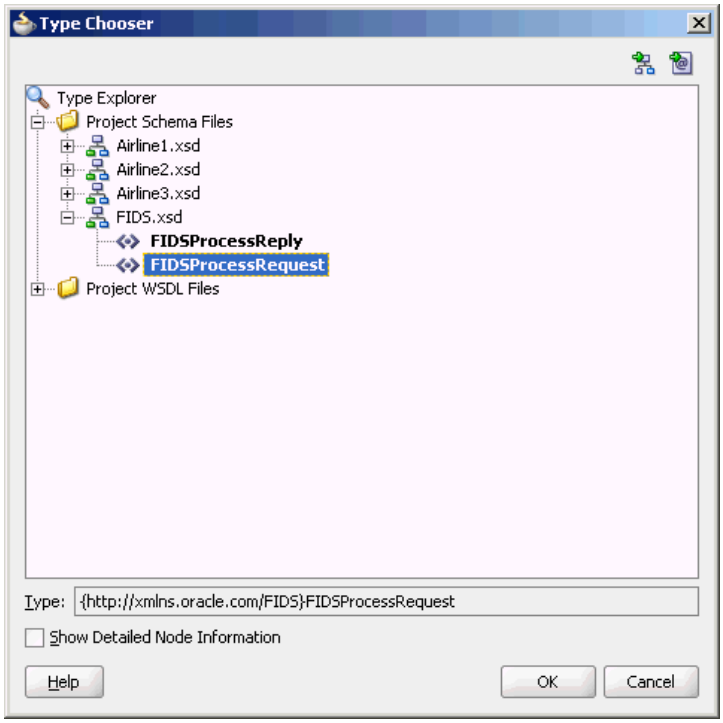
8. Enter `eis/socket/InboundSocketAdapter` in the **Socket Connection JNDI Name** field and then select **Specify Host and Port**, as shown in Figure 5-77.

Figure 5-77 The Adapter Configuration Wizard Socket Connection Page



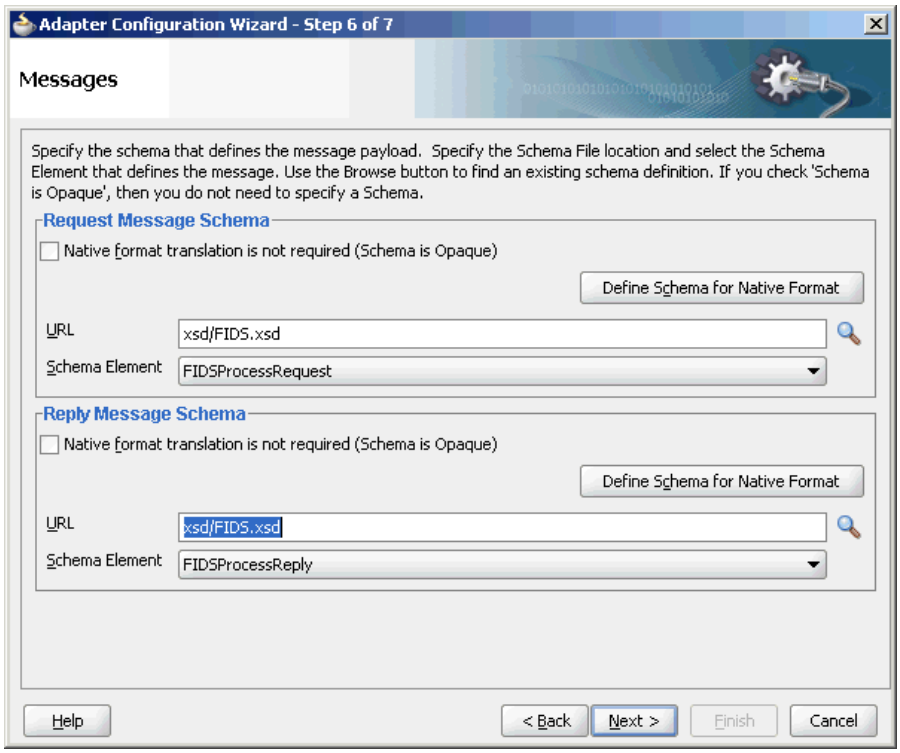
9. Enter `9000` in the PortNumber field and click **Next**. The **Messages** page is displayed.
10. Click **Browse For Schema File** that appears at the end of the URL field in the **Request Message Schema** box. The **Type Chooser** dialog is displayed.
11. Click **Project Schema Files**, **FIDS.xsd**, and **FIDSProcessRequest**, as shown in Figure 5-78.

Figure 5-78 The Type Chooser Dialog



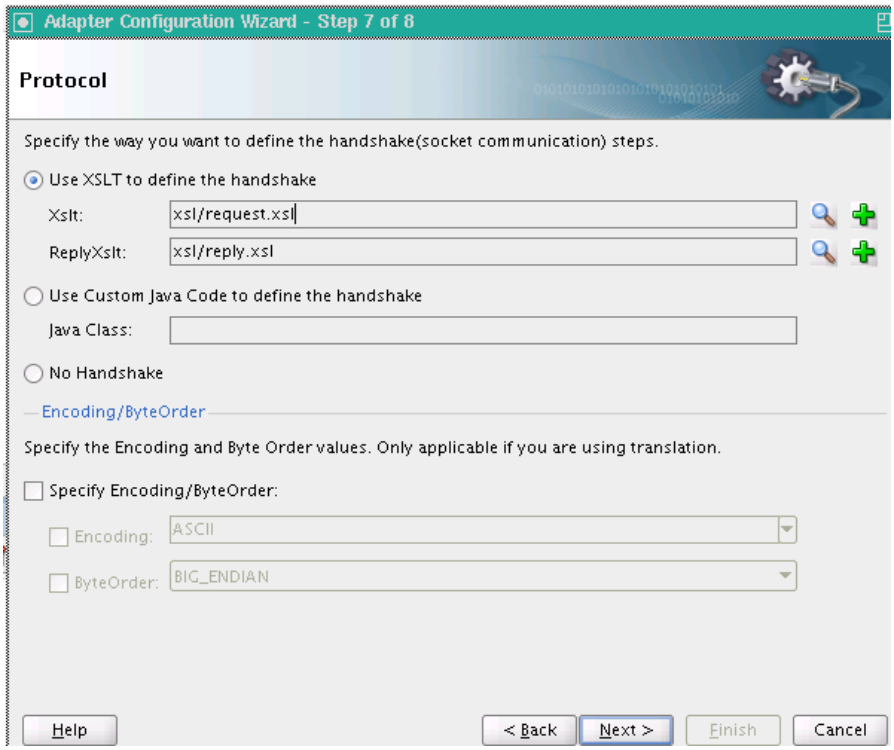
12. Click **OK**. The URL field in the **Messages** page is populated with the FIDS.xsd file.
13. Click **Browse For Schema File** that appears at the end of the URL field in the **Reply Message Schema** box. The Type Chooser dialog is displayed.
14. Click **Project Schema Files**, **FIDS.xsd**, and **FIDSProcessReply**.
15. Click **OK**. The URL fields in the **Messages** page are populated with the FIDS.xsd files, as shown in Figure 5-79.

Figure 5-79 The Adapter Configuration Wizard - Messages Page



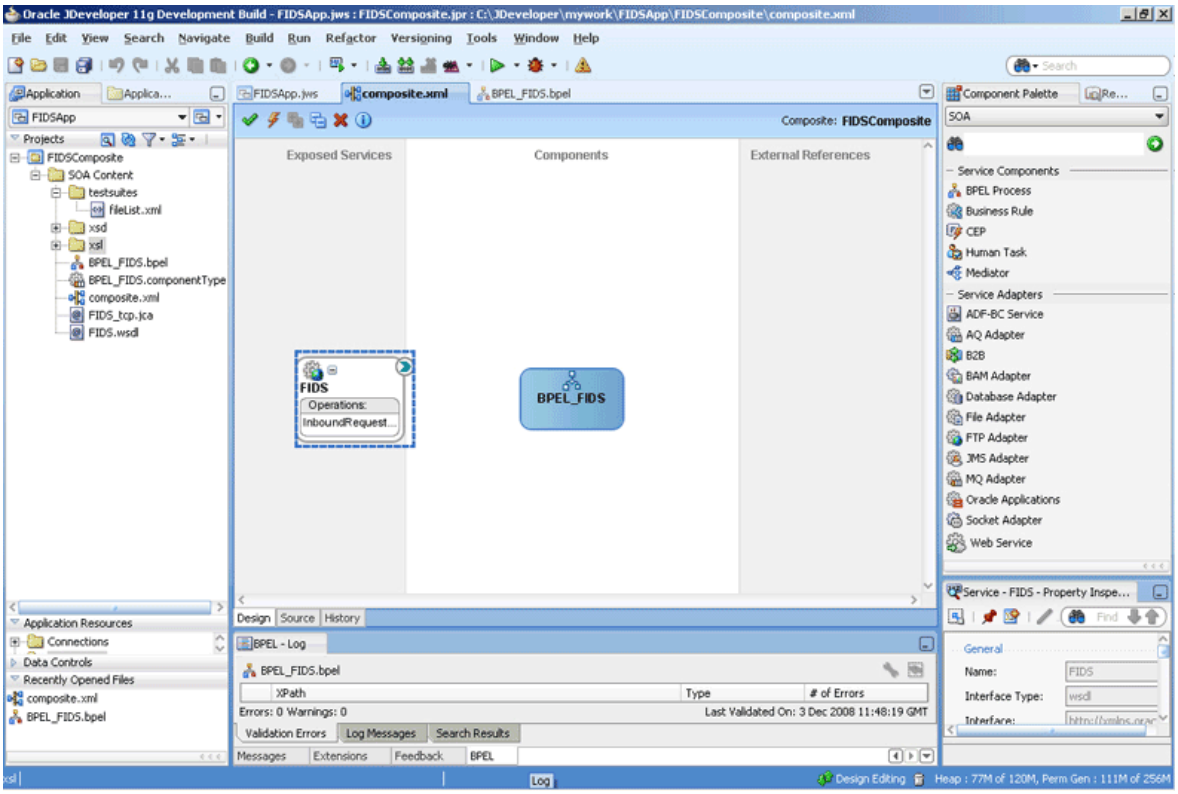
16. Click **Next**. The Protocol page is displayed.
17. Select **Use XSLT to define the handshake**.
18. Click the **Browse to select the XSL file** icon that appears at the end of the Xslt field. The **SOA Resource Browser** dialog is displayed.
19. Select **request.xsl** as the file name and click **OK**. The Xslt field is populated.
20. Click the **Browse to select the XSL file** icon that appears at the end of the ReplyXslt field. The **SOA Resource Browser** dialog is displayed.
21. Select **reply.xsl** as the file name and click **OK**. The Xslt field is populated, as shown in Figure 5-80.

Figure 5-80 The Adapter Configuration Wizard - Protocol Page



22. Click **Finish**. The composite.xml page appears, as shown in Figure 5-81.

Figure 5-81 The JDeveloper - composite.xml Page



Creating Outbound Oracle Socket Adapter Services

Perform the following steps to create an outbound Oracle Socket Adapter service for the Airline1 server socket:

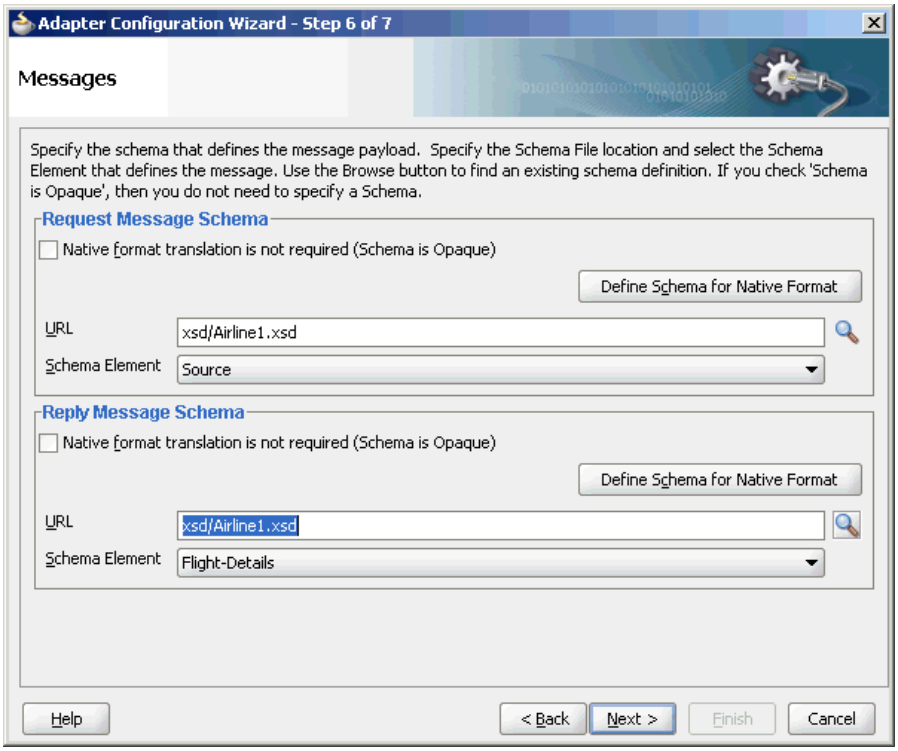
1. Drag and drop Socket Adapter from the Components window to the External References swim lane. The **Welcome** page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `Airline1` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Outbound Synchronous Request/Reply** as the Operation Type.
7. Click **Next**. The **Socket Connection** page is displayed.
8. Enter `eis/socket/OutboundSocketAdapter` in the **Socket Connection JNDI Name** field, as shown in Figure 5-82, and then select **Specify Host and Port**.

Figure 5-82 The Adapter Configuration Wizard - Socket Connection Page

9. Enter the name of the system where the Airline1 socket server program must run in the **HostName** field and `9001` in the **PortNumber** field, and click **Next**. The **Messages** page is displayed.
10. Click **Browse For Schema File** that appears at the end of the URL field in the **Request Message Schema** box. The Type Chooser dialog is displayed.

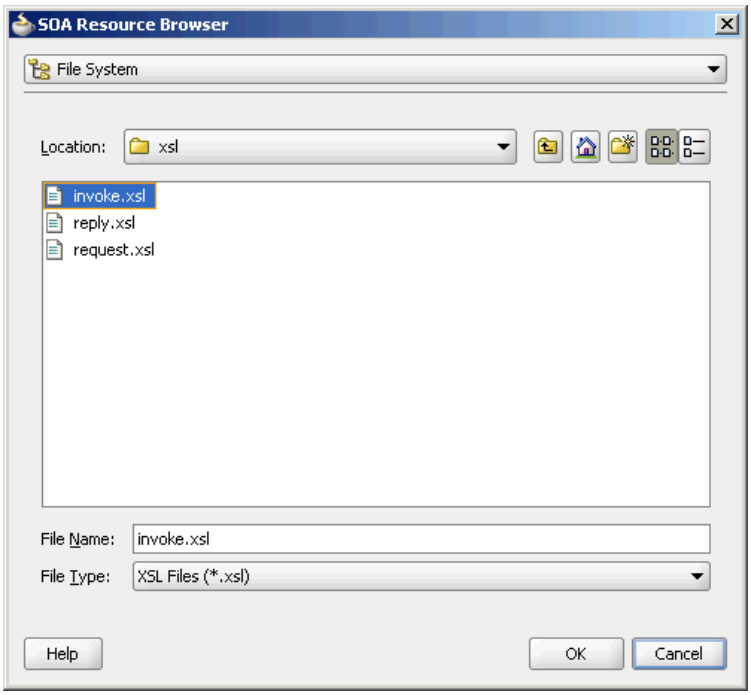
11. Click **Project Schema Files**, **Airline1.xsd**, and **Source**.
12. Click **OK**. The URL field in the **Messages** page is populated with the **Airline1.xsd** file.
13. Click **Browse For Schema File** that appears at the end of the URL field in the **Reply Message Schema** box. The **Type Chooser** dialog is displayed.
14. Click **Project Schema Files**, **Airline1.xsd**, and **Flight-Details**.
15. Click **OK**. The URL fields in the **Messages** page are populated with the `Airline1.xsd` files, as shown in Figure 5-83.

Figure 5-83 The Adapter Configuration Wizard - Messages Page



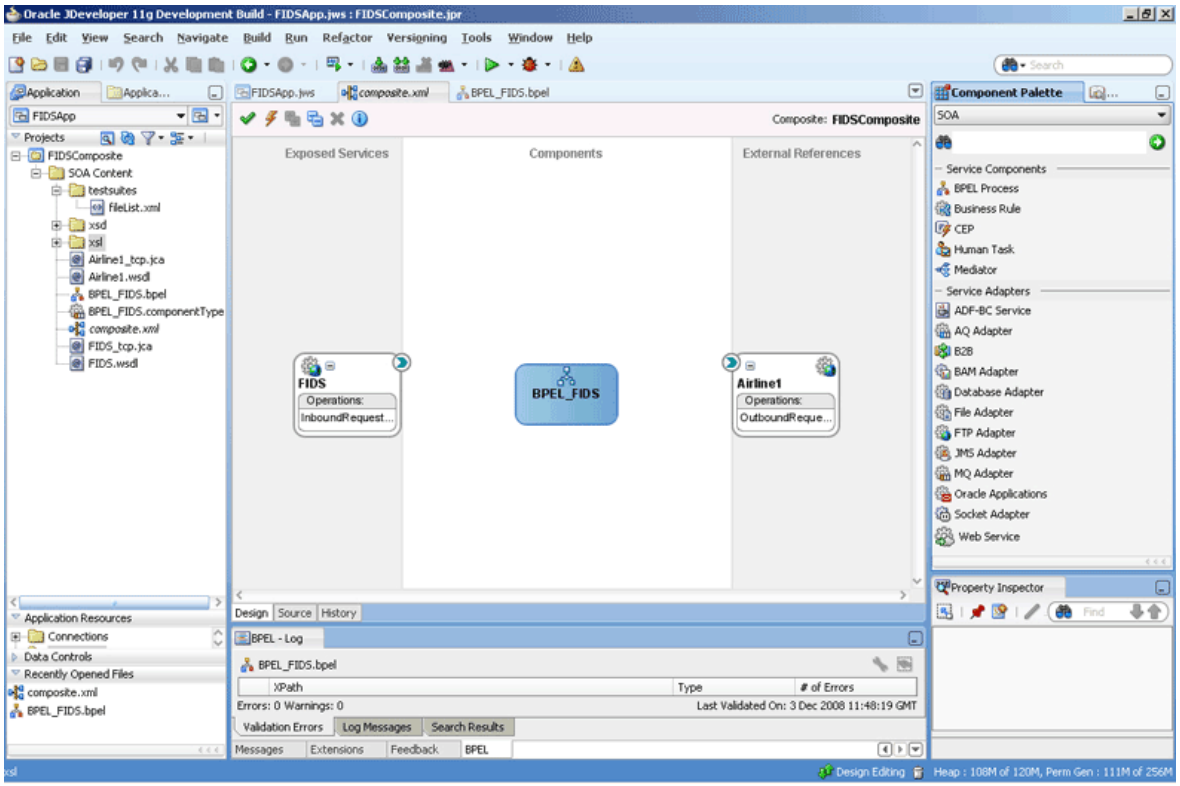
16. Click **Next**. The **Protocol** page is displayed.
17. Select **Use XSLT to define the handshake**.
18. Click **Browse to select the XSL file** that appears at the end of the Xslt field. The **SOA Resource Browser** dialog is displayed.
19. Select **invoke.xsl** as the file name, as shown in Figure 5-84, and click **OK**. The Xslt field is populated.

Figure 5-84 The SOA Resource Browser Dialog



20. Click **Finish**. The **composite.xml** page appears, as shown in Figure 5-85.

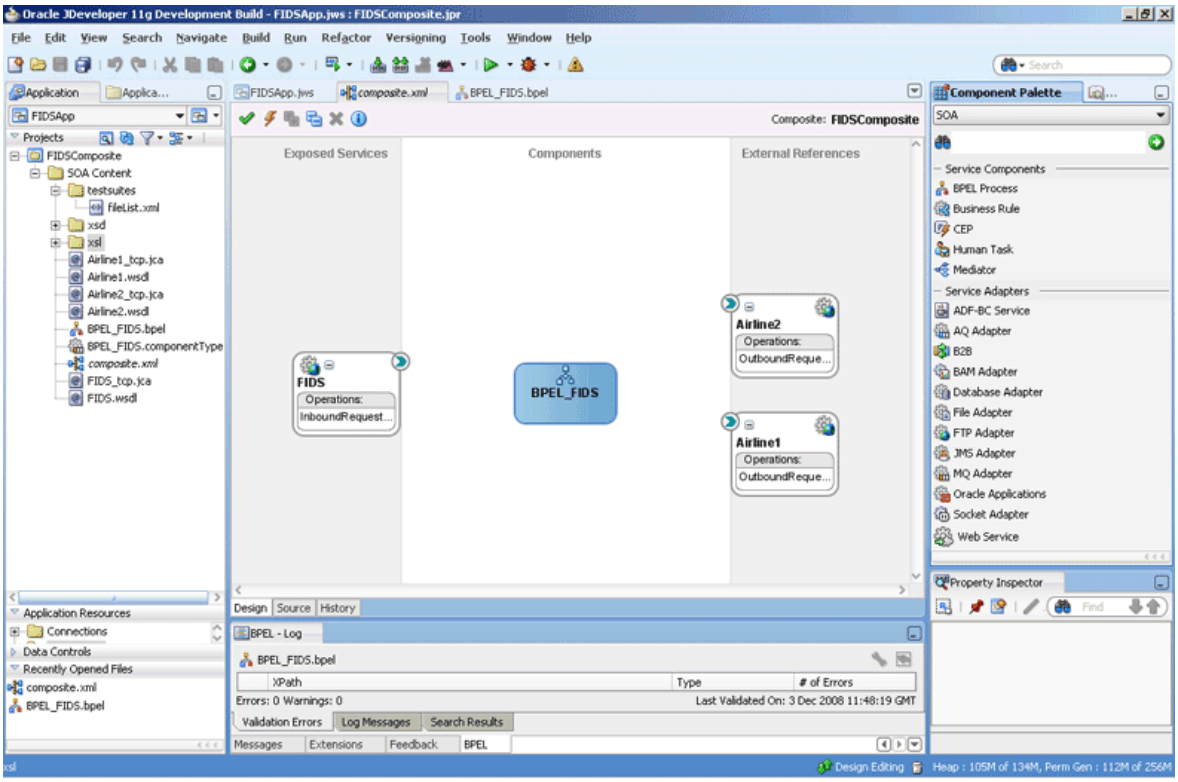
Figure 5-85 The JDeveloper - composite.xml Page



Perform the following steps to create an outbound Oracle Socket Adapter service for the Airline2 server socket:

1. Drag and drop **Socket Adapter** from the Components window to the External References swim lane. The **Welcome** page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `Airline2` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The **Operation** page is displayed.
6. Select **Outbound Synchronous Request/Reply** as the operation type.
7. Click **Next**. The Socket Connection page is displayed.
8. Enter `eis/socket/OutboundSocketAdapter` in the **Socket Connection JNDI Name** field and then select **Specify Host and Port**.
9. Enter the name of the system where the Airline2 socket server program must run in the **HostName** field and `9002` in the **PortNumber** field, and click **Next**. The Messages page is displayed.
10. Click **Browse For Schema File** that appears at the end of the URL field in the Request Message Schema box. The Type Chooser dialog is displayed.
11. Click **Project Schema Files**, **Airline2.xsd**, and **Source**.
12. Click **OK**. The URL field in the Messages page is populated with the `Airline2.xsd` file.
13. Click **Browse For Schema File** that appears at the end of the URL field in the Reply Message Schema box. The **Type Chooser** dialog is displayed.
14. Click **Project Schema Files**, **Airline2.xsd**, and **flight-details**.
15. Click **OK**. The URL fields in the **Messages** page are populated with the `Airline2.xsd` files.
16. Click **Next**. The **Protocol** page is displayed.
17. Select **No Handshake**.
18. Click **Finish**. The **composite.xml** page appears, as shown in Figure 5-86.

Figure 5-86 The JDeveloper - composite.xml Page

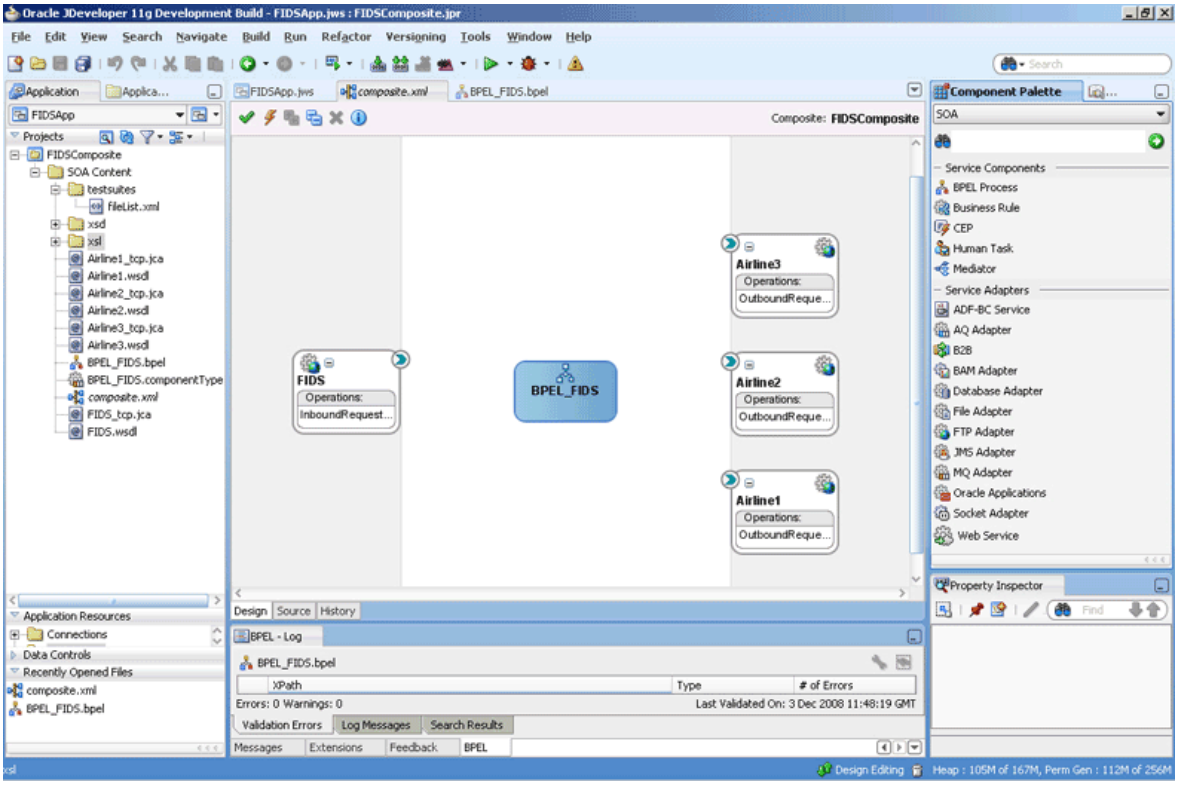


Perform the following steps to create an outbound Oracle Socket Adapter service for the Airline3 server socket:

1. Drag and drop **Socket Adapter** from the Components window to the External References swim lane. The **Welcome** page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `Airline3` in the **Service Name** field.
4. Click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Outbound Synchronous Request/Reply** as the operation type.
7. Click **Next**. The **Socket Connection** page is displayed.
8. Enter `eis/socket/OutboundSocketAdapter` in the **Socket Connection JNDI Name** field and then select **Specify Host and Port**.
9. Enter the name of the system where the Airline3 socket server program must run in the **HostName** field and `9003` in the **PortNumber** field, and click **Next**. The **Messages** page is displayed.
10. Click **Browse For Schema File** that appears at the end of the URL field in the **Request Message Schema** box. The **Type Chooser** dialog is displayed.
11. Click **Project Schema Files**, **Airline3.xsd**, and **src**.

12. Click **OK**. The URL field in the **Messages** page is populated with the `Airline3.xsd` file.
13. Click **Browse For Schema File** that appears at the end of the URL field in the Reply Message Schema box. The Type Chooser dialog is displayed.
14. Click **Project Schema Files**, **Airline3.xsd**, and **airline**.
15. Click **OK**. The URL fields in the Messages page are populated with the `Airline3.xsd` files.
16. Click **Next**. The Protocol page is displayed.
17. Select **Use Custom Java Code to define the handshake**.
18. Enter `com.oracle.socket.fids.custom.Airline3Custom` in the Java Class field.
19. Click **Finish**. The **composite.xml** page appears, as shown in Figure 5-87.

Figure 5-87 The JDeveloper - composite.xml Page



Wiring Services and Activities

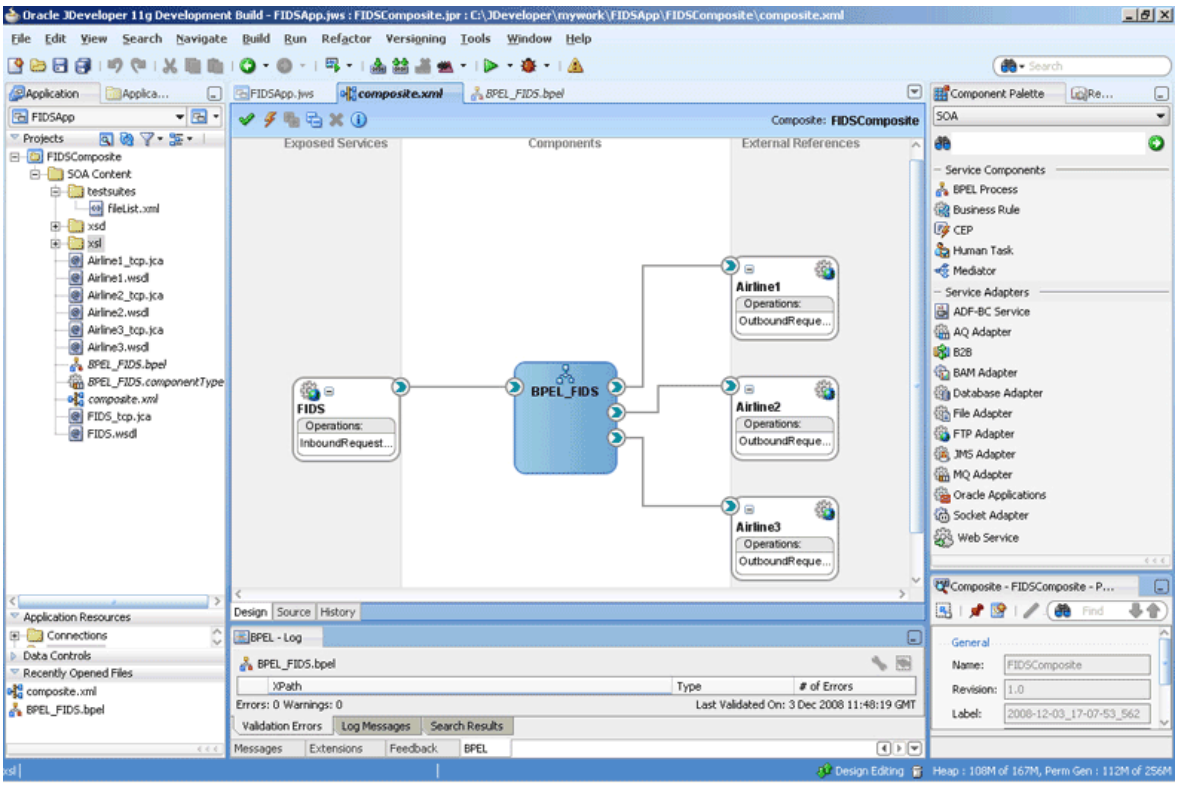
You have to assemble or wire the components that you have created: Inbound adapter service, BPEL process, Outbound adapter references. Perform the following steps to wire the components:

1. Drag the small triangle in the FIDS client in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL_FIDS process in the Components area.

2. Drag the small triangle in the BPEL_FIDS process in the Components area to the drop zone that appears as a green triangle in the Airline1, Airline2, and Airline3 servers in the External References area.

   The JDeveloper **composite.xml** file appears, as shown in Figure 5-88.

Figure 5-88 The JDeveloper - composite.xml

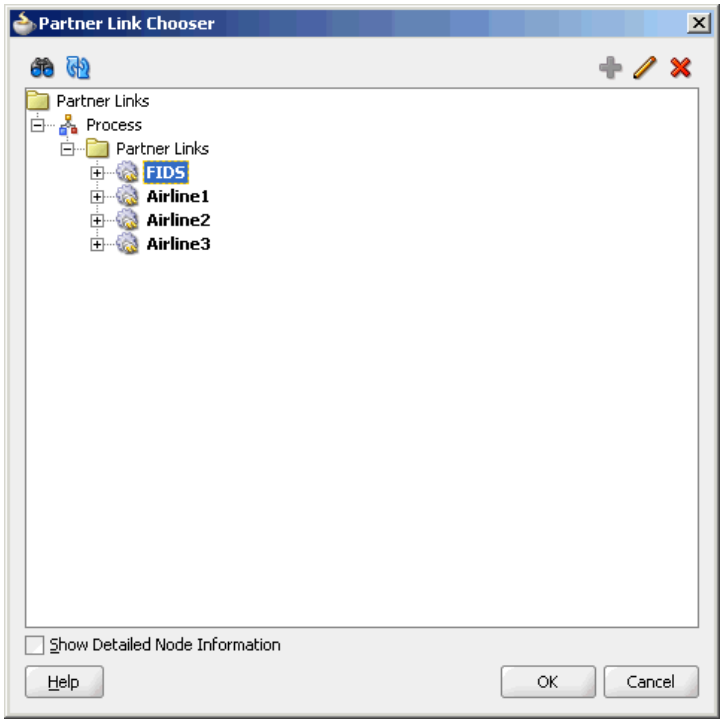


3. Click **File**, **Save All**.

Add a Receive Activity

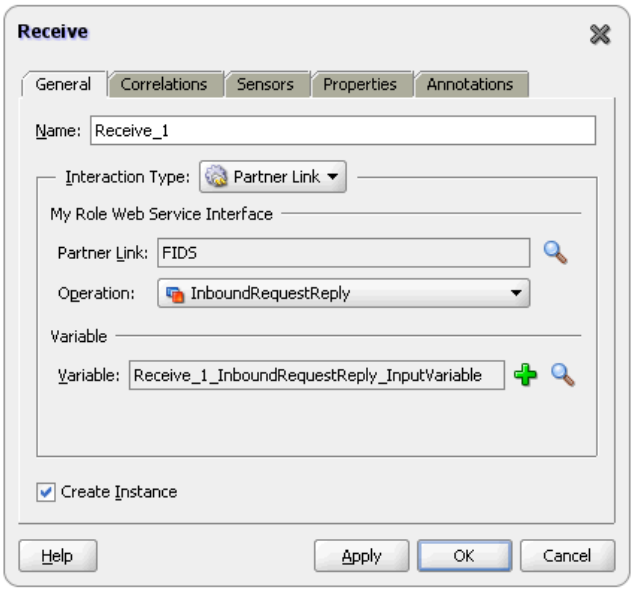
1. Double-click **BPEL_FIDS**. The **BPELFIDS.bpel** page is displayed.
2. Drag and drop a **Receive** activity from the **Components** window to the design area.
3. Double-click the **Receive** activity. The **Receive** dialog is displayed.
4. Retain the default name `Receive_1` in the **Name** field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.
6. Select **FIDS**, as shown in Figure 5-89, and click **OK**.

Figure 5-89 The Partner Link Chooser Dialog



7. Click the **Auto-Create Variable** icon to the right of the Variable field in the **Receive** dialog, as shown in Figure 5-90. The **Create Variable** dialog is displayed.

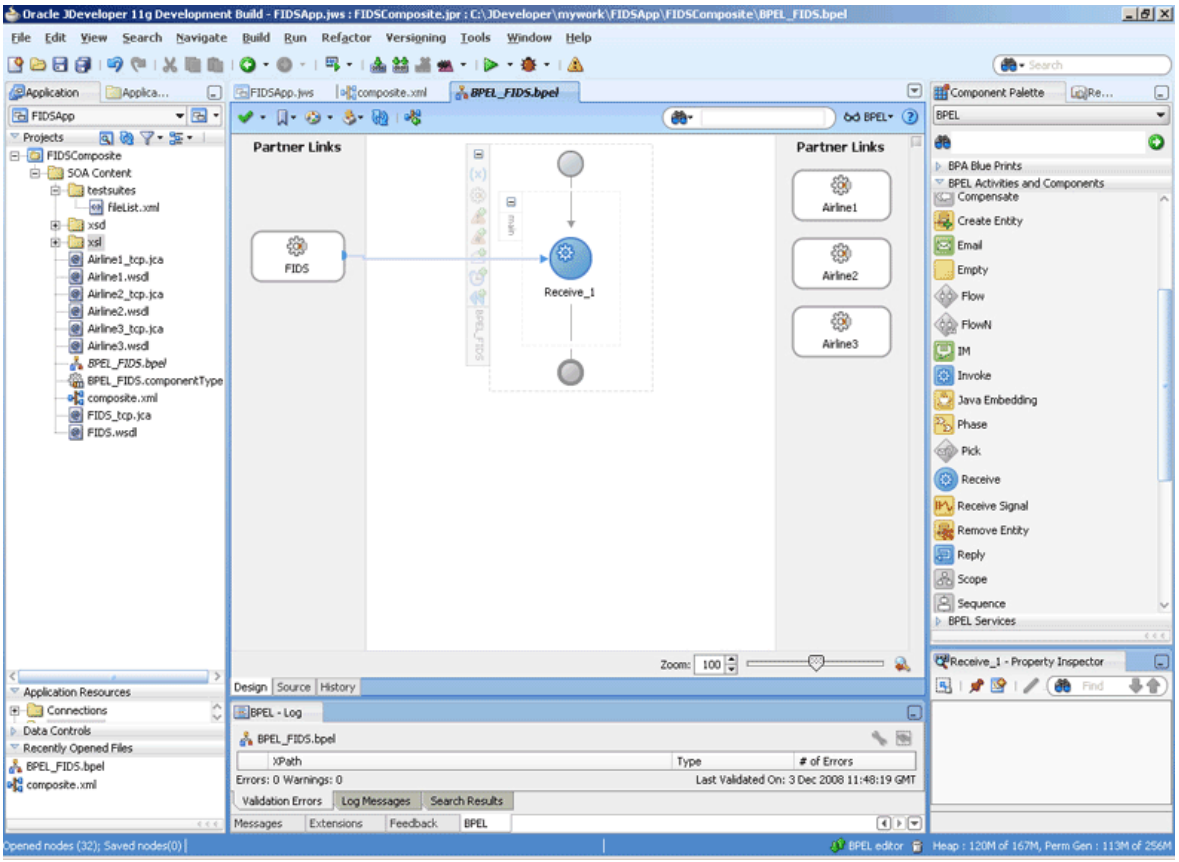
Figure 5-90 The Receive Dialog



8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.

9. Check **Create Instance**, and click **OK**. The JDeveloper BPEL_FIDS.bpel page appears, as shown in Figure 5-91.

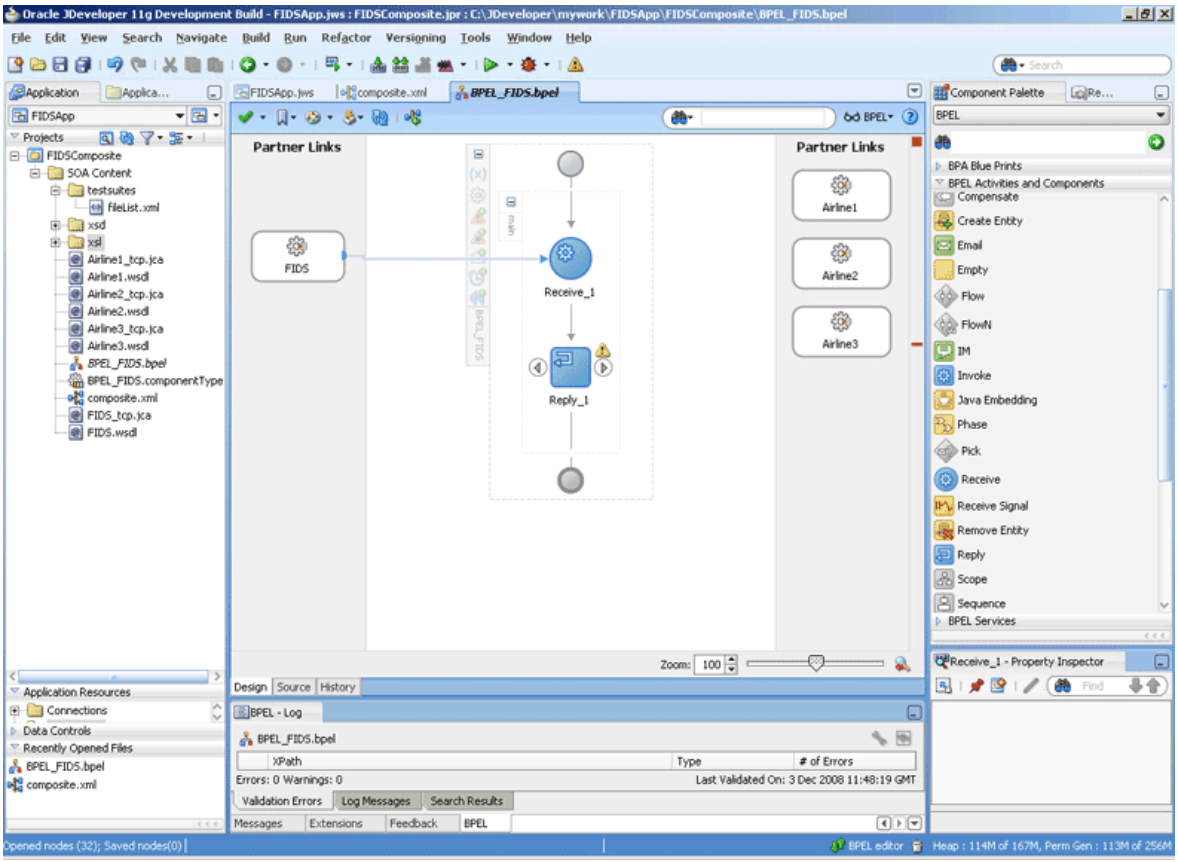
Figure 5-91 The JDeveloper - BPEL_FIDS.bpel



Add a Reply Activity

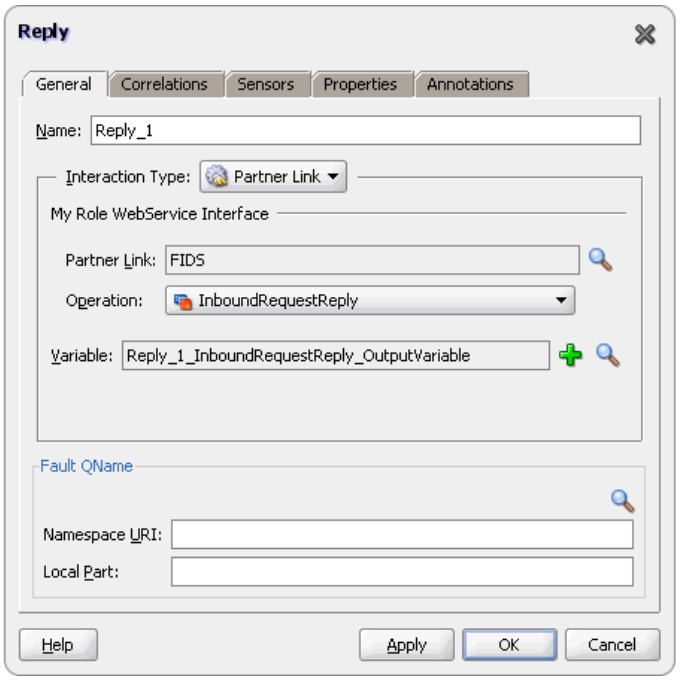
1. Drag and drop an **Reply** activity from the Components window to the design area, as shown in Figure 5-92.

Figure 5-92 The JDeveloper - BPEL_FIDS.bpel



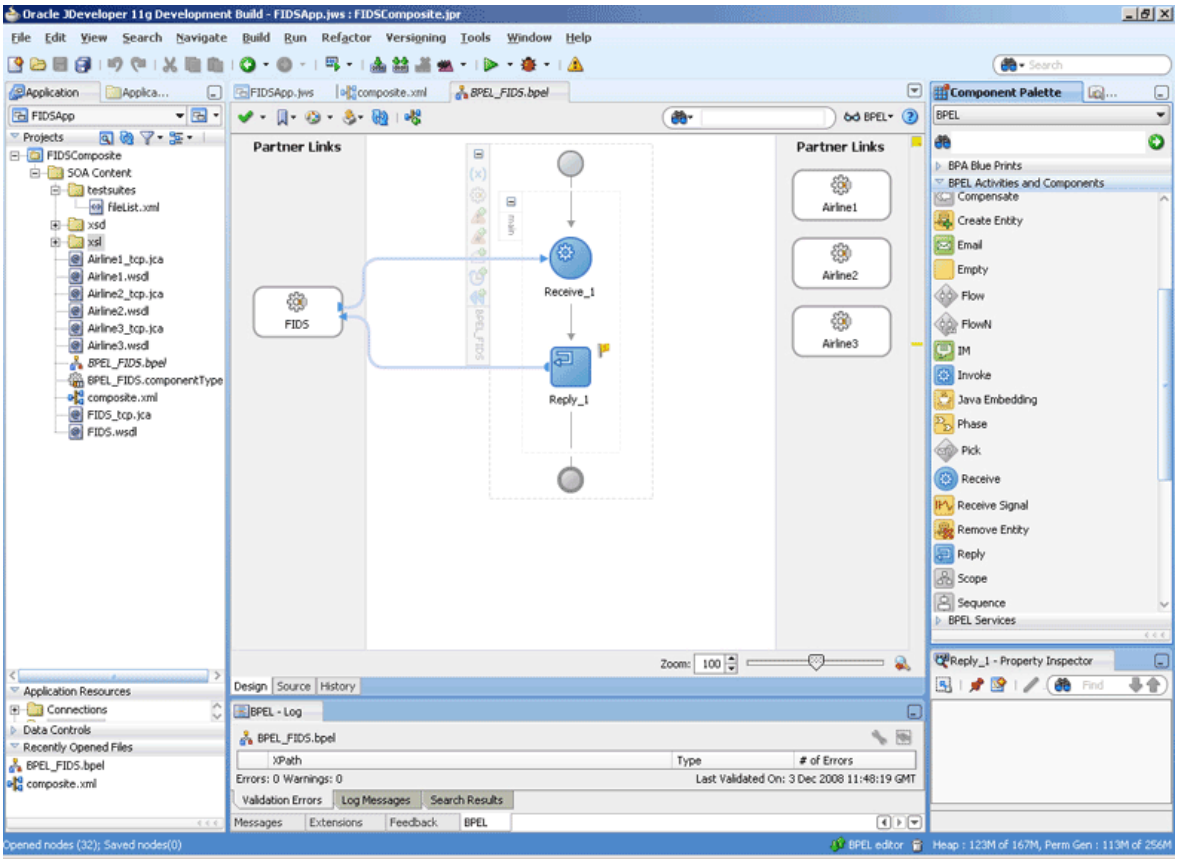
2. Double-click the **Reply** activity. The Reply dialog is displayed.
3. Retain the default name `Reply_1` in the Name field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **FIDS**, as shown in Figure 5-89, and click **OK**.
6. Click the **Auto-Create Variable** icon to the right of the Variable field in the **Reply** dialog. The **Create Variable** dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name, as shown in Figure 5-93.

Figure 5-93 The Reply Dialog



8. Click **OK**. The JDeveloper **BPEL_FIDS.bpel** page appears, as shown in Figure 5-94.

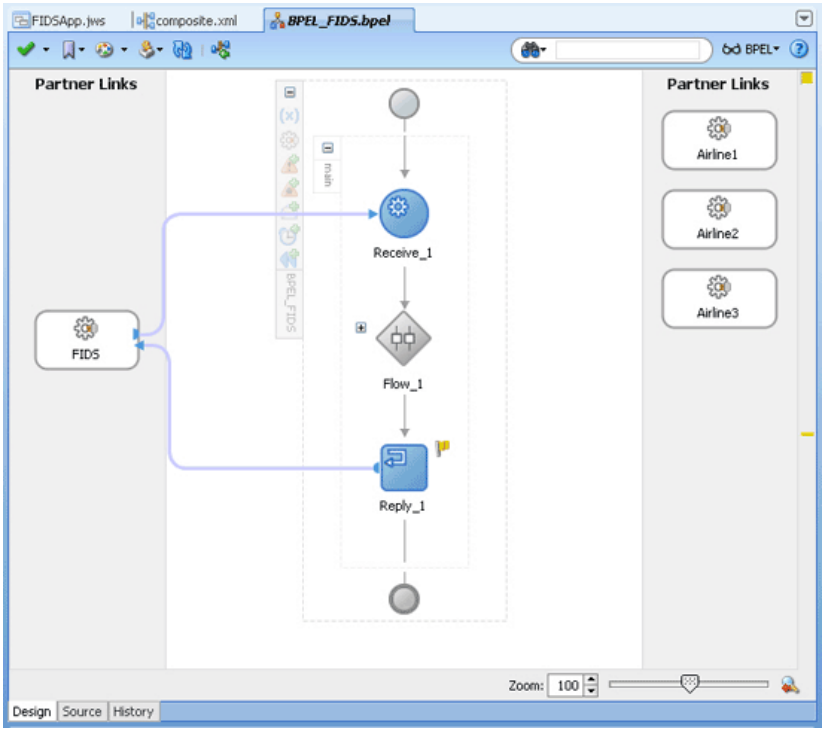
Figure 5-94 The JDeveloper - BPEL_FIDS.bpel



Add a Flow Activity

1. Drag and drop a **Flow** activity from the Components window in between the Receive and the Reply activities in the design area, as shown in Figure 5-95.

Figure 5-95 The JDeveloper - BPEL_FIDS.bpel

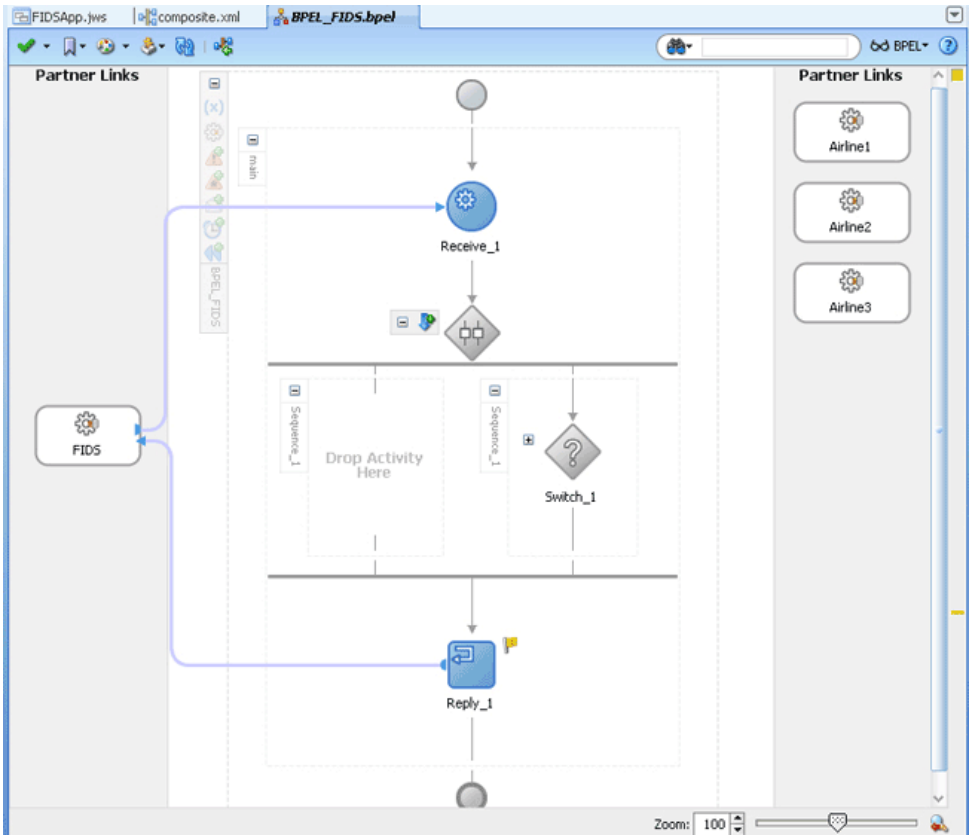


2. Expand the Flow_1 activity. This displays a screen to create sequences.

Design the Flow for Airline1 Server

1. Drag and drop a **Switch** activity from the Components window to Sequence_1, as shown in Figure 5-96.

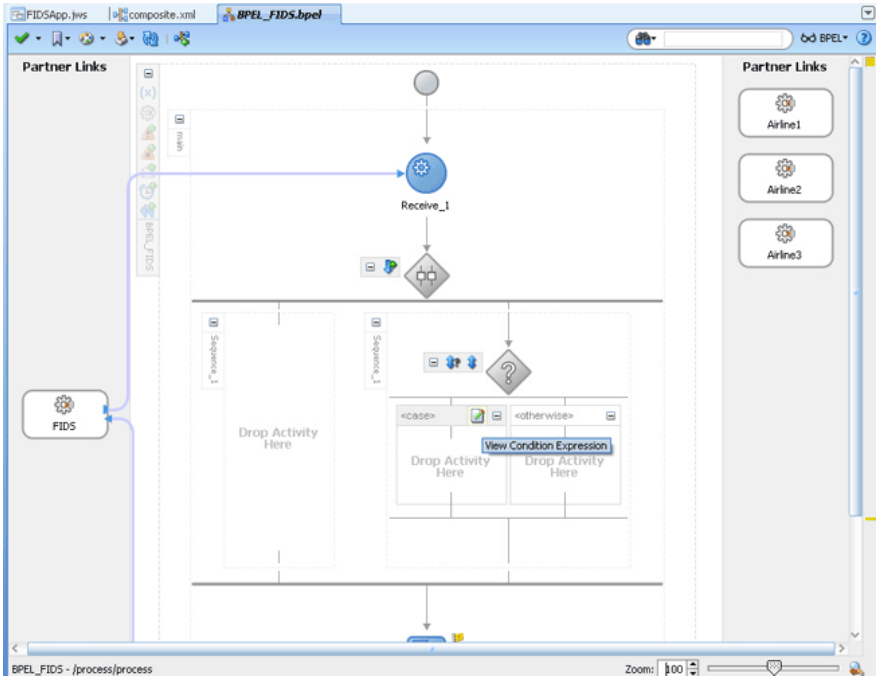
Figure 5-96 The JDeveloper - BPEL_FIDS.bpel Page



2. Expand the **Switch** activity. This displays a screen to enter the values for <case> and <otherwise>.

3. In the `<case>` section, click the **View Condition Expression** icon, as shown in Figure 5-97. The **Condition Expression** pop-up window is displayed.

Figure 5-97 The JDeveloper - BPEL_FIDS.bpel Page



4. Click the **Xpath Expression Builder** icon in the pop-up window. The **Expression Builder** dialog is displayed.

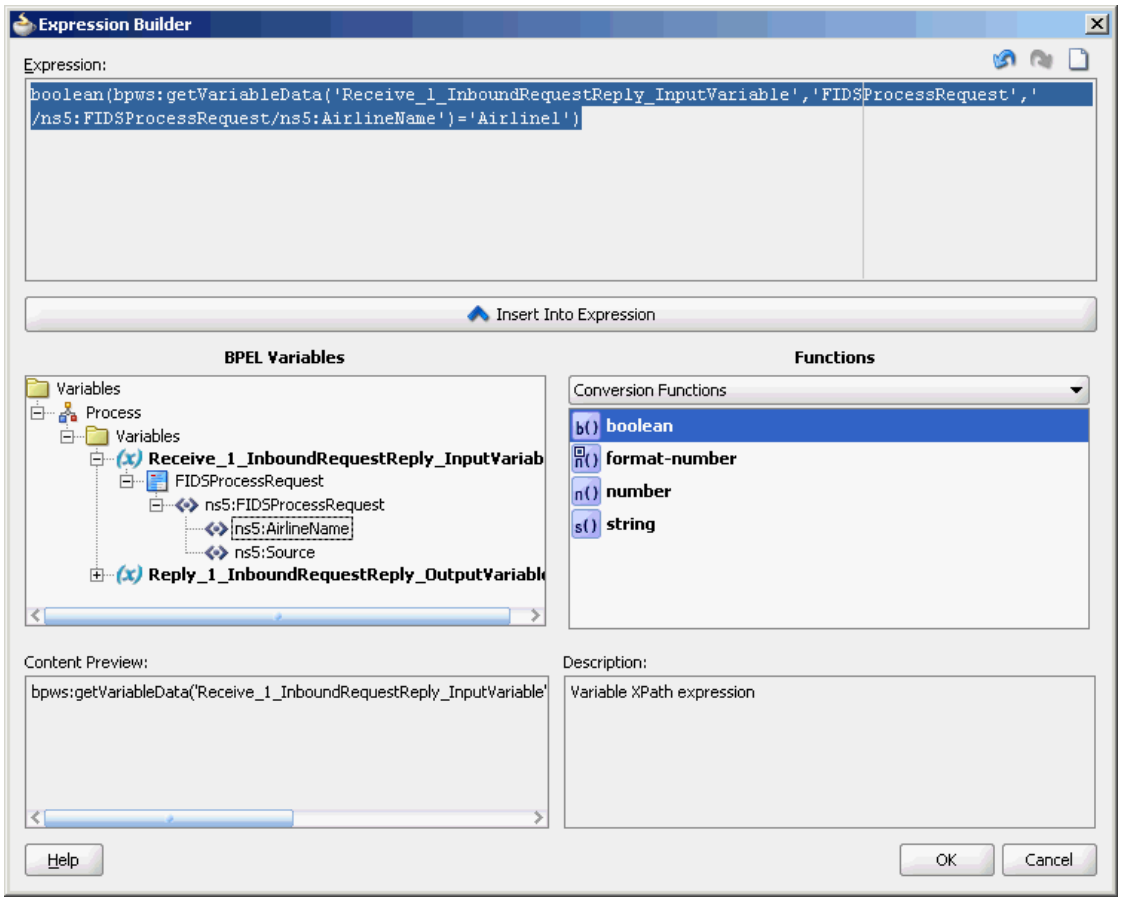
5. Enter `boolean(bpws:getVariableData('Receive_1_InboundRequestReply_InputVariable','FIDSProcessRequest','/ns5:FIDSProcessRequest/ns5:AirlineName')='Airline1')` as the expression, as shown in Figure 5-98, and click **OK**.

   The screen returns to the **Condition Expression** pop-up window.

Note:

This expression ensures that this flow is executed only when information for Airline1 is requested.

Figure 5-98 The Expression Builder Dialog



6. Add an Invoke activity to the <case> section.

   a. Drag and drop an **Invoke** activity in the <case> section.

   b. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.

   c. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.

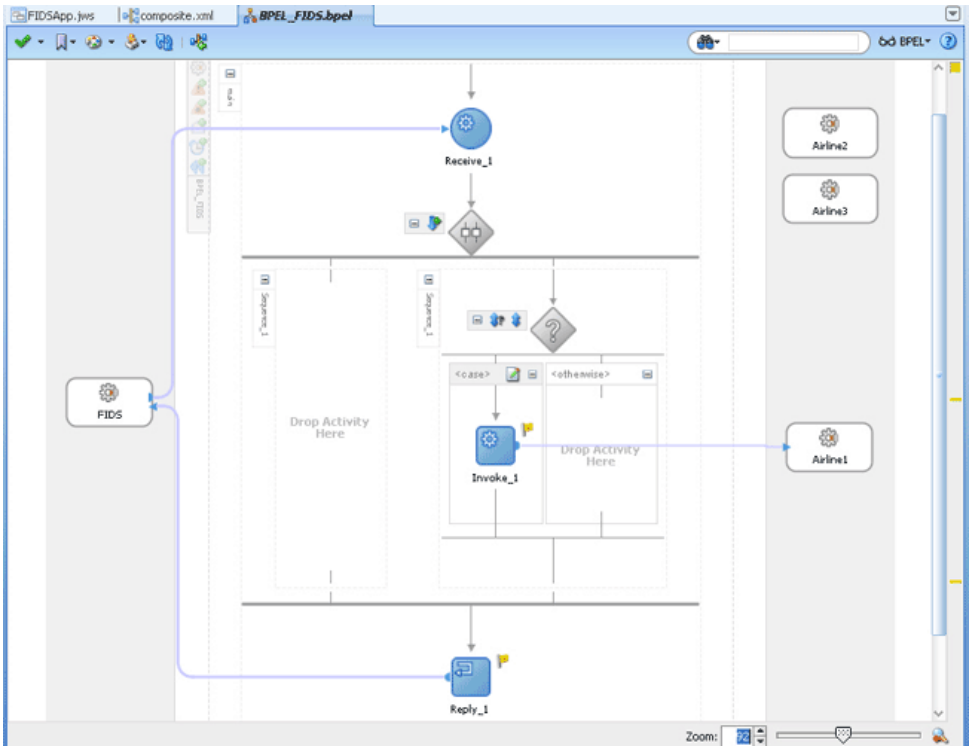
   d. Select **Airline1**, and click **OK**.

   e. Click the **Automatically Create Input Variable** and the **Automatically Create Output Variable** icons to the right of the Input and Output Variable fields in the Invoke dialog. The **Create Variable** dialogs are displayed.

   f. Select the default variable names and click **OK**. The Variable fields are populated with the default variable name. The Invoke dialog is displayed.

   g. Click **OK**. The JDeveloper **BPEL_FIDS.bpel** page is displayed, as shown in Figure 5-99.

Figure 5-99 The JDeveloper - BPEL_FIDS.bpel



7. Add an assign activity to the <case> section.

   a. Drag and drop an **Assign** activity from the Components window before the Invoke_1 activity in the <case> section.

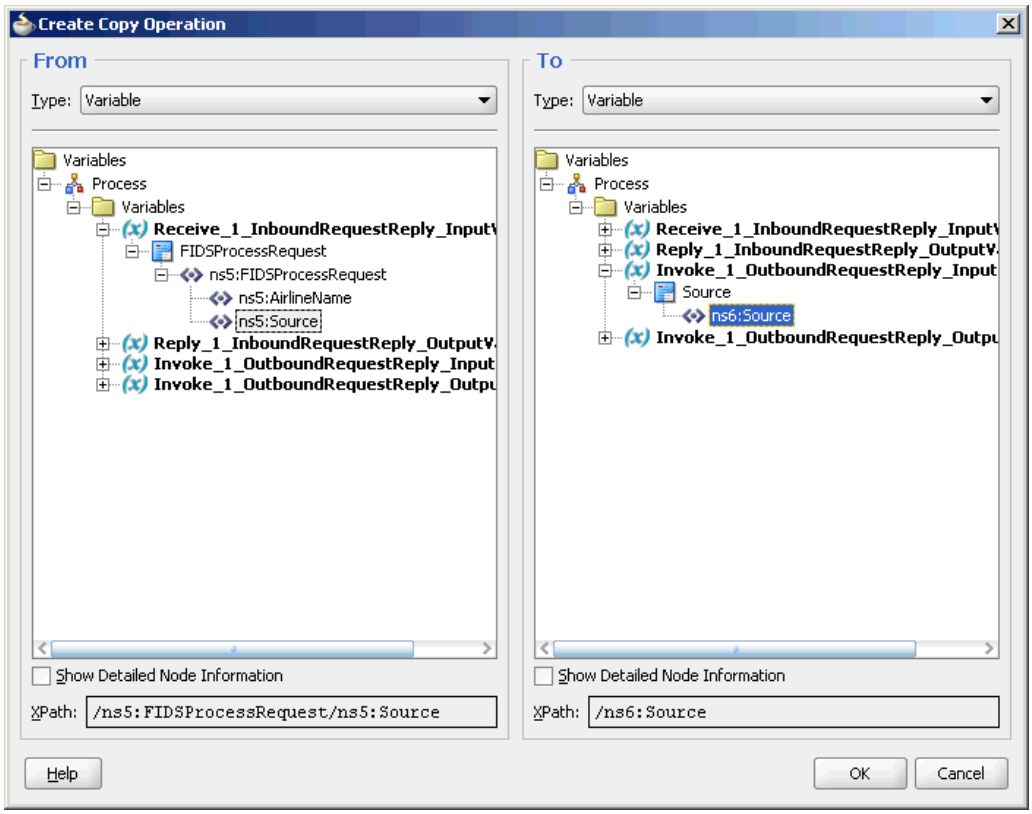
   b. Double-click the **Assign_1** activity. The **Assign** dialog is displayed.

   c. Click the **Copy Operation** tab. The **Assign** dialog is displayed.

   d. Select **Copy Operation**. The **Create Copy Operation** dialog is displayed.

   e. Create the copy operation between the source from the input variable of the Receive_1 activity and the source from the input variable of the Invoke_1 activity, as shown in Figure 5-100.

Figure 5-100 The Create Copy Operation Dialog

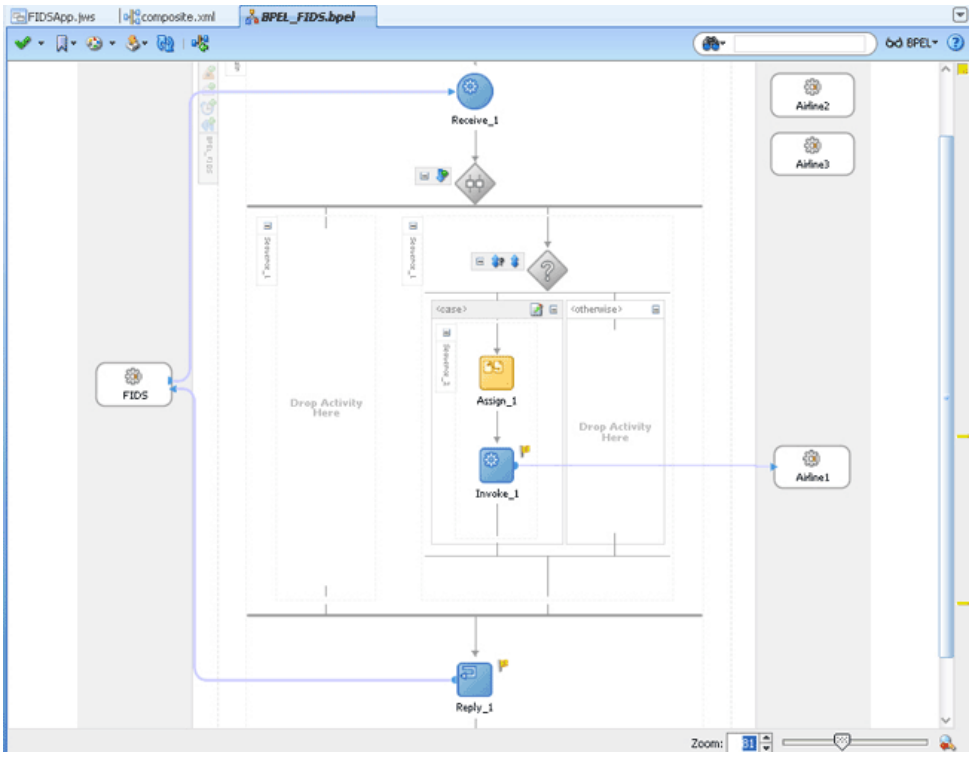


f. Click **OK** in the dialog. The **Create Copy Operation** dialog is displayed.

g. Click **OK**.

The **BPEL_FIDS.bpel** page is displayed, as shown in Figure 5-101.

Figure 5-101 The JDeveloper - BPELFIDS.bpel



8. Add a Transform activity to the <case> section.

   a. Drag and drop a **Transform** activity in the <case> section, after the Invoke_1 activity.

   b. Double-click the **Transform** activity.

   c. Click the **Transformation** tab.

   d. Click the **Create...(Alt+N)** icon. The Source Variable dialog is displayed.

   e. Select **Invoke_1_OutboundRequestReply_OutputVariable** from the Source Variable list and click **OK**.

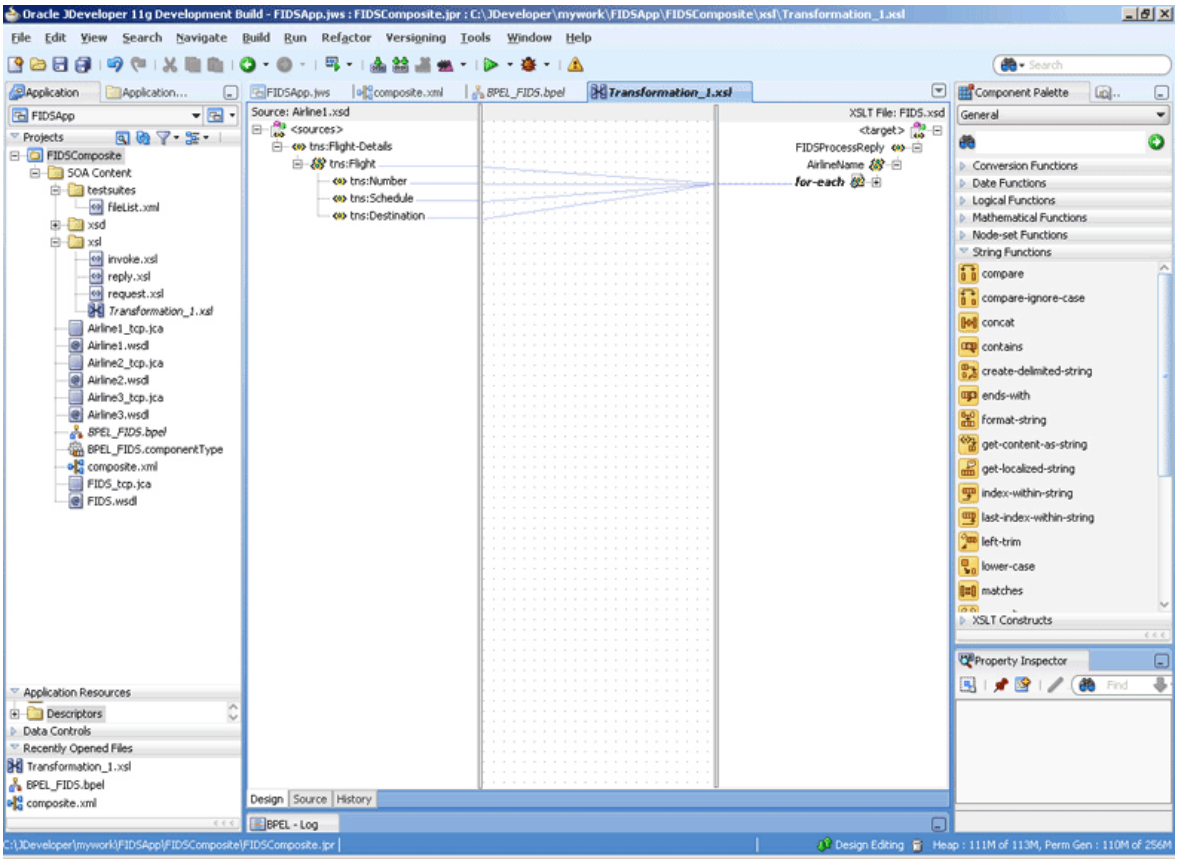
   f. Select **Reply_1_InboundRequestReply_OutputVariable** from the Target Variable list.

   g. Click **OK**. The XSL mapper tool is displayed.

   h. Link the `tns:Flight` node from the source, on the left pane to the target `FlightDetails` node on the right pane. The **Auto Map Preferences** dialog appears.

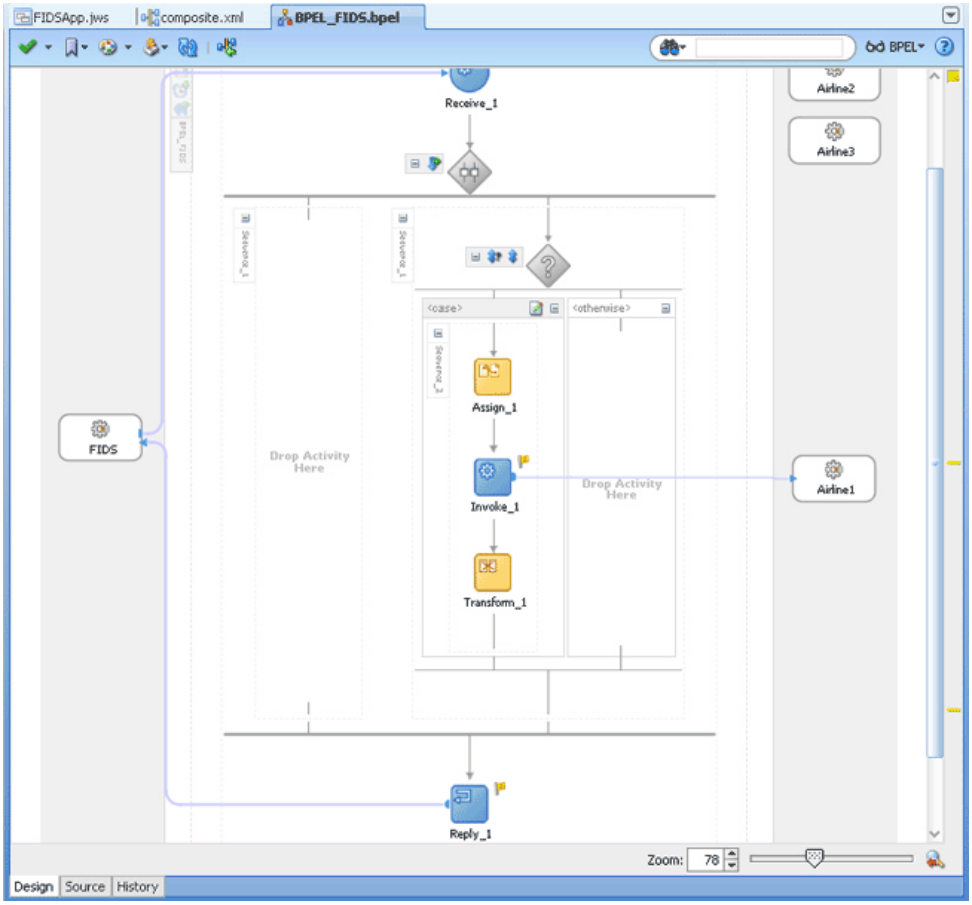
   i. Click **OK**. The Transformation_1.xsl (XSL mapper tool) is displayed, as shown in Figure 5-102.

Figure 5-102 The JDeveloper - Transformation_1.XSL Page



9. Click **File**, **Save All**. The **BPEL_FIDS.bpel** page is displayed, as shown in Figure 5-103, with the flow defined for the Airline1 server.

Figure 5-103 The JDeveloper - BPEL_FIDS.bpel



Design the Flow for Airline2 Server

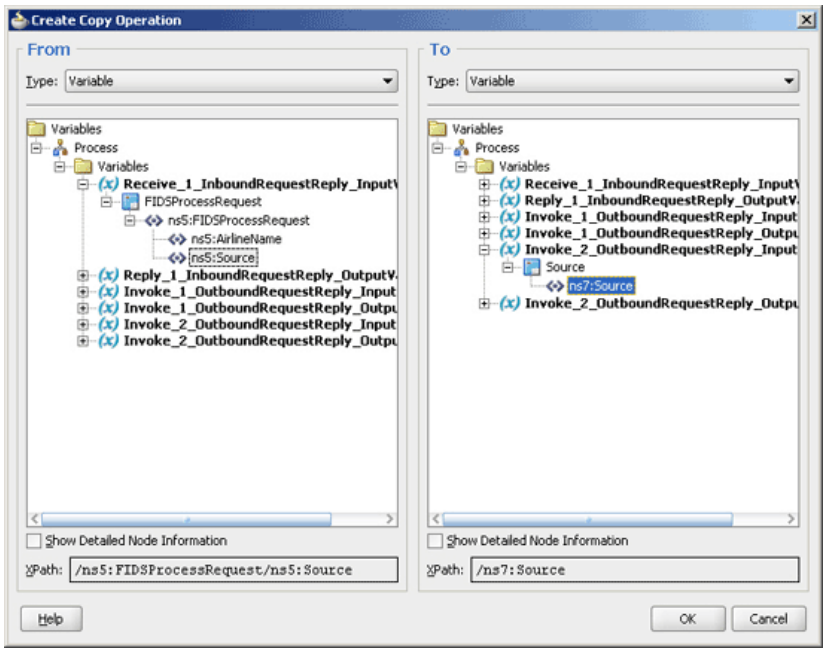
1. Double-click the empty Sequence activity and enter `Sequence_2` in the **Name** field.
2. Drag and drop a **Switch** activity from the Components window to Sequence_2.
3. Expand the **Switch** activity. This displays a screen to enter the values for <case> and <otherwise>.
4. In the `<case>` section, click the **View Condition Expression** icon. The Condition Expression pop-up window is displayed.
5. Click the **Xpath Expression Builder** icon in the pop-up window. The Expression Builder dialog is displayed.
6. Enter `boolean(bpws:getVariableData('Receive_1_InboundRequestReply_InputVariable','FIDSProcessRequest','/ns5:FIDSProcessRequest/ns5:AirlineName')='Airline2')` as the expression, and click **OK**. The screen returns to the Condition Expression pop-up window.

Note:

This expression ensures that this flow is executed only when information for Airline2 is requested.

7. Add an invoke activity to the <case> section.
   a. Drag and drop an **Invoke** activity in the <case> section.
   b. Double-click the **Invoke_2** activity. The Invoke dialog is displayed.
   c. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
   d. Select **Airline2**, and click **OK**.
   e. Click the **Automatically Create Input Variable** and the **Automatically Create Output Variable** icons to the right of the Input and Output Variable fields in the Invoke dialog. The **Create Variable** dialogs are displayed.
   f. Select the default variable names and click **OK**. The Input and Output Variable fields are populated with the default variable names. The Invoke dialog is displayed.
   g. Click **OK**. An Invoke activity is added to the JDeveloper **BPEL_FIDS.bpel** page under Sequence_2.
8. Add an assign activity to the <case> section.
   a. Drag and drop an **Assign** activity from the Components window before the Invoke activity in the <case> section.
   b. Double-click the **Assign_2** activity. The **Assign** dialog is displayed.
   c. Click the **Copy Operation** tab. The **Assign** dialog is displayed.
   d. Select **Copy Operation**. The **Create Copy Operation** dialog is displayed.
   e. Create the copy operation between the source from the input variable of the Receive_1 activity and the source from the input variable of the Invoke_2 activity, as shown in Figure 5-104.

Figure 5-104 The Create Copy Operation Dialog



f. Click **OK** in the Create Copy Operation dialog.

g. Click **OK**.

9. Create a temporary variable and add a Transform activity to the <case> section.

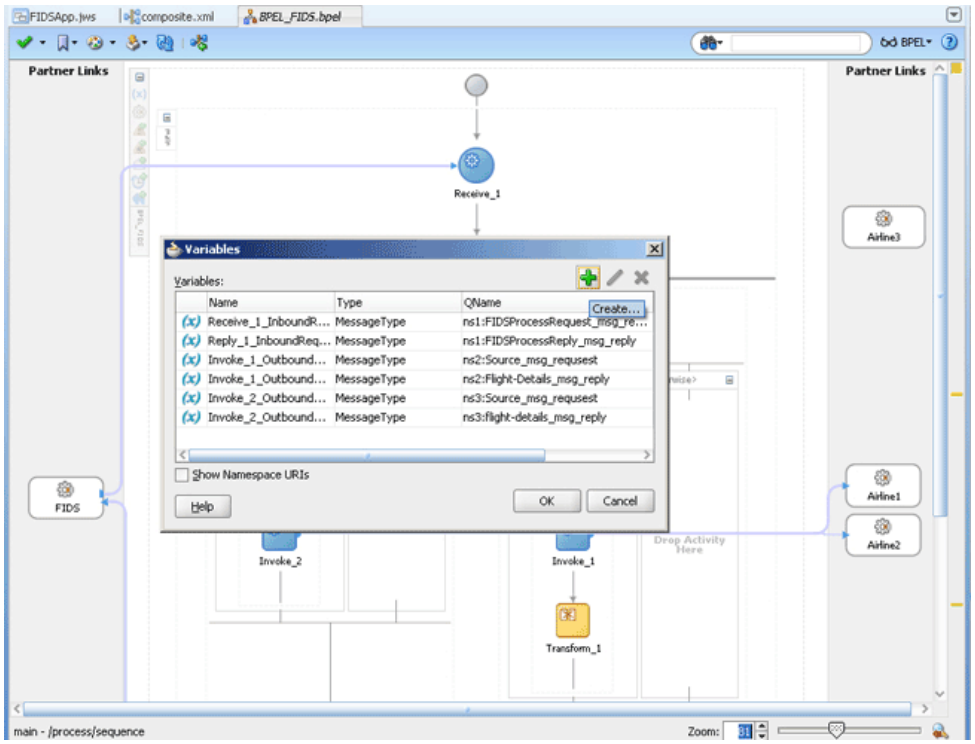
Note:

The temporary variable is used for storing flight details from the Airline2 server, which would later be appended to the reply variable.

---

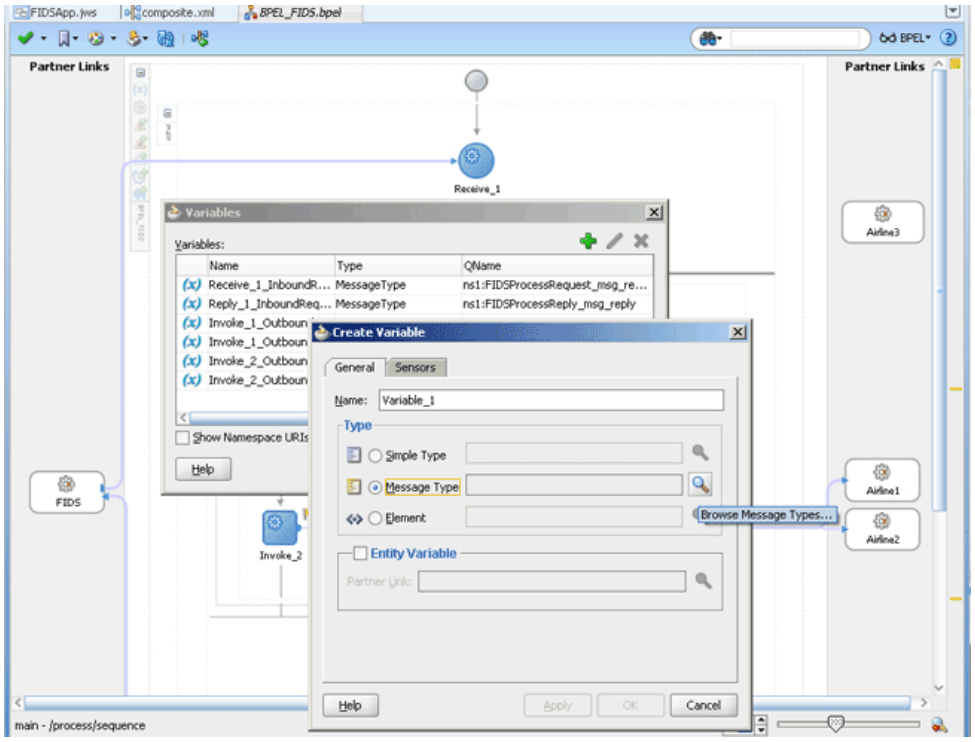
a. Click the **Variables...** icon represented by (x). The **Variables** dialog is displayed, as shown in Figure 5-105.

Figure 5-105 The Variables Dialog



b. Click the **Create...** icon. The Create Variable dialog is displayed, as shown in Figure 5-106.

Figure 5-106 The Create Variable Dialog

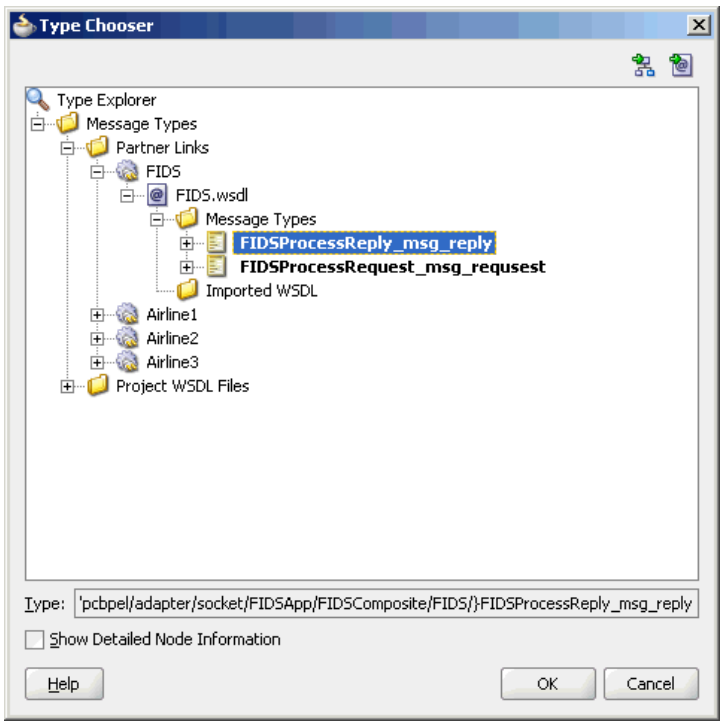


c. Select **Message Type** as the variable type.

d. Click the **Browse Message Types** icon at the end of the **Message Type** field. The Type Chooser dialog is displayed.

e. Click **Message Types**, **Partner Links**, **FIDS**, **FIDS.wsdl**, **Message Types**, and then **FIDSProcessReply_msg_reply**, as shown in Figure 5-107.

Figure 5-107 The Type Chooser Dialog



f. Click **OK**. The Message type field in the **Create Variable** dialog is populated with the FIDSProcessReply_msg_reply partner link.

g. Click **OK**. A variable, Variable_1, of type Message Type is added to the Variables list in the Variable dialog.

h. Click **OK** to return to the BPEL_FIDS.bpel page.

i. Drag and drop a **Transform** activity in the <case> section, after the Invoke_2 activity.

j. Double-click the **Transform_2** activity.

k. Click the **Transformation** tab.

l. Click the **Create...** icon. The **Source Variable** dialog is displayed.

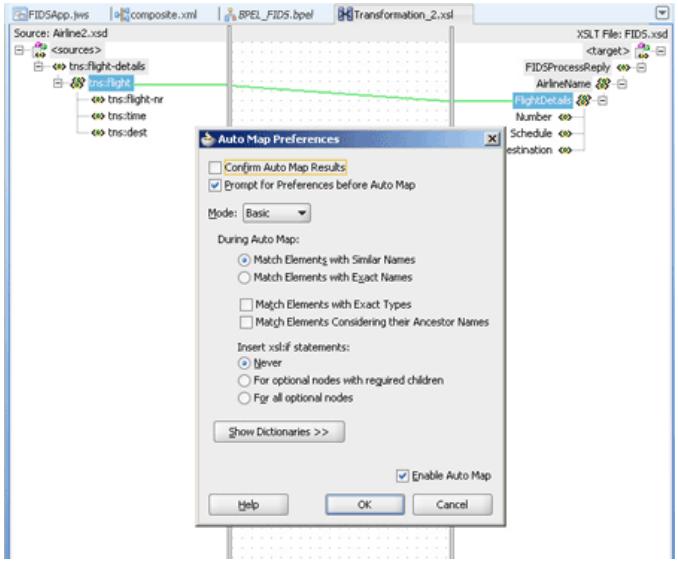
m. Select **Invoke_2_OutboundRequestReply_OutputVariable** from the Source Variable list and click **OK**.

n. Select **Variable_1** from the **Target Variable** list.

o. Click **OK**. The **XSL mapper tool** is displayed.

p. Link the `tns:flight` node from the source, on the left pane to the target `FlightDetails` node on the right pane. The **Auto Map Preferences** dialog appears, as shown in Figure 5-108.

Figure 5-108 The Transformation_2.XSL Page With Auto Map Preference Dialog

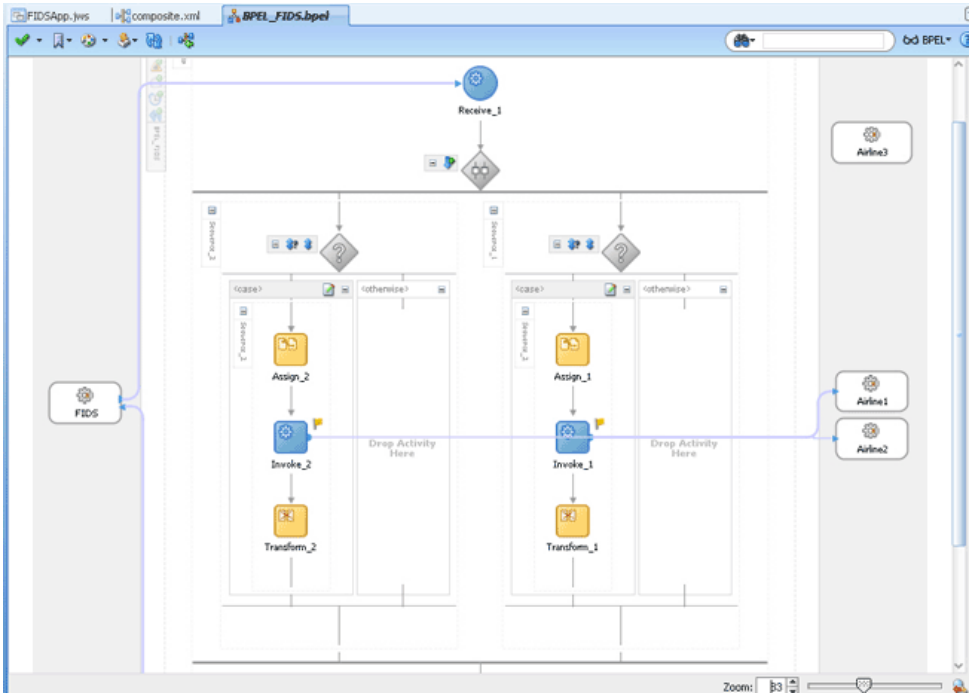


q. Click **OK**. The `Transformation_2.xsl` (XSL mapper tool) file with the XSL mapping is displayed.

10. Click **File**, **Save All**.

The **BPEL_FIDS.bpel** page is displayed, as shown in Figure 5-109, with the flow defined for the Airline2 server.

Figure 5-109 The JDeveloper - BPEL_FIDS.bpel



Design the Flow for Airline3 Server

1. Right-click the **Flow_1** activity. Click **Add Sequence** from the menu. `Sequence_3` is added.
2. Drag and drop a **Switch** activity from the Components window to Sequence_3.
3. Expand the **Switch** activity. This displays a screen to enter the values for <case> and <otherwise>.
4. In the `<case>` section, click the **View Condition Expression** icon. The **Condition Expression** pop-up window is displayed.
5. Click the **Xpath Expression Builder** icon in the pop-up window. The **Expression Builder** dialog is displayed.
6. Enter `boolean(bpws:getVariableData('Receive_1_InboundRequestReply_InputVariable','FIDSProcessRequest','/ns5:FIDSProcessRequest/ns5:AirlineName')='Airline3')` as the expression, and click **OK**. The screen returns to the Condition Expression pop-up window.

   Note:

   This expression ensures that this flow is executed only when information for Airline3 is requested.

   ---

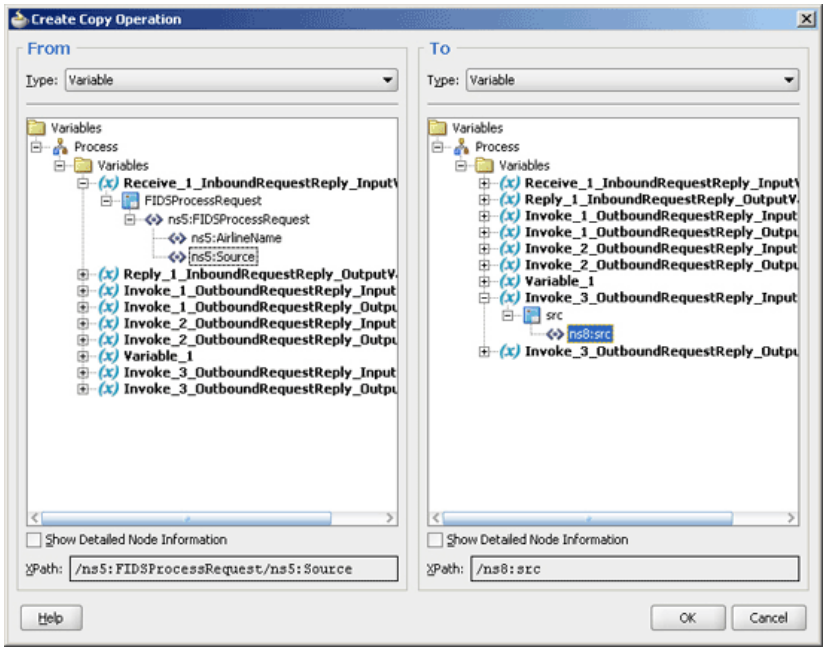
7. Add an invoke activity to the <case> section.
   a. Drag and drop an **Invoke** activity in the <case> section.
   b. Double-click the **Invoke_3** activity. The **Invoke** dialog is displayed.
   c. Click **Browse Partner Links** at the end of the Partner Link field. The **Partner Link Chooser** dialog is displayed.
   d. Select **Airline3**, and click **OK**.
   e. Click the **Automatically Create Input Variable** and **Automatically Create Output Variable** icon to the right of the Input and Output Variable field in the Invoke dialog. The Create Variable dialogs are displayed.
   f. Select the default variable names and click **OK**. The Input and Output Variable fields are populated with the default variable names. The Invoke dialog is displayed.
   g. Click **OK**. An Invoke activity is added to JDeveloper **BPEL_FIDS.bpel** page under Sequence_3.
8. Add an assign activity to the <case> section.
   a. Drag and drop an **Assign** activity from the Components window before the Invoke activity in the <case> section.
   b. Double-click the **Assign_3** activity. The **Assign** dialog is displayed.

c. Click the **Copy Operation** tab. The **Assign** dialog is displayed.

d. Select **Copy Operation**. The **Create Copy Operation** dialog is displayed.

e. Create the copy operation between the source from the input variable of the Receive_1 activity and the source from the input variable of the Invoke_3 activity, as shown in Figure 5-110.

Figure 5-110 The Create Copy Operation Dialog



f. Click **OK** in the **Create Copy Operation** dialog.

g. Click **OK**.

9. Create a temporary variable and add a Transform activity to the <case> section.

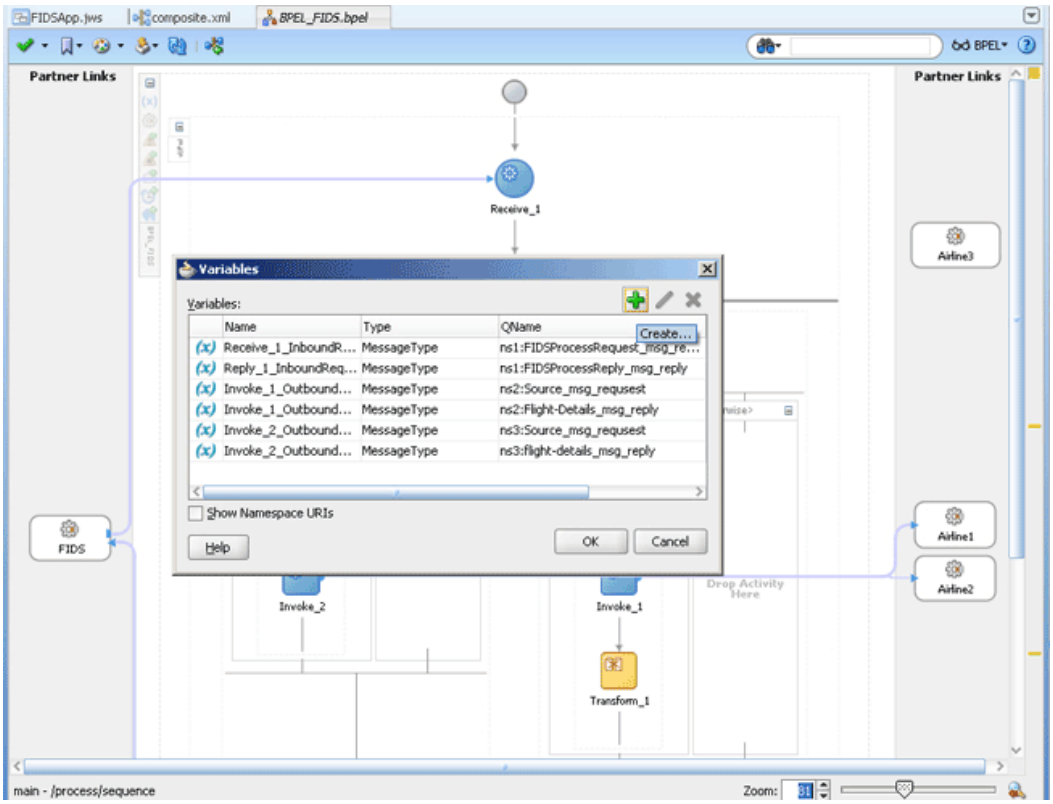
Note:

The temporary variable is used for storing flight details from the Airline3 server, which would later be appended to the reply variable.

---

a. Click the **Variables...** icon represented by (x). The Variables dialog is displayed, as shown in Figure 5-111.

Figure 5-111 The Variables Dialog



b. Click the **Create...** icon. The Create Variable dialog is displayed.

c. Select **Message Type** as the variable type.

d. Click the **Browse Message Types** icon at the end of the Message Type field. The Type Chooser dialog is displayed.

e. Click **Message Types**, **Partner Links**, **FIDS**, **FIDS.wsdl**, **Message Types**, and then **FIDSProcessReply_msg_reply**, as shown in Figure 5-107.

f. Click **OK**. The Message type field in the Create Variable dialog is populated with the FIDSProcessReply_msg_reply partner link.

g. Click **OK**. A variable, Variable_2, of type Message Type is added to the Variables list in the Variable dialog.

h. Click **OK** to return to the **BPEL_FIDS.bpel** page.

i. Drag and drop a **Transform** activity in the <case> section, after the Invoke_3 activity.

j. Double-click the **Transform_3** activity.

k. Click the **Transformation** tab.

l. Click the **Create...** icon. The **Source Variable** dialog is displayed.

m. Select **Invoke_3_OutboundRequestReply_OutputVariable** from the Source Variable list and click **OK**.

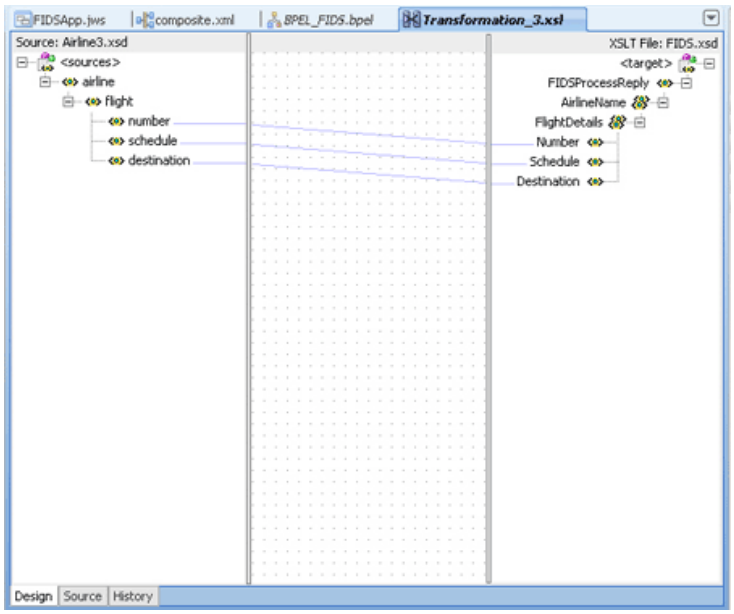
n. Select **Variable_2** from the **Target Variable** list.

o. Click **OK**. The **XSL mapper tool** is displayed.

p. Link the `tns:flight` node from the source, on the left pane to the target `FlightDetails` node on the right pane. The **Auto Map Preferences** dialog appears.

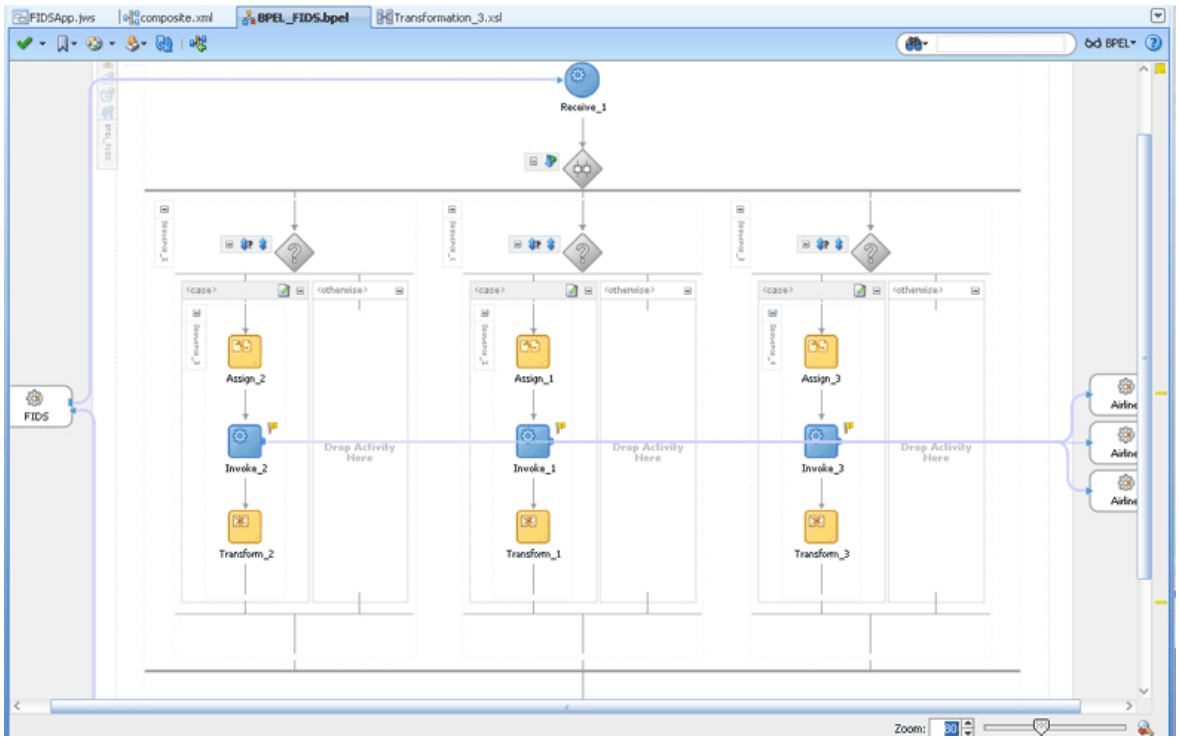
q. Click **OK**. The Transformation_3.xsl (XSL mapper tool) file with the XSL mapping is displayed, as shown in Figure 5-112.

Figure 5-112 The Transformation_3.XSL Page



10. Click **File**, **Save All**. The **BPEL_FIDS.bpel** page is displayed, as shown in Figure 5-113, with the flow defined for the Airline3 server.

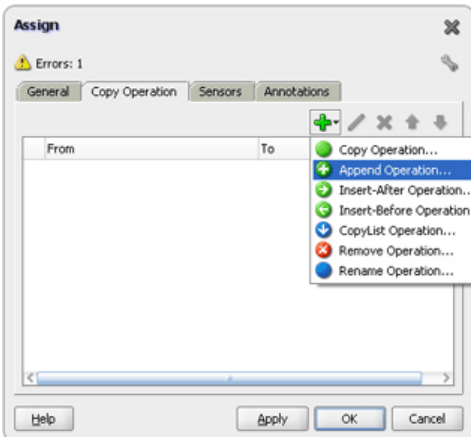
Figure 5-113 The JDeveloper - BPEL_FIDS.bpel



Add an Assign Activity

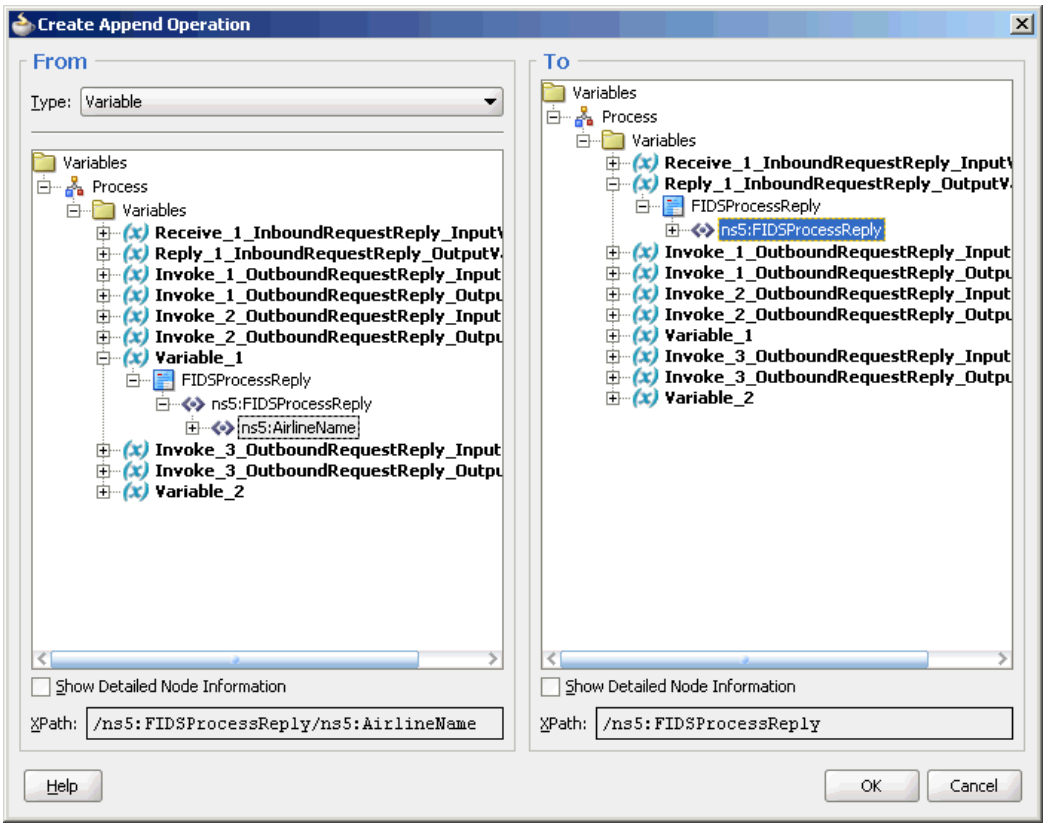
1. Drag and drop an **Assign** activity from the Components window in between the Reply and Receive activities in the design area.
2. Double-click the **Assign_4** activity. The Assign dialog is displayed.
3. Click the **Copy Operation** tab. The Assign dialog is displayed, as shown in Figure 5-114.

Figure 5-114 The Assign Dialog - Copy Operation Tab



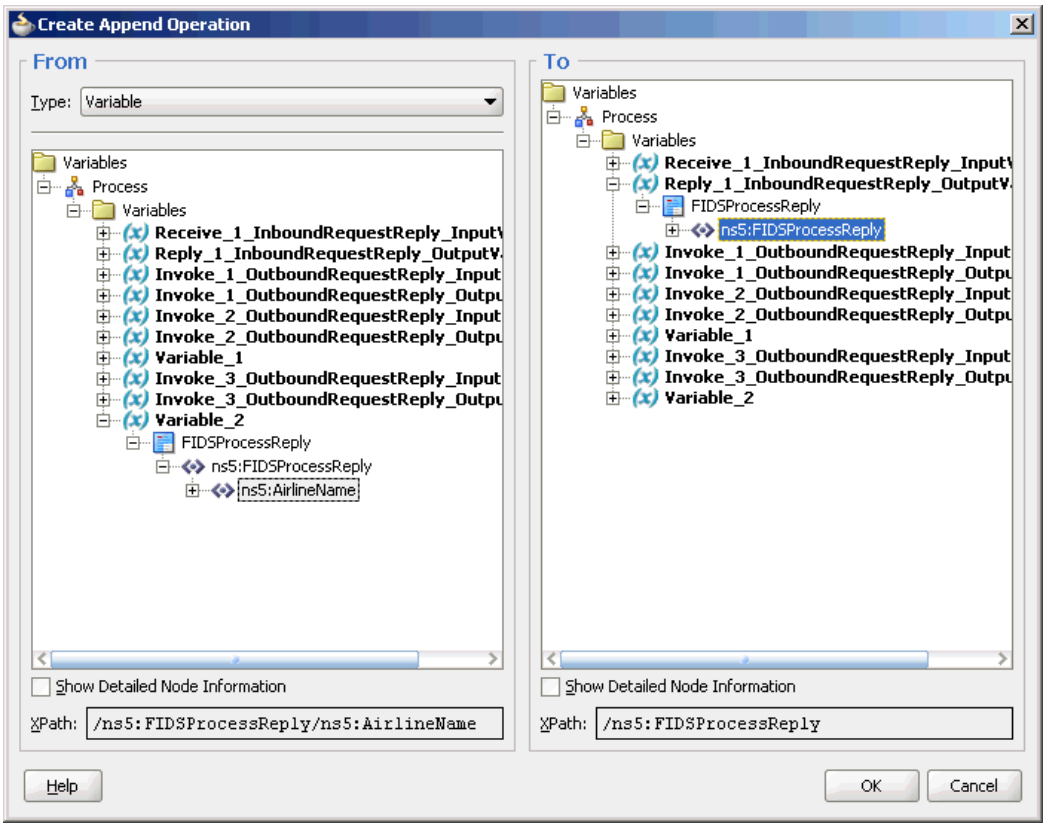
4. Select **Append Operation**. The **Create Append** Operation dialog is displayed.
5. Create an append operation to append the information stored in the temporary variable, `Variable_1`, to the reply variable, `Reply_1_InboundRequestReply_OutputVariable`, as shown in Figure 5-115.

Figure 5-115 The Create Append Operation Dialog



6. Click **OK**. The Assign dialog is displayed.
7. Select **Append Operation**. The **Create Append Operation** dialog is displayed.
8. Create another append operation to append the information stored in the temporary variable, `Variable_2`, to the reply variable, `Reply_1_InboundRequestReply_OutputVariable`, as shown in Figure 5-116.

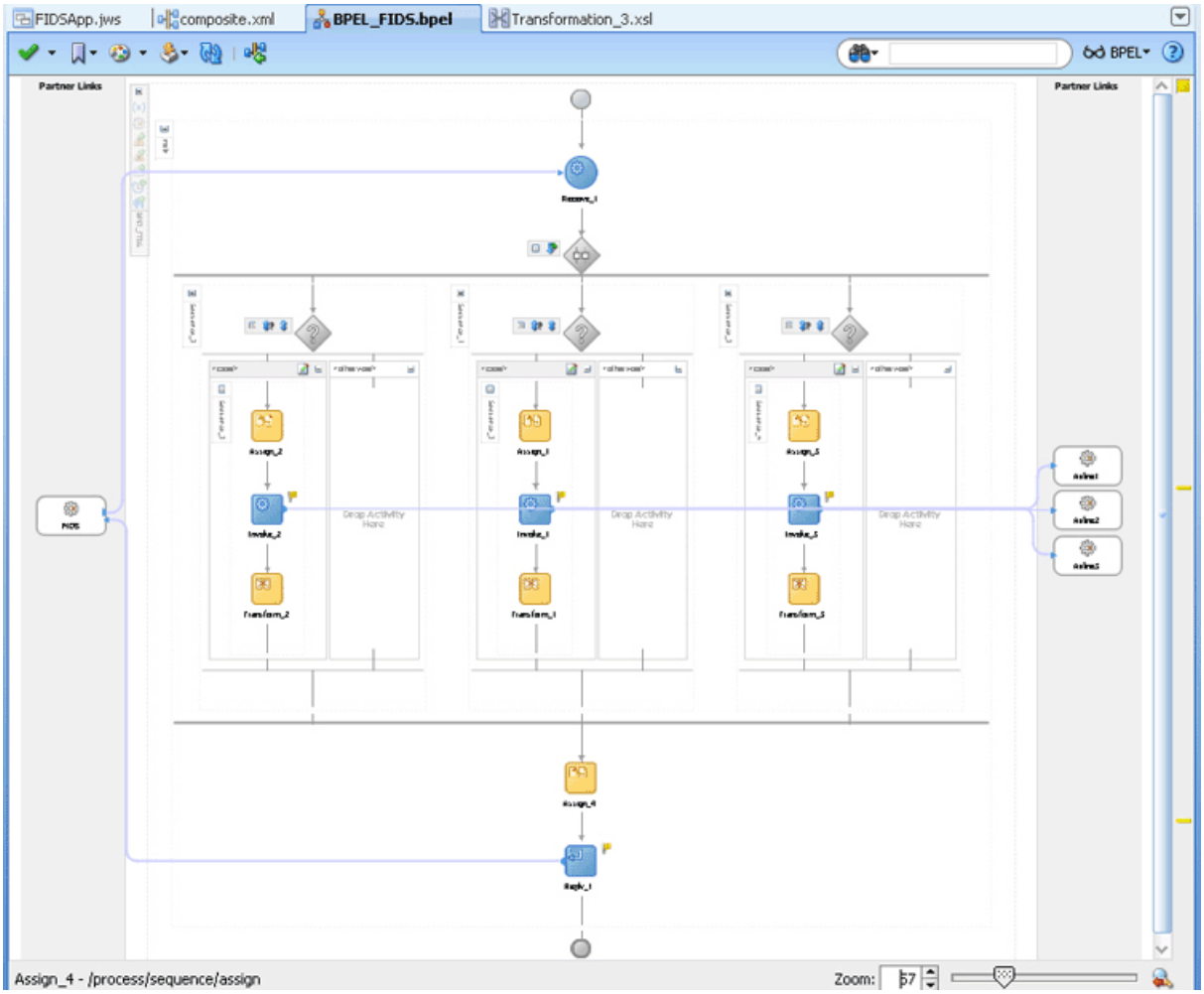
Figure 5-116 The Create Append Operation Dialog



9. Click **OK**. The **Assign** dialog is displayed.

10. Click **OK**. The JDeveloper **BPEL_FIDS.bpel** page is displayed, as shown in Figure 5-117.

Figure 5-117 The JDeveloper - HelloWorldFlow.bpel



11. Click **File**, **Save All**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from .

You must run the Server and Client java programs to test the application. For more information, see the associated README file.

Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.

2. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the **Recent Instances** area.

3. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.

4. Click the Instance ID that you noted in Step 2. The **Flow Trace** page is displayed.

5. Click your BPEL process instance. The **Audit Trail** of the BPEL process instance is displayed.

6. Expand a payload node to view payload details.

7. Click the **Flow** tab to view the process flow. Additionally, click an activity (such as invoke, receive) to view the details of an activity.

Cluster Support for Socket Adapter

A clustered environment is an environment where more than one Managed Server runs in a single machine with a single IP address. If you try to deploy an inbound Oracle Socket Adapter in a clustered environment, one of the Managed Servers displays an error message saying the server's port is already in use.

However, the Socket Adapter provides support for the clustered environment. You can configure the Socket Adapter so it does not throw an exception in a clustered environment.

Note that only in the Vertical Cluster environment can more than one Managed Server run on a single physical machine. In a Horizontal Cluster environment, this is not the case.

Comma separated ports are required for the vertical cluster topology.

Configuring the Socket Adapter for Use in a Clustered Environment

To configure a socket adapter for use in a clustered environment, you must specify the ports that the Socket Adapter uses as a comma-separated list within the appropriate connection factory.

Once you have configured this comma-separated list of ports, the Adapter tries to start the server socket on the first port in the list.

If the Socket Adapter fails to start server socket, it tries with the next port in the list. The Socket Adapter continues to try to start sockets until it starts a server socket on any of the ports.

If the Socket Adapter cannot start server socket on any of the port specified, it throws `oracle.tip.adapter.socket.SocketServerCreationException`

Performance Optmization with Coherence

The Socket Adapter uses the Coherence layer to obtain the port from the port list. Each instance of inbound Socket Adapter on Managed Server fetches a port from the list and makes the fetched port invisible to other instances of the Socket Adapter.

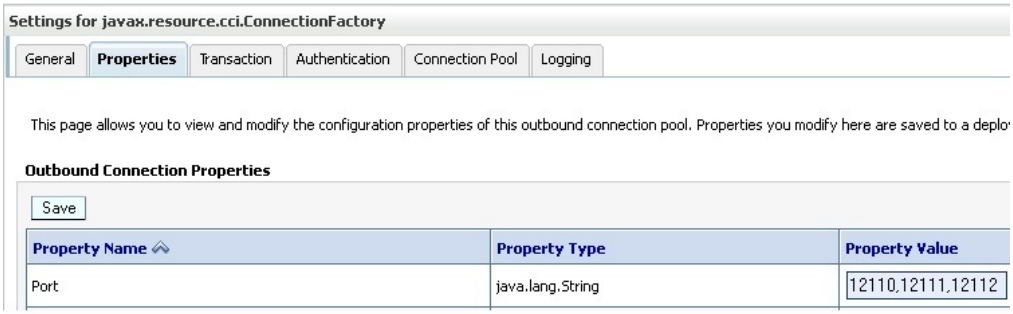
If an inbound Socket Adapter successfully opens the fetched port, the Socket Adapter deletes the fetched port from the list.

Otherwise, the fetched port is visible to other instances of inbound Socket Adapter. The other instances of inbound Socket Adapter on the Managed Server do not attempt to open the same port from the list.

Updating the Port Property of the JNDI Connection Factory to Enable Socket Adapter Support in a Clustered Environment

To enable Socket Adapter support in a clustered environment, you must specify the list of ports for an inbound Socket Adapter. To specify the list, you must open the WebLogic Console and to update the Port property of the JNDI connection factory with comma-separated values of ports as shown in Figure 5-118.

Figure 5-118 Specifying the List of Inbound Ports for the Inbound Socket Adapter



6

Native Format Builder Wizard

This chapter describes the Native Format Builder wizard, which enables you to create native schemas used for translation. It includes use cases and constructs for the schema.

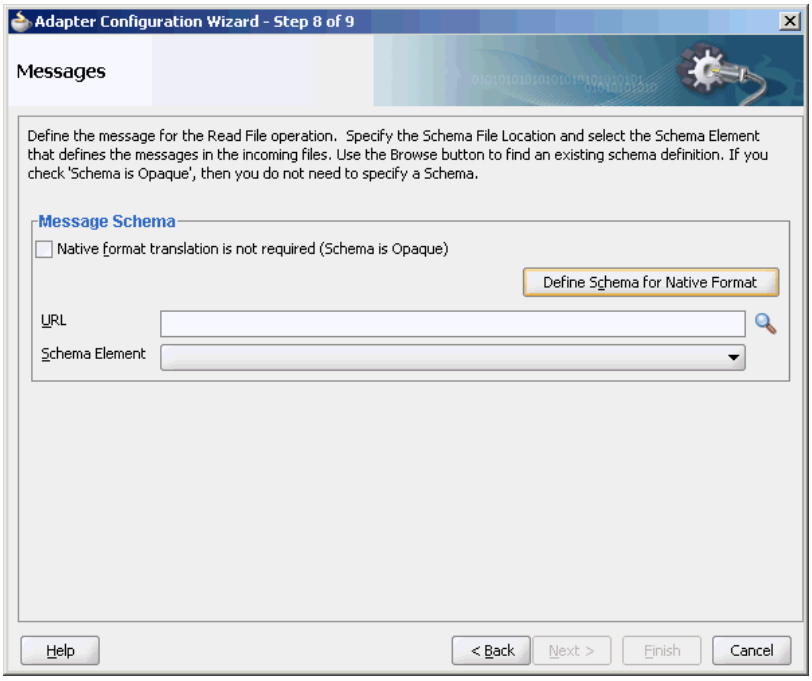
This chapter includes the following sections:

Creating Native Schema Files with the Native Format Builder Wizard

Oracle JCA Adapters are software components that enable the integration between various enterprise information systems (EIS) and the Oracle BPEL Process Manager (Oracle BPEL PM), the Oracle Mediator (Mediator), or the Oracle Service Bus (OSB). Adapters accept native messages in XML or non-XML format and publish them to Oracle BPEL PM or Mediator as XML messages. Adapters can also accept XML messages and convert them back to native EIS format. This translation from native data format to XML and back is performed using a definition file (non-XML schema definition), which itself is defined in XML schema format. The Native Format Builder wizard enables you to sample native data and create the native XSD (NXSD) grammar for translation of native data.

When you click the Define Schema for Native Format button in the Messages page of the Adapter Configuration Wizard shown in Figure 6-1, the Native Format Builder wizard is displayed. The Messages page is the last page that is displayed in the Adapter Configuration Wizard before the Finish page.

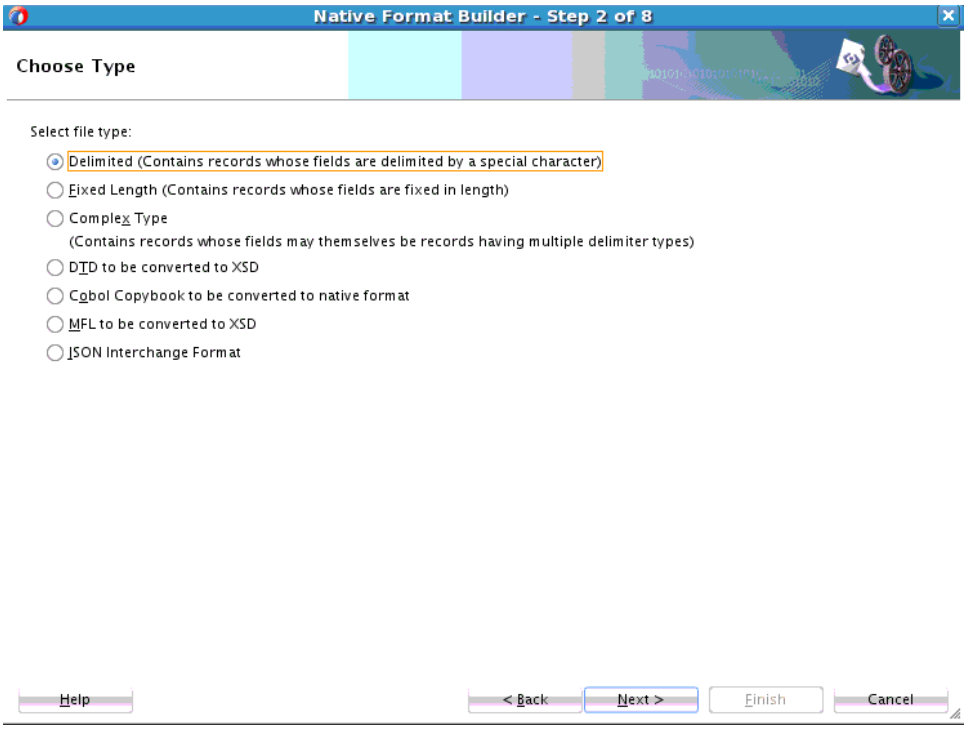
Figure 6-1 Starting the Native Format Builder Wizard



Supported File Formats

The Native Format Builder wizard guides you through the creation of a native schema file from the following file formats shown in Figure 6-2. You must have a sample data file format for the selected type to create a native schema. You can also select the option for editing an existing native schema created with this wizard, except for those generated from a Document Type Definition (DTD) or COBOL Copybook file types. For information on editing the native schema file, see Editing Native Schema Files.

Figure 6-2 Native Format Builder Wizard



Delimited

This option enables you to create native schemas for records, where the fields are separated by a value such as a comma or number sign (#).

Fixed Length (Positional)

This option enables you to create native schemas for records, where all fields are of fixed lengths.

Complex Type

This option enables you to create native schema for records, where the fields may themselves be records having multiple delimiter types.

DTD

This option enables you to generate native schema from the user-supplied DTD, which contains information about the structure of an XML document.

COBOL Copybook

This option enables you to generate native schema from the user-supplied COBOL Copybook definition.

A COBOL mainframe application typically uses a COBOL Copybook file to define its data layout. The converter creates a native schema from a COBOL Copybook so that the runtime translator can parse the associated data file.

A COBOL Copybook is typically a collection of group items (structures). These group items contain other items, which can be groups or elementary items. Elementary items are items that cannot be further subdivided. For example:

```
01 Purchase-Order
        05 Buyer
                10 BuyerName PIC X(5) USAGE DISPLAY.
        04 Seller
                08 SellerName PICTURE XXXXX.
```

`Purchase-order` is a group item with two child group items (`Buyer`, `Seller`). The numbers `01`, `05`, `04`, and so on indicate the level of the group (that is, the hierarchy of data within that group).

Groups can be defined that have different level-numbers for the same level in the hierarchy. For example, `Buyer` and `Seller` have different level numbers, but are at the same level in the hierarchy. A group item includes all group and elementary items that follow it until a level number less than or equal to the level number of that group is encountered.

Each of the group items (`Buyer` and `Seller`) has a child elementary item. The `PIC` or `PICTURE` clause defines the data layout. For example, `BuyerName` defines an alphanumeric type of size equal to five characters. `SellerName` has the same data layout as `BuyerName`.

Group items in COBOL can be mapped to elements in XML schema with the `complexType` type. Similarly, elementary items can be mapped to elements of type `simple type` with certain native format annotations to help the runtime translator parse the corresponding data file. For example, the `Buyer` item can be mapped to the following definition:

```
<!--COBOL declaration : 05 Buyer-->
<element name="Buyer">
   <complexType>
      <sequence>
         <!--COBOL declaration : 10 Name PIC X(5)-->
         <element name="Name" type="string" nxsd:style="fixedLength"
            nxsd:padStyle="tail" nxsd:paddedBy=" " nxsd:length="5"/>
         </sequence>
   </complexType>
</element>
```

User Inputs

You are expected to provide the following information:

- Target namespace for the native schema to be generated
- Character set of the host computer on which the data file was generated. By default, this is set to EBCDIC (`ebcdic-cp-us`).
- Byte order of the host computer on which the data file was generated. By default, this is set to big-endian.
- Record delimiter, which is typically the new line character, or no delimiter, or any user-supplied string.
- Container tag name for generated native schema. By default, this is set to `Root-Element`.

COBOL Clauses

Table 6-1 describes COBOL clauses. The numeric types covered in Table 6-1 are stored as one character per digit. Support for clauses is defined as follows:

- Y indicates that the clause is supported.
- N indicates that the clause is not supported.
- I indicates that the clause is ignored.

Table 6-1 COBOL Clauses (Numeric Types Stored as One Character Per Digit)

COBOL Clause	Design-Time Support	Runtime Support	Supported Synonyms	Comments
PIC X(*n*)	Y	Y	XXX…	Alphanumeric – An allowable character from the character set of the computer. Each X corresponds to one byte.
PIC A(*n*)	Y	Y	AA…	Alphabetic – Any letter of the alphabet or space. Each A corresponds to one byte.
PIC 9(*n*) DISPLAY	Y	Y	9999…	Any character position that contains a numeral. Each nine is counted in the size of the item.
OCCURS *n* TIMES	Y	Y		Fixed-length array
JUSTIFIED	Y	Y		For A and X types. Right justifies with the space pad. Data is aligned at the right-most character position.
REDEFINES	Y	Y		Allows the same computer memory area to be described by different data items.
PIC 9(*m*)V9(*n*) DISPLAY	Y	Y		Size = *n*+*m* bytes
OCCURS DEPENDING ON	Y	Y		NA
BLANK WHEN ZERO	I	I		Ignored
RENAMES	N	N		This is rarely seen in COBOL Copybooks
INDEX	N	N		Four-byte index
SYNCHRONIZED	I	I	SYNC	NA
POINTER	N	N		NA
PROCEDURE-POINTER				NA
FILLER	Y	Y		NA

The numeric types described in Table 6-1 are stored as one character per digit. Table 6-2 describes the numeric types that are stored in a more efficient manner.

Table 6-2 COBOL Clauses (Numeric Types Stored More Efficiently)

COBOL Clause	Design-Time Support	Runtime Support	Supported Synonyms	Comments
`USAGE [IS]`	Y	Y		Both these keywords are optional.
`PIC 9(`*n*`) COMP`	Y	Y	`COMPUTATIONAL, BINARY, COMP-4`	Length varies with *n*: • *n* = `1-4` (2 bytes) • *n* = `5-9` (4 bytes) • *n* = `10-18` (8 bytes)
`COMP-1`	Y	Y	`COMPUTATIONAL-1`	Single precision, floating point number that is four bytes long.
`COMP-2`	Y	Y	`COMPUTATIONAL-2`	Double precision, floating point number that is eight bytes long.
`PIC 9(`*n*`) COMP-3`	Y	Y	`PACKED-DECIMAL, COMPUTATIONAL-3`	Two digits are stored in each byte. An additional half byte at the end is allocated for the sign, even if the value is unsigned.
`PIC 9(`*n*`) COMP-4`	Y	Y	`COMPUTATIONAL-4`	Treated the same as a `COMP` type and given its own data type for customizing requirements.
`PIC 9(`*n*`) COMP-5`	N	N		Capacity of the native binary representation.
`PIC S9(`*n*`) DISPLAY`	Y	Y	`PIC S99…`	Sign nibble in the right-most zone by default. `S` is not counted in the size.
`PIC S9(`*n*`) COMP`	Y	Y		Same as `COMP`. Negative numbers are represented as two's complement.
`PIC S9(`*n*`) COMP-3`	Y	Y		NA
`PIC 9(`*m*`)V9(`*n*`) COMP`	Y	Y		Length is the same as `COMP`.
`PIC 9(`*m*`)V9(`*m*`) COMP-3`	Y	Y		Length = `Ceiling ((`*n*`+`*m*`+1)/2)`

The following clauses can be added to impact the sign position.

- `SIGN IS LEADING`

  Used with signed zoned numerics.

- `SIGN IS TRAILING`

  Used with signed zoned numerics.

- `SIGN IS LEADING SEPARATE`

The character S is counted in the size.

- `SIGN IS TRAILING SEPARATE`

  The character S is counted in the size.

> **Note:**
>
> These assume that the numerics are stored using IBM COBOL format. If these are generated for other platforms with different data storage formats, then a custom data handler for that type must be written.

Picture Editing Types

Table 6-3 describes picture editing types.

Table 6-3 Edited Pictures

Edited Pictures	Supported Editing Types	Unsupported Editing Types
Edited alphanumeric	Simple Insertion: B(blank) `0 / ,`	
Edited float numeric	Special insertion: `.` (period)	
Edited numeric	• Simple Insertion: B(blank) `0 / ,` • Special insertion: `.` (period) • Fixed Insertion: `cs + - CR DB` (Inserts a symbol at the beginning or end)	• Floating Insertion: `cs + -` • Zero suppression: `Z *` • Replacement insertion: `Z * + - c`

Edited pictures are more for presentation purposes and are rarely seen in data files. It is assumed that the editing symbols are also present in the data. For example, if you have:

```
05   AMOUNT     PIC 999.99
```

then, this field is six bytes wide and has a decimal point in the data.

Simple, special, and fixed insertions are handled by this method. Floating insertion, zero suppression, and replacement insertion are not supported.

MFL to be Converted into XSD

Enables you to convert an MFL file into XSD format. For more information, see Converting an MFL Format File to Schema Format.

JSON Interchange Format

Generates an NXSD schema from a JSON sample file or JSON file at a specified location. JSON is a text-based, open standard for human-readable data interchange. JSON comes from the JavaScript scripting language for representing simple data structures and associative arrays, called objects.

If you select this option, The Native Format Builder Wizard displays the JSON File Description screen. Here you can enter:

- File Name—The name of the JSON file.
- Target Namespace—Displays the target Namespace.
- Root Element—Specifies the JSON root level.
- Character Set —Select the character set to use.

For more information on using JSON, see "Integrating REST Operations in SOA Composite Applications"Developing SOA Applications with Oracle SOA Suite.

Editing Native Schema Files

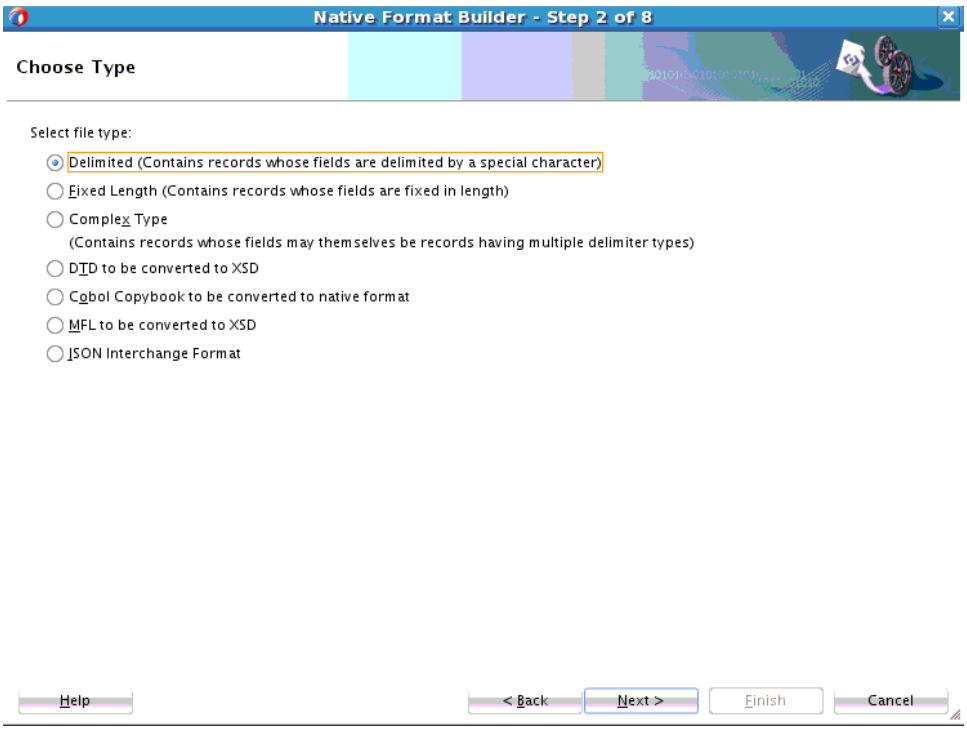
You can edit an existing native schema generated using the Native Format Builder wizard by sampling a delimited, fixed length, or complex type file. To edit an existing native schema select the **Edit existing** option in the Choose Type page of the Native Format Builder wizard, and click **Browse** to navigate to the location of the existing schema file and then select the native schema file that must be edited. The Native Format Builder wizard guides you through the editing of the native schema file.

Note:

You cannot edit native schemas generated from a Document Type Definition (DTD) or COBOL Copybook file types.

Figure 6-3 shows the Native Format Builder - Choose Type page with the Edit existing option selected.

Figure 6-3 The Native Format Builder Wizard - Choose Type Page



Before you edit a native schema file, you must ensure that the sample file specified in the annotation within the schema exists. This annotation is automatically added when the native schema is generated the first time from the sample file.

The example below shows the annotation that is generated:

Example - Generated Annotation

```
<xsd:annotation>
     <xsd:appinfo>NXSDSAMPLE=/scratch/bob/sample.txt</xsd:appinfo>
     <xsd:appinfo>USEHEADER=true</xsd:appinfo>
</xsd:annotation>
```

For example, if the specified sample file path in the annotation is `<!--NXSDWIZ:C:\Temp\Book1Out.csv:-->` and if the file is not located at the path specified, then the wizard displays an error.

Native Schema Constructs

This section provides an overview of the various constructs of native schema used to translate the native format data to XML and also explains the usage of these native schema constructs.

This section includes the following topics:

- Understanding Native Schema Constructs
- Using Native Schema Constructs

Understanding Native Schema Constructs

Table 6-4 shows the constructs applicable only on the `<schema>` tag.

Table 6-4 Constructs Applicable Only on the <schema> Tag

Construct	Description
`byteOrder`	The byte order of the native data as `bigEndian` or `littleEndian`.
`encoding`	The encoding in which the actual data is stored. UTF-8 is typically recommended for interoperability and Unicode support. You can specify any legal encoding supported by the Java runtime environment. For a complete listing of supported encodings, visit `http://download.oracle.com/javase/6/docs/technotes/guides/intl/encoding.doc.html`. You can specify the encoding in the (N)XSD associated with the adapter proxy meta data. For example, `nxsd:encoding="iso-8859-1`
`nxsd:alwaysQuote`	Set to true if quotes must be forced around native non-xml data in the outbound.
`headerLines`	A positive integer specifying the number of lines to be skipped, before translating the native data.
`headerLinesTerminatedBy`	Skip until the specified string, before translating the native data.
`standalone`	If declared, adds the standalone attribute in the XML declaration prolog of the translated XML, with the actual value as that specified in `nxsd:standalone`. Allowed values are `true` and `false`.

Table 6-4 (Cont.) Constructs Applicable Only on the <schema> Tag

Construct	Description
`stream`	Whether the data is stored as characters or bytes. Allowed values are `CHARS` and `BYTES`.
`uniqueMessageSeparator`	String specifying the unique message separator in the native data, in a batch of messages.
`version`	The type of native data. Possible values are `NXSD`, `DTD`, `XSD`, and `OPAQUE`.
`xmlversion`	If declared, adds the XML declaration prolog to the translated XML with the actual value as that specified in `nxsd:xmlversion`. Allowed values are `1.0` and `1.1`.
`outboundHeader`	String specifying the header value to be inserted in the outbound message.
`dataLines`	Integer specifying the number of lines to process in the native file.
`fieldValidation`	If set to `true`, then translator performs data type validation on the tokens read from the native. The field`Validation` construct is supported for built in simple types only.
`validation`	If set to `true`, then the translator performs result validation both on the inbound and outbound.
`validateNxsd`	If set to `true`, then a thorough native grammar validation is performed. This construct is switched off by default and must be switched off in production for better performance.
`useArrayIdentifiers`	If set to `true`, then it optimizes the native framework for handling array identifiers. This may result in a performance hit for very large payloads. By default, arrayIdentifiers are not supported.
`parseBom`	If set to `true`, then the byte order mark is looked for in the native stream and encoding is derived from this.
`encodeLineTerminators`	If set to `true`, then the semantic interpretation of `${eol}` is `\r \n` instead of `\n`.

Table 6-5 shows the constructs applicable on all tags other than the `<schema>` tag.

Table 6-5 Constructs Applicable On All Tags Other Than the <schema> Tag

Construct	Description
`arrayIdentifierLength`	The length of the array being stored in the native data occupying the specified length

Table 6-5 (Cont.) Constructs Applicable On All Tags Other Than the <schema> Tag

Construct	Description
`arrayLength`	The value of this construct is used as the length of the array, which can also be a variable resolved to a valid number. This value overrides any `minOccurs` and `maxOccurs` attributes of the particle where it is specified. Use this feature as follows: `nxsd:style="array" nxsd:arrayLength="10"` This indicates that the array length is `10`.
`arrayTerminatedBy`	The last item in the array being terminated by the specified string
`assign`	Assigns a value to the variable that is declared
`cellSeparatedBy`	The cells of the array in the native data being separated by the specified string
`choiceCondition`	Either `fixedLength` or `terminated`
`conditionValue`	Matches the string read from the native stream for the `choiceCondition` construct, against the specified string in the `conditionValue` construct
`dataLines`	The value specified in this construct is used to translate only a portion of the data and not the entire data.
`dateFormat`	A valid Java date format representing the date in the native data
`identifierLength`	The number of characters and bytes in which the actual length of the data is stored
`itemSeparatedBy`	The items in the list being separated by the specified string
`leftSurroundedBy`, `rightSurroundedBy`	The native data surrounded
`length`	The length of the native data to be read. Used with fixed-length style.
`listTerminatedBy`	The last item in the list being terminated by the specified string
`lookAhead`	Looks for a match ahead of the current position in the input stream. If a match is found, then the node on which this construct is specified is processed; otherwise, it is skipped. Use this feature as follows: `nxsd:lookAhead="20" nxsd:lookFor="abc"` This indicates to skip `20` characters and look for the string `abc` starting from that location. If this is found, then the node is processed; otherwise, it is skipped.
`paddedBy`	The string used for padding
`padStyle`	`head`, `tail`, or `none`

Table 6-5 (Cont.) Constructs Applicable On All Tags Other Than the <schema> Tag

Construct	Description
`quotedBy`	The native data being quoted by the specified string. By default, the specified string is `&quot;` ("). If your data includes this character, you must override this default even if the field is not quoted. For more information, see **Native Data Format to Be Translated: Data Includes Default Quote Character** under Defining Terminated Data.
`skip`	Skips the specified number of bytes or characters
`skipLines`	Skips the number of lines specified
`skipUntil`	Skips until the string specified
`startsWith`	Looks for the specified string in the native data. If it exists, then proceeds with the element where it is specified; otherwise, skips and processes the next element.
`style`	The style used to read the native data from the input stream. Allowed values are `fixedLength`, `surrounded`, `terminated`, `list`, and `array`.
`surroundedBy`	The native data being surrounded by the specified string.
`terminatedBy`	The native data being terminated by the string specified.
`variable`	Declares a single variable.
`variables`	Declares a set of variables or assigns the declared variables a value.

Using Native Schema Constructs

This section includes the following topics:

- Defining Fixed-Length Data
- Defining Terminated Data
- Defining Surrounded Data
- Defining Lists
- Defining Arrays
- Conditional Processing
- Defining Dates
- Using Variables
- Defining Prefixes and Suffixes
- Defining Skipping Data
- Defining fixed and default Values

- Defining write
- Defining LookAhead
- Defining outboundHeader
- Defining Complex Condition in conditionValue
- Defining Complex Condition in choiceCondition
- Defining dataLines
- Defining Date Formats with Time Zone
- Implementing Validation During Translation
- Processing Files with BOM

Defining Fixed-Length Data

Fixed-length data in the native format can be defined in the native schema by using the fixed-length style. There are three types of fixed length:

- With padding
- Without padding
- With the actual length also being read from the native data

Native Data Format to be Translated: With Padding

The actual data may be less than the length specified. In this case, you can specify `paddedBy` and `padStyle` as `head` or `tail`. When the data is read, the pads are trimmed accordingly. The following is a sample native data to be translated:

```
GBP*UK000012550.00
```

Native Schema: With Padding

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="fixedlength">
  <complexType>
    <sequence>
      <element name="currency_code" nxsd:style="fixedLength" nxsd:length="4"
         nxsd:padStyle="tail" nxsd:paddedBy="*">
        <simpleType>
          <restriction base="string">
            <maxLength value="4" />
          </restriction>
        </simpleType>
      </element>
      <element name="country_code" nxsd:style="fixedLength" nxsd:length="2"
         nxsd:padStyle="none">
        <simpleType>
```

```
          <restriction base="string">
            <length value="2" />
          </restriction>
        </simpleType>
      </element>
      <element name="to_usd_rate" nxsd:style="fixedLength" nxsd:length="12"
         nxsd:padStyle="head" nxsd:paddedBy="0">
        <simpleType>
          <restriction base="string">
            <maxLength value="12" />
          </restriction>
        </simpleType>
      </element>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema: With Padding

```
<fixedlength xmlns="http://www.oracle.com/ias/processconnect">
   <currency_code>GBP</currency_code>
   <country_code>UK</country_code>
   <to_usd_rate>12550.00</to_usd_rate>
</fixedlength>
```

Native Data Format to be Translated: Without Padding

To define a fixed-length data in native schema, you can use the fixed-length style. In case the actual data is less than the length specified, the white spaces are not trimmed. The following is a sample native data to be translated:

```
GBP*UK000012550.00
```

Native Schema: Without Padding

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="fixedlength">
  <complexType>
    <sequence>
      <element name="currency_code" nxsd:style="fixedLength" nxsd:length="4">
        <simpleType>
          <restriction base="string">
            <maxLength value="4" />
          </restriction>
        </simpleType>
      </element>
      <element name="country_code" nxsd:style="fixedLength" nxsd:length="2">
        <simpleType>
          <restriction base="string">
            <length value="2" />
          </restriction>
```

```
        </simpleType>
      </element>
      <element name="to_usd_rate" nxsd:style="fixedLength" nxsd:length="12">
        <simpleType>
          <restriction base="string">
            <maxLength value="12" />
          </restriction>
        </simpleType>
      </element>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema: Without Padding

```
<fixedlength xmlns="http://www.oracle.com/ias/processconnect">
   <currency_code>GBP*</currency_code>
   <country_code>UK</country_code>
   <to_usd_rate>000012550.00</to_usd_rate>
</fixedlength>
```

Native Data Format to be Translated: Actual Length Also Being Read from the Native Data

When the length of the data is also stored in the native stream, this style is used to first read the length, and subsequently read the data according to the length read. The following is a sample native data to be translated:

```
03joe13DUZac.1HKVmIY
```

Native Schema: Actual Length Also Being Read from the Native Data

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="fixedlength">
  <complexType>
    <sequence>
      <element name="user" type="string" nxsd:style="fixedLength"
         nxsd:identifierLength="2" />
      <element name="encr_user" type="string" nxsd:style="fixedLength"
         nxsd:identifierLength="2" />
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema: Actual Length also Being Read from the Native Data

```
<fixedlength xmlns="http://www.oracle.com/ias/processconnect">
   <user>joe</user>
   <encr_user>DUZac.1HKVmIY</encr_user>
</fixedlength>
```

Defining Terminated Data

This format is used when the terminating mark itself is supposed to be treated as part of the actual data and not as a delimiter. When it is not clear whether the mark is part of actual data or not, you can use `nxsd:quotedBy` to be safe. Specifying `nxsd:quotedBy` means that the corresponding native data may or may not be quoted. If it is quoted, then the actual data is read from the begin quotation to the end quotation as specified in `nxsd:quotedBy`. Otherwise, it is read until the `terminatedBy` character is found.

By default, the terminating mark is `&quot;` ("). If your data includes this character, you must override this default even if the field is not quoted. For more information, see **Native Data Format to be Translated: Data Includes Default Quote Character** below.

Examples for the Optionally quoted, Not quoted, and Includes default quote character scenarios are provided in the following sections:

Native Data Format to be Translated: Optionally Quoted

The following is a sample native data to be translated:

```
Fred,"2 Old Street, Old Town,Manchester",20-08-1954,0161-499-1718
```

Native Schema: Optionally Quoted

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="terminated">
  <complexType>
    <sequence>
      <element name="PersonName" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," />
      <element name="Address" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," />
      <element name="Telephone" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
    </sequence>
  </complexType>
</element>
```

Translated XML Using the Native Schema: Optionally Quoted

```
<terminated xmlns="http://www.oracle.com/ias/processconnect">
   <PersonName>Fred</PersonName>
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
```

```
   <Telephone>0161-499-1718</Telephone>
</terminated>
```

Native Data Format to be Translated: Not Quoted

This is used when the data is terminated by a particular string or character. The following is a sample native data to be translated:

```
1020,16,18,,1580.00
```

Native Schema: Not Quoted

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="terminated">
  <complexType>
    <sequence>
      <element name="product" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," />
      <element name="ordered" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," />
      <element name="inventory" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="," />
      <element name="backlog" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="," />
      <element name="listprice" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema: Not Quoted

```
<terminated xmlns="http://www.oracle.com/ias/processconnect">
   <product>1020</product>
   <ordered>16</ordered>
   <inventory>18</inventory>
   <backlog></backlog>
   <listprice>1580.00</listprice>
</terminated>
```

Native Data Format to be Translated: Data Includes Default Quote Character

The following is a sample native data to be translated:

```
aaa,"bbbbb,[cccc
```

In this case, fields are terminated by commas, the " character is part of the data in the second field, and the [character is part of the data in the third field.

Because the default `nxsd:quotedBy` terminating mark is `&quot;` ("), the Oracle File Adapter fails to translate field two even if you specify that this field is terminated by a

comma character. To successfully translate this data, you must override the default `nxsd:quotedBy` terminating mark to any character that is not be part of the data for this field. In this example, you override the default `nxsd:quotedBy` terminating mark to `<` (<) because this character never appears in field two:

```
<element name="FieldTwo" type="string" nxsd:style="terminated" nxsd:terminatedBy=","
nxsd:quotedBy="&lt;"/>
```

By contrast, for field three, you must only specify `nxsd:terminatedBy=","` because the `[` character does not conflict with the default `nxsd:quotedBy` terminating mark:

```
<element name="FieldThree" type="string" nxsd:style="terminated"
nxsd:terminatedBy="," />
```

Native Schema: Data Includes Default Quote Character

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="terminated">
  <complexType>
    <sequence>
      <element name="FieldOne" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," />
      <element name="FieldTwo" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," nxsd:quotedBy="&lt;"/>
      <element name="FieldThree" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," />
    </sequence>
  </complexType>
</element>
```

Translated XML Using the Native Schema: Data Includes Default Quote Character

```
<terminated xmlns="http://www.oracle.com/ias/processconnect">
   <FieldOne>aaa</FieldOne>
   <FieldTwo>"bbbbb</FieldTwo>
   <FieldThree>[cccc</FieldThree>
</terminated>
```

Defining Surrounded Data

This is used when the native data is surrounded by a mark.

The following are types of surrounded data:

- Left and right surrounding marks are different.
- Left and right surrounding marks are the same.

Native Data Format to be Translated: Left and Right Surrounding Marks are Different

The following is a sample native data to be translated for which the left and the right surrounding marks are different:

```
(Ernest Hemingway Museum){Whitehead St.}
```

Native Schema: Left and Right Surrounding Marks are Different

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        xmlns:tns="http://www.oracle.com/ias/processconnect"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">
<element name="limstring">
  <complexType>
    <sequence>
      <element name="Landmark" type="string" nxsd:style="surrounded"
nxsd:leftSurroundedBy="(" nxsd:rightSurroundedBy=")" />
      <element name="Street" type="string" nxsd:style="surrounded"
nxsd:leftSurroundedBy="{" nxsd:rightSurroundedBy="}" />
    </sequence>
  </complexType>
</element>
</schema>
```

Translated XML Using the Native Schema: Left and Right Surrounding Marks are Different

```
<limstring xmlns="http://www.oracle.com/ias/processconnect">
   <Landmark>Ernest Hemingway Museum</Landmark>
   <Street>Whitehead St.</Street>
</limstring>
```

Native Data Format to be Translated: Left and Right Surrounding Marks are the Same

The following is a sample native data to be translated for which the left and the right surrounding marks are the same:

```
.FL..Florida Keys.+Key West+
```

Native Schema: Left and Right Surrounding Marks are the Same

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        xmlns:tns="http://www.oracle.com/ias/processconnect"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">
<element name="limstring">
  <complexType>
    <sequence>
      <element name="State" type="string" nxsd:style="surrounded"
nxsd:surroundedBy="."/>
      <element name="Region" type="string" nxsd:style="surrounded"
nxsd:surroundedBy="." />
      <element name="City" type="string" nxsd:style="surrounded"
```

```
nxsd:surroundedBy="+" />
    </sequence>
  </complexType>
</element>
</schema>
```

Translated XML Using the Native Schema: Left and Right Surrounding Marks are the Same

```
<limstring xmlns="http://www.oracle.com/ias/processconnect">
   <State>FL</State>
   <Region>Florida Keys</Region>
   <City>Key West</City>
</limstring>
```

Defining Lists

This format applies to lists with the following characteristics:

- All Items Separated by the Same Mark_ but the Last Item Terminated by a Different Mark (Bounded)
- All Items Separated by the Same Mark_ Including the Last Item (Unbounded)

All Items Separated by the Same Mark, but the Last Item Terminated by a Different Mark (Bounded)

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to be Translated

```
125,200,255
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        xmlns:tns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">


<element name="list" type="tns:Colors" />


<complexType name="Colors" nxsd:style="list" nxsd:itemSeparatedBy=","
   nxsd:listTerminatedBy="${eol}">

  <sequence>
    <element name="Red" type="string" />
    <element name="Green" type="string" />
    <element name="Blue" type="string" />
  </sequence>
</complexType>

</schema>
```

Translated XML Using the Native Schema

```
<list xmlns="http://www.oracle.com/ias/processconnect">
   <Red>125</Red>
   <Green>200</Green>
   <Blue>255</Blue>
</list>
```

All Items Separated by the Same Mark, Including the Last Item (Unbounded)

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to be Translated

```
configure;startup;runtest;shutdown;
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        xmlns:tns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="list" type="tns:CommandSet" />

<complexType name="CommandSet" nxsd:style="list" nxsd:itemSeparatedBy=";">
  <sequence>
    <element name="Cmd1" type="string" />
    <element name="Cmd2" type="string" />
    <element name="Cmd3" type="string" />
    <element name="Cmd4" type="string" />
  </sequence>
</complexType>

</schema>
```

Translated XML Using the Native Schema

```
<list xmlns="http://www.oracle.com/ias/processconnect">
   <Cmd1>configure</Cmd1>
   <Cmd2>startup</Cmd2>
   <Cmd3>runtest</Cmd3>
   <Cmd4>shutdown</Cmd4>
</list>
```

Defining Arrays

This is for an array of complex types where the individual cells are separated by a separating character and the last cell of the array is terminated by a terminating character.

The following are examples of array types:

- All Cells Separated by the Same Mark_ but the Last Cell Terminated by a Different Mark (Bounded)
- All Cells Separated by the Same Mark_ Including the Last Cell (Unbounded)
- Cells Not Separated by Any Mark_ but the Last Cell Terminated by a Mark (Bounded)
- The Number of Cells Being Read from the Native Data
- Explicit Array Length

All Cells Separated by the Same Mark, but the Last Cell Terminated by a Different Mark (Bounded)

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to be Translated

```
"Smith, John","1 Old Street, Old Town, Manchester",,"0161-499-1717".
Fred,"2 Old Street, Old Town,Manchester","20-08-1954","0161-499-1718".
"Smith, Bob",,,0161-499-1719.#
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD"
>
<element name="array">
  <complexType>
    <sequence>
      <element name="Member" maxOccurs="unbounded"
               nxsd:style="array" nxsd:cellSeparatedBy="${eol}"
 nxsd:arrayTerminatedBy="#">
        <complexType>
          <sequence>
            <element name="Name" type="string" nxsd:style="terminated"
 nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Address" type="string" nxsd:style="terminated"
 nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="DOB" type="string" nxsd:style="terminated"
 nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Telephone" type="string" nxsd:style="terminated"
 nxsd:terminatedBy="." nxsd:quotedBy='"'/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema

```
<array xmlns="http://www.oracle.com/ias/processconnect">
   <Member>
      <Name>Smith, John</Name>
      <Address>1 Old Street, Old Town, Manchester</Address>
      <DOB></DOB>
      <Telephone>0161-499-1717</Telephone>
   </Member>
   <Member>
      <Name>Fred</Name>
      <Address>2 Old Street, Old Town,Manchester</Address>
      <DOB>20-08-1954</DOB>
      <Telephone>0161-499-1718</Telephone>
   </Member>
   <Member>
      <Name>Smith, Bob</Name>
      <Address></Address>
      <DOB></DOB>
      <Telephone>0161-499-1719</Telephone>
   </Member>
</array>
```

All Cells Separated by the Same Mark, Including the Last Cell (Unbounded)

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to be Translated

```
"Smith, John","1 Old Street, Old Town, Manchester",,"0161-499-1717".
Fred,"2 Old Street, Old Town,Manchester","20-08-1954","0161-499-1718".
"Smith, Bob",,,0161-499-1719.
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="array">
  <complexType>
    <sequence>
      <element name="Member" maxOccurs="unbounded"
               nxsd:style="array" nxsd:cellSeparatedBy="\r\n">
        <complexType>
          <sequence>
            <element name="Name" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Address" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="DOB" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Telephone" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="." nxsd:quotedBy='"'/>
          </sequence>
        </complexType>
      </element>
```

```
    </sequence>
  </complexType>
</element>


</schema>
```

Translated XML Using the Native Schema

```
<array xmlns="http://www.oracle.com/ias/processconnect">
   <Member>
      <Name>Smith, John</Name>
      <Address>1 Old Street, Old Town, Manchester</Address>
      <DOB></DOB>
      <Telephone>0161-499-1717</Telephone>
   </Member>
   <Member>
      <Name>Fred</Name>
      <Address>2 Old Street, Old Town,Manchester</Address>
      <DOB>20-08-1954</DOB>
      <Telephone>0161-499-1718</Telephone>
   </Member>
   <Member>
      <Name>Smith, Bob</Name>
      <Address></Address>
      <DOB></DOB>
      <Telephone>0161-499-1719</Telephone>
   </Member>
</array>
```

Cells Not Separated by Any Mark, but the Last Cell Terminated by a Mark (Bounded)

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to be Translated

```
"Smith, John","1 Old Street, Old Town, Manchester",,"0161-499-1717"
Fred,"2 Old Street, Old Town,Manchester","20-08-1954","0161-499-1718"
"Smith, Bob",,,0161-499-1719
#
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="array">
  <complexType>
    <sequence>
      <element name="Member" maxOccurs="unbounded"
               nxsd:style="array" nxsd:arrayTerminatedBy="#">
        <complexType>
          <sequence>
            <element name="Name" type="string" nxsd:style="terminated"
```

```
                nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Address" type="string" nxsd:style="terminated"
                nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="DOB" type="string" nxsd:style="terminated"
                nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Telephone" type="string" nxsd:style="terminated"
                nxsd:terminatedBy="\r\n" nxsd:quotedBy='"'/>          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema

```
<array xmlns="http://www.oracle.com/ias/processconnect">
   <Member>
      <Name>Smith, John</Name>
      <Address>1 Old Street, Old Town, Manchester</Address>
      <DOB></DOB>
      <Telephone>0161-499-1717</Telephone>
   </Member>
   <Member>
      <Name>Fred</Name>
      <Address>2 Old Street, Old Town,Manchester</Address>
      <DOB>20-08-1954</DOB>
      <Telephone>0161-499-1718</Telephone>
   </Member>
   <Member>
      <Name>Smith, Bob</Name>
      <Address></Address>
      <DOB></DOB>
      <Telephone>0161-499-1719</Telephone>
   </Member>
</array>
```

The Number of Cells Being Read from the Native Data

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to be Translated

```
3"Smith, John","1 Old Street, Old Town, Manchester",,"0161-499-1717"
Fred,"2 Old Street, Old Town,Manchester","20-08-1954","0161-499-1718"
"Smith, Bob",,,0161-499-1719
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="arrayidentifierlength">
```

```
  <complexType>
    <sequence>
      <element name="Member" maxOccurs="unbounded" nxsd:style="array"
          nxsd:arrayIdentifierLength="1">
        <complexType>
          <sequence>
            <element name="Name" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Address" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="DOB" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Telephone" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="\r\n" nxsd:quotedBy='"'/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema

```
<arrayidentifierlength xmlns="http://www.oracle.com/ias/processconnect">
   <Member>
      <Name>Smith, John</Name>
      <Address>1 Old Street, Old Town, Manchester</Address>
      <DOB></DOB>
      <Telephone>0161-499-1717</Telephone>
   </Member>
   <Member>
      <Name>Fred</Name>
      <Address>2 Old Street, Old Town,Manchester</Address>
      <DOB>20-08-1954</DOB>
      <Telephone>0161-499-1718</Telephone>
   </Member>
   <Member>
      <Name>Smith, Bob</Name>
      <Address></Address>
      <DOB></DOB>
      <Telephone>0161-499-1719</Telephone>
   </Member>
</arrayidentifierlength>
```

Explicit Array Length

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to be Translated

```
3;John;Steve;Paul;Todd;
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
```

```
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="array">
  <annotation>
    <appinfo>
      <nxsd:variables>
        <nxsd:variable name="len" />
      </nxsd:variables>
    </appinfo>
  </annotation>

  <complexType>
    <sequence>
      <element name="TotalMembers" type="string" nxsd:style="terminated"
nxsd:terminatedBy=";">
        <annotation>
          <appinfo>
            <nxsd:variables>
              <nxsd:assign name="len" value="${0}" />
            </nxsd:variables>
          </appinfo>
        </annotation>
      </element>
      <element name="Member" type="string" minOccurs="0" maxOccurs="unbounded"
              nxsd:style="array,terminated" nxsd:arrayLength="${len}"
nxsd:terminatedBy=";" />
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema

```
<array xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <TotalMembers>3</TotalMembers>
   <Member>John</Member>
   <Member>Steve</Member>
   <Member>Paul</Member>
</array>
```

Conditional Processing

This section provides the following examples of conditional processing:

- Processing One Element Within a Choice Model Group Based on the Condition
- Processing Elements Within a Sequence Model Group Based on the Condition

Processing One Element Within a Choice Model Group Based on the Condition

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to be Translated

```
PO28/06/2004^|ABCD Inc.|Oracle
OracleApps025070,000.00
```

```
Database  021230,000.00
ProcessCon021040,000.00
PO01/07/2004^|EFGH Inc.|Oracle
DB2       021230,000.00
Eclipse   021040,000.00
SO29/06/2004|Oracle Apps|5
Navneet Singh
PO28/06/2004^|IJKL Inc.|Oracle
Weblogic  025070,000.00
Tuxedo    021230,000.00
JRockit   021040,000.00
IN30/06/2004;Navneet Singh;Oracle;Oracle Apps;5;70,000.00;350,000.00
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        xmlns:tns="http://www.oracle.com/ias/processconnect"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="container">

  <complexType>
    <choice maxOccurs="unbounded" nxsd:choiceCondition="fixedLength"
 nxsd:length="2">

      <element ref="tns:PurchaseOrder" nxsd:conditionValue="PO" />

      <element ref="tns:SalesOrder" nxsd:conditionValue="SO" />

      <element ref="tns:Invoice" nxsd:conditionValue="IN" />

    </choice>
  </complexType>
</element>

<!-- PO -->
<element name="PurchaseOrder" type="tns:POType"/>

<complexType name="POType">
  <sequence>

    <element name="Date" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="^" />
    <element name="Buyer" type="string" nxsd:style="surrounded"
       nxsd:surroundedBy="|" />
    <element name="Supplier" type="string" nxsd:style="terminated"
 nxsd:terminatedBy="${eol}" />
    <element name="Items">
      <complexType>
        <sequence>
          <element name="Line-Item" minOccurs="3" maxOccurs="3">
            <complexType>
              <group ref="tns:LineItems" />
            </complexType>
          </element>
```

```
        </sequence>
      </complexType>
    </element>
  </sequence>
</complexType>

<group name="LineItems">
  <sequence>
    <element name="Id" type="string" nxsd:style="fixedLength" nxsd:length="10"
        nxsd:padStyle="none"/>
    <element name="Quantity" type="string" nxsd:style="fixedLength"
        nxsd:identifierLength="2" />
    <element name="Price" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
  </sequence>
</group>

<!-- SO -->
<element name="SalesOrder" type="tns:SOType" />

<complexType name="SOType">
  <sequence>
    <element name="Date" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="|" />
    <element name="Item" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="|" />
    <element name="Quantity" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
    <element name="Buyer" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
  </sequence>
</complexType>

<!-- INV -->
<element name="Invoice" type="tns:INVType" />

<complexType name="INVType">
  <sequence>
    <element name="Date" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Purchaser" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Seller" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Item" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Price" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Quantity" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="TotalPrice" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />
  </sequence>
</complexType>

</schema>
```

Translated XML Using the Native Schema

```
<container xmlns="http://www.oracle.com/ias/processconnect">
   <PurchaseOrder>
```

```
   <Date>28/06/2004</Date>
   <Buyer>ABCD Inc.</Buyer>
   <Supplier>Oracle</Supplier>
   <Items>
      <Line-Item>
         <Id>OracleApps</Id>
         <Quantity>50</Quantity>
         <Price>70,000.00</Price>
      </Line-Item>
      <Line-Item>
         <Id>Database  </Id>
         <Quantity>12</Quantity>
         <Price>30,000.00</Price>
      </Line-Item>
      <Line-Item>
         <Id>ProcessCon</Id>
         <Quantity>10</Quantity>
         <Price>40,000.00</Price>
      </Line-Item>
   </Items>
</PurchaseOrder>
<PurchaseOrder>
   <Date>01/07/2004</Date>
   <Buyer>EFGH Inc.</Buyer>
   <Supplier>Oracle</Supplier>
   <Items>
      <Line-Item>
         <Id>DB2       </Id>
         <Quantity>12</Quantity>
         <Price>30,000.00</Price>
      </Line-Item>
      <Line-Item>
         <Id>Eclipse   </Id>
         <Quantity>10</Quantity>
         <Price>40,000.00</Price>
      </Line-Item>
   </Items>
</PurchaseOrder>
<SalesOrder>
   <Date>29/06/2004</Date>
   <Item>Oracle Apps</Item>
   <Quantity>5</Quantity>
   <Buyer>Navneet Singh</Buyer>
</SalesOrder>
<PurchaseOrder>
   <Date>28/06/2004</Date>
   <Buyer>IJKL Inc.</Buyer>
   <Supplier>Oracle</Supplier>
   <Items>
      <Line-Item>
         <Id>Weblogic  </Id>
         <Quantity>50</Quantity>
         <Price>70,000.00</Price>
      </Line-Item>
      <Line-Item>
         <Id>Tuxedo    </Id>
         <Quantity>12</Quantity>
         <Price>30,000.00</Price>
      </Line-Item>
      <Line-Item>
         <Id>JRockit   </Id>
```

```
            <Quantity>10</Quantity>
            <Price>40,000.00</Price>
         </Line-Item>
      </Items>
   </PurchaseOrder>
   <Invoice>
      <Date>30/06/2004</Date>
      <Purchaser>Navneet Singh</Purchaser>
      <Seller>Oracle</Seller>
      <Item>Oracle Apps</Item>
      <Price>5</Price>
      <Quantity>70,000.00</Quantity>
      <TotalPrice>350,000.00</TotalPrice>
   </Invoice>
</container>
```

Processing Elements Within a Sequence Model Group Based on the Condition

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to be Translated

```
PO28/06/2004^|ABCD Inc.|Oracle
OracleApps025070,000.00
Database  021230,000.00
ProcessCon021040,000.00
PO01/07/2004^|EFGH Inc.|Oracle
DB2       021230,000.00
Eclipse   021040,000.00
SO29/06/2004|Oracle Apps|5
Navneet Singh
PO28/06/2004^|IJKL Inc.|Oracle
Weblogic  025070,000.00
Tuxedo    021230,000.00
JRockit   021040,000.00
IN30/06/2004;Navneet Singh;Oracle;Oracle Apps;5;70,000.00;350,000.00
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        xmlns:tns="http://www.oracle.com/ias/processconnect"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="container">

  <complexType>
    <sequence maxOccurs="unbounded">

      <element ref="tns:PurchaseOrder" minOccurs="0" nxsd:startsWith="PO" />

      <element ref="tns:SalesOrder" minOccurs="0" nxsd:startsWith="SO" />

      <element ref="tns:Invoice" minOccurs="0" nxsd:startsWith="IN" />
```

```
    </sequence>
  </complexType>
</element>

<!-- PO -->
<element name="PurchaseOrder" type="tns:POType"/>

<complexType name="POType">
  <sequence>

    <element name="Date" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="^" />    <element name="Buyer" type="string"
nxsd:style="surrounded"
       nxsd:surroundedBy="|" />
    <element name="Supplier" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />
    <element name="Items">
      <complexType>
        <sequence>
          <element name="Line-Item" minOccurs="3" maxOccurs="3">
            <complexType>
              <group ref="tns:LineItems" />
            </complexType>
          </element>
        </sequence>
      </complexType>
    </element>
  </sequence>
</complexType>

<group name="LineItems">
  <sequence>
    <element name="Id" type="string" nxsd:style="fixedLength" nxsd:length="10"
        nxsd:padStyle="none"/>
    <element name="Quantity" type="string" nxsd:style="fixedLength"
        nxsd:identifierLength="2" />
    <element name="Price" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
  </sequence>
</group>

<!-- SO -->
<element name="SalesOrder" type="tns:SOType" />

<complexType name="SOType">
  <sequence>
    <element name="Date" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="|" />
    <element name="Item" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="|" />
    <element name="Quantity" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />
    <element name="Buyer" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />  </sequence>
</complexType>

<!-- INV -->
<element name="Invoice" type="tns:INVType" />

<complexType name="INVType">
  <sequence>
```

```
    <element name="Date" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Purchaser" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Seller" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Item" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />    <element name="Price" type="string"
nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Quantity" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="TotalPrice" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />
  </sequence>
</complexType>

</schema>
```

Translated XML Using the Native Schema

```
<container xmlns="http://www.oracle.com/ias/processconnect">
   <PurchaseOrder>
      <Date>28/06/2004</Date>
      <Buyer>ABCD Inc.</Buyer>
      <Supplier>Oracle</Supplier>
      <Items>
         <Line-Item>
            <Id>OracleApps</Id>
            <Quantity>50</Quantity>
            <Price>70,000.00</Price>
         </Line-Item>
         <Line-Item>
            <Id>Database  </Id>
            <Quantity>12</Quantity>
            <Price>30,000.00</Price>
         </Line-Item>
         <Line-Item>
            <Id>ProcessCon</Id>
            <Quantity>10</Quantity>
            <Price>40,000.00</Price>
         </Line-Item>
      </Items>
   </PurchaseOrder>
   <PurchaseOrder>
      <Date>01/07/2004</Date>
      <Buyer>EFGH Inc.</Buyer>
      <Supplier>Oracle</Supplier>
      <Items>
         <Line-Item>
            <Id>DB2       </Id>
            <Quantity>12</Quantity>
            <Price>30,000.00</Price>
         </Line-Item>
         <Line-Item>
            <Id>Eclipse   </Id>
            <Quantity>10</Quantity>
            <Price>40,000.00</Price>
         </Line-Item>
      </Items>
   </PurchaseOrder>
```

```
   <SalesOrder>
      <Date>29/06/2004</Date>
      <Item>Oracle Apps</Item>
      <Quantity>5</Quantity>
      <Buyer>Navneet Singh</Buyer>
   </SalesOrder>
   <PurchaseOrder>
      <Date>28/06/2004</Date>
      <Buyer>IJKL Inc.</Buyer>
      <Supplier>Oracle</Supplier>
      <Items>
         <Line-Item>
            <Id>Weblogic  </Id>
            <Quantity>50</Quantity>
            <Price>70,000.00</Price>
         </Line-Item>
         <Line-Item>
            <Id>Tuxedo    </Id>
            <Quantity>12</Quantity>
            <Price>30,000.00</Price>
         </Line-Item>
         <Line-Item>
            <Id>JRockit   </Id>
            <Quantity>10</Quantity>
            <Price>40,000.00</Price>
         </Line-Item>
      </Items>
   </PurchaseOrder>
   <Invoice>
      <Date>30/06/2004</Date>
      <Purchaser>Navneet Singh</Purchaser>
      <Seller>Oracle</Seller>
      <Item>Oracle Apps</Item>
      <Price>5</Price>
      <Quantity>70,000.00</Quantity>
      <TotalPrice>350,000.00</TotalPrice>
   </Invoice>
</container>
```

Defining Dates

This examples shows how to define dates.

Native Data to be Translated

```
11/16/0224/11/02
11-20-2002
23*11*2002
01/02/2003 01:02
01/02/2003 03:04:05
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">
```

```
<element name="dateformat">
  <complexType>
    <sequence>
      <element name="StartDate" type="dateTime" nxsd:dateFormat="MM/dd/yy"
         nxsd:style="fixedLength" nxsd:length="8" />
      <element name="EndDate" type="dateTime"   nxsd:dateFormat="dd/MM/yy"
         nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
      <element name="Milestone" type="dateTime" nxsd:dateFormat="MM-dd-yyyy"
         nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
      <element name="DueDate" type="dateTime"   nxsd:dateFormat="dd*MM*yyyy"
         nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
      <element name="Date" type="dateTime"   nxsd:dateFormat="MM/dd/yyyy hh:mm"
         nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
      <element name="Date" type="dateTime"   nxsd:dateFormat="MM/dd/yyyy hh:mm:ss"
         nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema

```
<dateformat xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <StartDate>2002-11-16T00:00:00</StartDate>
   <EndDate>2002-11-24T00:00:00</EndDate>
   <Milestone>2002-11-20T00:00:00</Milestone>
   <DueDate>2002-11-23T00:00:00</DueDate>
   <Date>2003-01-02T01:02:00</Date>
   <Date>2003-01-02T03:04:05</Date>
</dateformat>
```

Note:

`nxsd:dateParsingMode="lax/strict"` and locale support have been added to the existing date format.

Defining Dates: With Locale Support

The following example depicts the use of `nxsd:dateParsingMode="lax/strict"` and locale support.

Native Data Format to be Translated

```
11/16/0224/11/02
11-20-2002
23*11*2002
01/02/2003 01:02
01/02/2003 03:04:05
Thu, 26 May 2005 15:50:11 India Standard Time
Do, 26 Mai 2005 15:43:10 Indische Normalzeit
20063202
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
```

```
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="dateformat">
  <complexType>
    <sequence>
      <element name="StartDate" type="date" nxsd:dateFormat="MM/dd/yy"
nxsd:localeLanguage="en" nxsd:style="fixedLength" nxsd:length="8" />
      <element name="EndDate" type="date"   nxsd:dateFormat="dd/MM/yy"
nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
      <element name="Milestone" type="dateTime" nxsd:dateFormat="MM-dd-yyyy"
nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
      <element name="DueDate" type="dateTime"   nxsd:dateFormat="dd*MM*yyyy"
nxsd:style="terminated" nxsd:terminatedBy="${eol}" />

      <element name="Date" type="dateTime"   nxsd:dateFormat="MM/dd/yyyy hh:mm"
nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
      <element name="Date" type="dateTime"   nxsd:dateFormat="MM/dd/yyyy hh:mm:ss"
nxsd:style="terminated" nxsd:terminatedBy="${eol}" />

      <element name="LongDateInEnglish" type="dateTime"   nxsd:dateFormat="EEE, d
MMM yyyy HH:mm:ss zzzz" nxsd:localeLanguage="en" nxsd:localeCountry="US"
nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
      <element name="LongDateInGerman" type="dateTime"   nxsd:dateFormat="EEE, d
MMM yyyy HH:mm:ss zzzz" nxsd:localeLanguage="de" nxsd:style="terminated"
nxsd:terminatedBy="${eol}" />

      <element name="InvalidDate" type="dateTime"   nxsd:dateParsingMode="lax"
nxsd:dateFormat="yyyyMMdd" nxsd:style="terminated" nxsd:terminatedBy="${eol}" />

    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML

```
<dateformat xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <StartDate>2002-11-16</StartDate>
   <EndDate>2002-11-24</EndDate>
   <Milestone>2002-11-20T00:00:00</Milestone>
   <DueDate>2002-11-23T00:00:00</DueDate>
   <Date>2003-01-02T01:02:00</Date>
   <Date>2003-01-02T03:04:05</Date>
   <LongDateInEnglish>2005-05-26T15:50:11</LongDateInEnglish>
   <LongDateInGerman>2005-05-26T15:43:10</LongDateInGerman>
   <InvalidDate>2008-08-02T00:00:00</InvalidDate>
</dateformat>
```

Using Variables

This example shows how to use variables.

Native Data Format to Be Translated

```
{,;}Fred,"2 Old Street, Old Town,Manchester","20-08-1954";"0161-499-1718"
phone-2
phone-3
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">
 <element name="variable">
  <annotation>
    <documentation>
      1. var1 - variable declaration
      2. var2 - variable declaration with default value
      3. EOL - variable declaration with referencing a system variable
    </documentation>
    <appinfo>
      <junkies/>
      <nxsd:variables>
        <nxsd:variable name="var1" />
        <nxsd:variable name="var2" value="," />
        <nxsd:variable name="SystemEOL" value="${system.line.separator}" />
      </nxsd:variables>
      <junkies/>
      <junkies/>
      <junkies/>
    </appinfo>
  </annotation>

  <complexType>
    <sequence>
      <element name="delims" type="string" nxsd:style="surrounded"
          nxsd:leftSurroundedBy="{" nxsd:rightSurroundedBy="}" >
         <annotation>
           <appinfo>
             <junkies/>
             <junkies/>
             <junkies/>
             <nxsd:variables>
              <nxsd:assign name="var1" value="${0,1}"/>
              <nxsd:assign name="var2" value="${1}" />
             </nxsd:variables>
           </appinfo>
         </annotation>
      </element>

      <element name="PersonName" type="string" nxsd:style="terminated"
          nxsd:terminatedBy="${var1}" nxsd:quotedBy="&quot;" />
      <element name="Address" type="string" nxsd:style="terminated"
          nxsd:terminatedBy="${var1}" nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
          nxsd:terminatedBy="${var2}" nxsd:quotedBy='"'/>
      <element name="Telephone1" type="string" nxsd:style="terminated"
          nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
      <element name="Telephone2" type="string" nxsd:style="terminated"
          nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
```

```
      <element name="Telephone3" type="string" nxsd:style="terminated"
          nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema

```
<variable xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <delims>,;</delims>
   <PersonName>Fred</PersonName>
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
   <Telephone1>0161-499-1718</Telephone1>
   <Telephone2>phone-2</Telephone2>
   <Telephone3>phone-3</Telephone3>
</variable>
```

Defining Prefixes and Suffixes

In native format, when data is read, the specified data is prefixed, suffixed, or both, as shown in the following example.

Native Data to Be Translated

```
     Fred,  "2 Old Street, Old Town,Manchester","20-08-1954",0161-499-1718
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD"
        >

<element name="terminated">
  <complexType>
    <sequence>
      <element name="PersonName" type="string" nxsd:prefixWith="Mr."
       nxsd:style="terminated" nxsd:terminatedBy="," nxsd:quotedBy="&quot;" />
      <element name="Address" type="string" nxsd:suffixWith="]]"
       nxsd:prefixWith="[[" nxsd:style="terminated" nxsd:terminatedBy=","
       nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema

```
<terminated xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Mr.Fred</PersonName>
   <Address>[[2 Old Street, Old Town,Manchester]]</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>0161-499-1718</Telephone>
</terminated>
```

Defining Skipping Data

Translator skips, before or after the data is read, depending on the `skipMode` construct, as shown in the following example:

Native Data to Be Translated

```
     Fred,   "2 Old Street, Old Town,Manchester","20-08-1954",0161-499-1718
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD"
        >

<element name="terminated">
  <complexType>
    <sequence>
      <element name="PersonName" type="string" nxsd:skip="5"
       nxsd:style="terminated" nxsd:terminatedBy="," nxsd:quotedBy="&quot;" />
      <element name="Address" type="string" nxsd:skipMode="before" nxsd:skip="3"
       nxsd:style="terminated" nxsd:terminatedBy="," nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:skipMode="after" nxsd:skip="6"
       nxsd:style="terminated" nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using Native Schema

```
<terminated xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Fred</PersonName>
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>99-1718</Telephone>
</terminated>
```

Defining fixed and default Values

When an element is declared without `nxsd` annotations but the value specified is either `fixed` or `default`, the translator uses the value provided and does not throw any exceptions.

Native Data to Be Translated

```
Fred,"2 Old Street, Old Town,Manchester","20-08-1954","0161-499-1718"
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="terminated">
  <annotation>
    <appinfo>
      <nxsd:variables>
        <nxsd:variable name="x" value="hello" />
      </nxsd:variables>
      <junkies/>
      <junkies/>
      <junkies/>
    </appinfo>
  </annotation>

  <complexType>
    <sequence>
      <element name="PersonName" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy="&quot;" />
      <element name="Age" type="string" fixed="16"  />
      <element name="Address" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="salutation" type="string" default="${x}"  />
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using Native Schema

```
<terminated xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Fred</PersonName>
   <Age>16</Age>
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
   <salutation>hello</salutation>
   <Telephone>0161-499-1718</Telephone>
</terminated>
```

Defining write

The `write` construct writes the literal at the current position in the output stream, either before writing the actual data or after writing it.

Input XML

```
<terminated xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Fred</PersonName>
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>0161-499-1718</Telephone>
</terminated>
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD"
        >

<element name="terminated">
  <complexType>
    <sequence>
      <element name="PersonName" type="string" nxsd:writeMode="before"
       nxsd:write="Mr." nxsd:style="terminated" nxsd:terminatedBy=","
       nxsd:quotedBy="&quot;" />
      <element name="Address" type="string" nxsd:writeMode="after"
       nxsd:write="Over." nxsd:style="terminated" nxsd:terminatedBy=","
       nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated Data Using Native Schema

```
Mr.Fred,"2 Old Street, Old Town,Manchester",Over.20-08-1954,0161-499-1718
```

Defining LookAhead

The `LookAhead` construct is of the following types:

- Type 1: `LookAhead` X chars, read the value from a position using a style, and match against the specified literal.
- Type 2: `LookAhead` X chars, read the value from a position using a style, and store that value in a variable to be used later.

LookAhead: Type 1

`LookAhead` X chars, read the value from a position using a style, and match against the specified literal.

Native Data Format to Be Translated

```
Fred,"2 Old Street, Old Town,Manchester","20-08-1954","0161-499-1718",YES
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="LookAhead">
  <complexType>
    <sequence minOccurs="0" nxsd:lookAhead="70" nxsd:lookFor="YES">
      <element name="PersonName" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy="&quot;" />
      <element name="Address" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using Native Schema

```
<LookAhead xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Fred</PersonName>
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>0161-499-1718</Telephone>
</LookAhead>
```

LookAhead: Type 2

In native schema, `LookAhead` X chars, read the value from a position using a style, and store that value in a variable to be used later.

Native Data Format to Be Translated

```
Name1,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", YES
Name2,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", NO
Name3,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", NO
Name4,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", YES
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">
```

```
<!--
nxsd:lookAhead="70" nxsd:scan="3"
-->

<element name="LookAhead">
  <complexType>
    <choice maxOccurs="unbounded" nxsd:choiceCondition="${x}" nxsd:lookAhead="70"
       nxsd:scanLength="3" nxsd:assignTo="${x}">
      <element name="Record1" type="string" nxsd:conditionValue="YES"
       nxsd:style="terminated" nxsd:terminatedBy="," nxsd:skipMode="after"
       nxsd:skipUntil="${eol}" />
      <element name="Record2" type="string" nxsd:conditionValue="NO "
       nxsd:style="terminated" nxsd:terminatedBy="," nxsd:skipMode="after"
       nxsd:skipUntil="${eol}" />
    </choice>
  </complexType>
</element>

</schema>
```

Translated XML Using Native Schema

```
<LookAhead xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <Record1>Name1</Record1>
   <Record2>Name2</Record2>
   <Record2>Name3</Record2>
   <Record1>Name4</Record1>
</LookAhead>
```

Defining Complex Look Ahead Strategies for Conditional Processing of Record Using Regular Expressions

The NXSD wizard provides look ahead attributes that enable you to specify Regular Expression patterns for filtering both fixed length records and variable length records.

You can use conditional processing support, using look ahead attributes with Fixed Length, Delimited and Complex file types.

Using this conditional processing, you can obtain results where only the records that satisfy the regular expression are published in XML.

You can also specify the option to raise an exception when a record does not satisfies regular expression. This assists in conditional processing by providing a choice condition.

For Regular Expression support you can specify the Character Sequence against which the Regular Expression can be matched. Note that the Character Sequence should not go beyond the record boundary.

For Fixed Length records, the length of the record and the record's various fields are known in advance. This helps you specify the starting position and the length of the stream against which the Regular Expression is to be matched.

See Table 6-6 for a list of fixed length record nxsd constructs and their definition.

The construct is provided in its nxsd: format, but has its equivalent in the Native Format Builder user interface and the construct is referenced in the discussion of the appropriate screen in the Native Format Builder Wizard.

Table 6-6 Fixed Length Record LookAhead Constructs

Construct	Description

nxsd:lookAhead	Looks for a match ahead of the current position in the input stream. If a match is found, the record for which you have specified this construct is processed; otherwise, it is skipped. Use this feature as follows:
	`nxsd:lookAhead="20"`
	`nxsd:lookForPattern=".*Pattern.*"`
	`nxsd:scanLength="30"`
	This indicates to skip 20 characters and to look for the `java.util.regex.Pattern` ".*Pattern.*" in the next 30 characters. If pattern is found, the record is processed; otherwise, it is skipped.
	This construct can be user with both `lookFor` and `lookForPattern`.

`nxsd:lookForPattern`	Specify the regular expression, the `java.util.regex.Pattern` , which is matched against the input stream.
`nxsd:scanLength`	Specify the length of the data in the input stream after lookAhead. This data is matched against the `lookForPattern`.
`nxsd:reportLookAhead Error`	Specify the Boolean value, whether an error is raised for not matching the data with lookForPattern. Default value is 'false' which means do not raise an error for not matching data with lookForPattern and skip the unmatched record. This might result in unmatched records not being processed and might appear to a user as loss of data.

Refer to the following use cases.

- You do not specify `nxsd:scanLength` is, the `uniqueMessageSeparator` is used to determine the Character Sequence.

  `uniqueMessageSeparator` is a String specifying the unique message separator in the native data, in a batch of messages. If the `uniqueMessageSeparator` is not specified then the default ${eol} is used.

  In the following scenario, since `nxsd:scanLength` is not specified, the Translator uses `uniqueMessageSeparator` to determine the character sequence, which is matched against the regular expression.

  – Scan all the records where the address is New Town. The Native Data is:

  ```
  #Fred,"2 Old Street, Old Town, Manchester","17-08-1954","0161-498-1718", YES
  #Tred,"2 Old Street, Old Town, Manchester","16-08-1974","0161-486-1718", YES
  John,"2 Old Street, New Town, Manchester","20-08-2004","0161-497-1718", NO
  #
  ```

  The NXSD for this is the following:, using uniqueMessageSeparator to help determine the character sequence, is:

  ```
  <schema …
   nxsd:uniqueMessageSeparator="#">
  ```

```
<sequence minOccurs="0" maxOccurs="unbounded" nxsd:lookAhead="0"
nxsd:lookForPattern="*New Town*">
```

– Scan all the records where address is in New Town, and where `nxsd:uniqueMessageSeparator` is not specified. The Translator uses `${eol}` to determine the Character Sequence, which is matched against the regular expression.

```
Fred,"2 Old Street, Old Town, Manchester","17-08-1954","0161-498-1718", YES
Tred,"2 Old Street, Old Town, Manchester","16-08-1974","0161-486-1718", YES
John,"2 Old Street, New Town, Manchester","20-08-2004","0161-497-1718", NAH
```

The NXSD for this is:

```
<sequence minOccurs="0" maxOccurs="unbounded" nxsd:lookAhead="0"
nxsd:lookForPattern="*New Town*">
```

- The following use cases do not use `nxsd:uniqueMessageSeparator`:

  In a second use case, you specify `nxsd:scanLength` to determine the Character Sequence.

```
Fred,"2 Old Street, New Town, Manchester","17-08-1954","0161-498-1718", YES
Tred,"2 Old Street, Old Town, Manchester","16-08-1974","0161-486-1718", YES
John,"2 Old Street, New Town, Manchester","20-08-2004","0161-497-1718", NAH
```

The NXSD to use against this data is:

```
<sequence minOccurs="0" maxOccurs="3" nxsd:lookAhead="0"
nxsd:lookForPattern=".*New Town.*" nxsd:scanLength="73">
```

- – Scan all records where the second field is made up of digits only. The Native Data is:

```
Fred, add, 20-08-1954, 0161-498-1718", YES
Tred, 100, 20-08-1954, 0161-486-1718", YES
John, add, 20-08-1954, 0161-497-1718", NAH
```

The NXSD to determine if the second field is made up of digits only is:

```
<sequence minOccurs="0" maxOccurs="3" nxsd:lookAhead="0"
nxsd:lookForPattern=".*,\d+,.*" nxsd:scanLength="38">
```

– Scan all records where zip contains 5 digits only. The Native Data is:

```
Fred, zipID, 20-08-1954, 0161-498-1718", YES
Tred, 90210, 20-08-1954, 0161-486-1718", YES
John, 9021Z, 20-08-1954, 0161-497-1718", NAH
```

The NXSD to look for the appropriate pattern and make the 5-digit determination is:

```
<sequence minOccurs="0" maxOccurs="3" nxsd:    lookAhead="0"
nxsd:lookForPattern=".*,\d{5},.*" nxsd:scanLength="40">
```

– Scan all records where the name does not include any digit and the second field contains a five-digit zip code. The Native Data is:

```
JohnBrown, zipID, 20-08-1954, 0161-498-1718", YES
Sweety101, 90210, 20-08-1954, 0161-486-1718", YES
CrowyBowy, 90210, 20-08-1954, 0161-497-1718", NAH
```

The NXSD for this, again using the regular expressions, is:

```
<sequence minOccurs="0" maxOccurs="4" nxsd:lookAhead="0"
nxsd:lookForPattern="[a-zA-Z]*,\d{5},.*" nxsd:scanLength="45">
```

Using scan length works well with Fixed records, but the limitation with a variable length record is that the length of this type of record in not known in advance. Hence, the `nxsd:scanLength` attribute cannot be used in such scenarios. The Translator would require a way to determine the character sequence which would be matched against regular expression.

To address this limitation with variable length records, rather than using nxsd:scanLength, you can use two attributes when doing a complex lookahead, `nxsd:LookFrom` and `nxsd:lookTill`.

The character sequence, where a pattern can be matched, can be derived through `nxsd:lookFrom` and `nxsd:lookTill` attributes. The basic NXSD for a lookahead scan for a variable length record is:

```
nxsd:lookFrom="FirstInstanceOfString" nxsd:lookTill="firstInstanceOfString after
lookFrom"
nxsd:lookForPattern="java.util.regex.Pattern" [nxsd:skipUntill="${eol} or Literal"]
```

See Table 6-7 for a list of Variable Length Record lookAhead Constructs.

The construct is provided in its nxsd: format, but has its equivalent in the Native Format Builder user interface and the construct is referenced in the discussion of the appropriate screen.

Table 6-7 Variable Length Record lookahead Constructs

Construct	Definition
`nxsd:lookFrom`	Specify the Literal, data is matched only after `lookFrom` Literal is found in input stream
`nxsd:lookForPattern`	Specify the `java.util.regex.Pattern` , (regular expression) which is matched against the input stream.
`nxsd:lookTill`	Specify the Literal up to which data in input stream is matched. Data in input stream between `nxsd:lookFrom` (excluded) and `lookTill` (excluded) is matched against `lookForPattern`.
`nxsd:skipUntil`	Specify Literal up to which data in the input stream is skipped, when the data does not match against lookForPattern.
`nxsd:reportLookAhead Error`	Specify the Boolean value if an error is raised for not matching the data with `lookForPattern`. The default value is 'false' which means do not raise an error for not matching data with `lookForPattern` and also skip the unmatched record.

Note that when you create the NXSD for translating Native files, you can specify `lookAhead` in the schema to extract values based on the regular expression specified using `lookForPattern`.

However, when extracting variable length records, because you do not know the length of record, you can use `lookFrom` and `lookTill` attributes in NXSD element, thus, using JDeveloper Native Format Builder User Interface window, you specify values to `lookFrom` and `lookTill` with or without `skipUntil`.

There are a few use cases relevant to an understanding of these constructs when using complex look ahead with variable length records.

- For the first use case, scan all records with address in New Town where `nxsd:lookTill` is the end of the record. In this case, the Translator must know what position to skip to if schema element cardinality is not exceeded and `nxsd:lookForPattern` does not match.

  ```
  Tred,"2 Old Street, NEW Town,Manchester",@NAH
  Bred,"2 Old Street, Old Town,Manchester",@NAH
  dred,"2 Old Street, New Town,Manchester",@YES
  Hred,"2 Old Street, Old Town,Manchester",@YES
  Bred,"2 Old Street, Old Town,Manchester",@YES
  ```

  The NXSD for this use case is the following, indicating to look for New Town, with the end of record indicated by nxsd:lookTill="${eol}

  ```
  <sequence minOccurs="0" maxOccurs="3" nxsd:lookFrom="Street" nxsd:lookTill="$
  {eol}" nxsd:lookForPattern=".*New Town.*">
  ```

- In another use case, `nxsd:lookTill` is not the end of the record. In this case, the Translator does not know till what position to skip when a record does not match.

  You can indicate the position to skip to when a record does not match information by using the attribute `nxsd:skipUntil`.

  You must explicitly define the `nxsd:skipUntil` clause in conjunction with the `nxsd:lookTill` clause.

  When using `nxsd:skipUntil`, always specify the end of record, that is:

  `nxsd:skipUntil=" ${eol} or Literal"`

  – In this use case, scan all records with address in New Town. The Native Data is:

    ```
    Tred,"2 Old Street, NEW Town, Manchester", @NAH
    Bred,"2 Old Street, Old Town, Manchester", @NAH
    Dred, "2 Old Street, New Town, Manchester", @YES
    Hred,"2 Old Street, Old Town, Manchester", @YES
    Bred,"2 Old Street, Old Town, Manchester", @YES
    ```

    The NXSD for this is indicates to skip until end of file if a record is not matched, but only to look until "Manchester":

    ```
    <sequence minOccurs="0" maxOccurs="3" nxsd:lookFrom="Street" nxsd:lookTill="
    Manchester" nxsd:lookForPattern=".*New Town.*" nxsd:skipUntil="${eol}">
    ```

  – You could also match the regular expression from some predefined position to the occurrence of some string. To do this, specify the starting position using `nxsd:lookAhead` along with the `nxsd:lookTill` attribute.

    This is useful in scenarios where you do not have knowledge of the `nxsd:lookFrom` attribute but where you do know the end of record.

    In such scenarios, you can specify the `nxsd:lookAhead="0"` and `nxsd:lookTill="end of record"`. In this case, you must ensure that the `nxsd:lookTill` is specified till end of record.

    In the following example, you want to scan all records with address in New Town. The Native Data is:

    ```
    Tred,"2 Old Street, NEW Town, Manchester", @NAHBred,"2 Old Street, Old Town,
    Manchester", @NAHDred,"2 Old Street, New Town, Manchester", @YESHred,"2 Old
    Street, Old Town, Manchester", @YESBred,"2 Old Street, Old Town, Manchester",
    @YES
    ```

The NXSD for this use case, where you know the end of record, is:

```
<sequence minOccurs="0" maxOccurs="3" nxsd:lookAhead="0"
nxsd:lookForPattern=".*New Town.*" nxsd:lookTill="${eol}">
```

– In another case `nxsd:lookTill` is not the end of record. You must specify the `nxsd:skipUntil`. This indicates the point at which the Translator has to skip the record where the record does not match against the given Regular Expression.

  In the following example, scan all records with address in New Town.

–
```
Tred,"2 Old Street, NEW Town, Manchester", @NAH
Bred,"2 Old Street, Old Town, Manchester", @NAH
Dred,"2 Old Street, New Town, Manchester", @YES
Hred,"2 Old Street, Old Town, Manchester", @YES
Bred,"2 Old Street, Old Town, Manchester", @YES
```

  The NXSD for this, when `nxsd:skipUntil` defines the end of record is:

```
<sequence minOccurs="0" maxOccurs="3" nxsd:lookAhead="0"
nxsd:lookForPattern=".*New Town.*" nxsd:lookTill="Manchester" nxsd:skipUntil="$
{eol}">
```

– Using the JDeveloper Native Format Builder User Interface window, you can specify values to `lookAhead` and `lookTill` with or without using skipUntil, but when you select them in the JDeveloper UI and proceed next to see the NXSD source, the schema misses the `lookTill` attribute. To avoid this issue, explicitly write `<namespace>:lookTill="literal"` with the lookAhead attribute.

Including the Newline Character when Looking for a Pattern

For every row in a record that has a newline character, the newline character should be included as char while defining the Complex Look Ahead patterns for finding the last word in the statement.

For example, if you are looking for the following record

```
canvas,"1 17th Cross, North
London,England","17-08-1954","0161-498-1718",YES
```

You might think you could use the following as the pattern for which to look.

```
nxsd:lookAhead="73" nxsd:scanLength="3"
nxsd:lookForPattern=".ES"
```

However, if the record searched for is actually the following, with the newline character:

```
canvas,"1 17th Cross, North
London,England","17-08-1954","0161-498-1718",YES\n
```

Thus, the `lookForPattern` clause to find this record should be the following:

```
nxsd:lookForPattern=".ES\n"
```

or, to accommodate for the delimiter,

```
nxsd:lookForPattern=".ES."
```

Hence the attributes are

```
nxsd:lookAhead="73" nxsd:scanLength="4"
nxsd:lookForPattern=".ES\n" or nxsd:lookForPattern=".ES."
```

Defining outboundHeader

The actual content of `outboundHeader` can use variables, specifically `${eol}`. When `headerLines` and `outboundHeader` both are available, `outboundHeader` takes precedence in the outbound.

Note:

In the inbound direction, the Skipping Headers feature is supported. Only predefined variables can be used in a header because other variables might either not be accessible or would have only literals.

Input XML

```
<terminated xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Fred</PersonName>
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>0161-499-1718</Telephone>
</terminated>
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD"
        nxsd:hasHeader="true"
        nxsd:outboundHeader="This is a header ${eol}">

<element name="terminated">
  <complexType>
    <sequence>
      <element name="PersonName" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy="&quot;" />
      <element name="Address" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated Data

```
This is a header
Fred,"2 Old Street, Old Town,Manchester",20-08-1954,0161-499-1718
```

Defining Complex Condition in conditionValue

When you use the `conditionValue` construct along with the `choiceCondition` construct, you can specify match criteria such as equals (==) and not equals (!=), along with the Boolean operators AND and OR, for comparison between the value read and the value specified in the `conditionValue` construct.

Native Data Format to Be Translated

```
Order, ID41678, 20May2000
Item1, GigaWidget, 60, $75
Item2, MegaBucket, 48, $125
Cust1, Hopkins Associates, ID26490
Order, ID41680, 20May2000
Item3, Rt.Clopper, 40, $100
Item4, Lt.Clopper, 50, $100
Cust2, Jersey WebInovaters, ID46786
```

Native Schema

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/extensions/SampleNS"
            xmlns:tns="http://xmlns.oracle.com/pcbpel/nxsd/extensions/SampleNS"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified" nxsd:encoding="US-ASCII"
nxsd:stream="chars" nxsd:version="NXSD">

<xsd:element name="Container">
    <xsd:complexType>
      <xsd:choice minOccurs="1" maxOccurs="unbounded"
       nxsd:choiceCondition="terminated" nxsd:terminatedBy=",">
        <xsd:element name="Customer" nxsd:conditionValue="(== Cust1) or (== Cust2)
         and (!= emp)">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="C1" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="C2" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="${eol}" nxsd:quotedBy="&quot;">
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Item" nxsd:conditionValue="(== Item1) or (== Item2) or
         (==Item3) or (== Item4)">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="C1" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="C2" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="C3" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="${eol}" nxsd:quotedBy="&quot;">
              </xsd:element>
            </xsd:sequence>
```

```
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Order" nxsd:conditionValue="Order">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="C1" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="C2" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="${eol}" nxsd:quotedBy="&quot;">
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Translated XML Using Native Schema

```
<Container xmlns="http://xmlns.oracle.com/pcbpel/nxsd/extensions/SampleNS">
   <Order>
      <C1> ID41678</C1>
      <C2> 20May2000</C2>
   </Order>
   <Item>
      <C1> GigaWidget</C1>
      <C2> 60</C2>
      <C3> $75</C3>
   </Item>
   <Item>
      <C1> MegaBucket</C1>
      <C2> 48</C2>
      <C3> $125</C3>
   </Item>
   <Customer>
      <C1> Hopkins Associates</C1>
      <C2> ID26490</C2>
   </Customer>
   <Order>
      <C1> ID41680</C1>
      <C2> 20May2000</C2>
   </Order>
   <Item>
      <C1> Rt.Clopper</C1>
      <C2> 40</C2>
      <C3> $100</C3>
   </Item>
   <Item>
      <C1> Lt.Clopper</C1>
      <C2> 50</C2>
      <C3> $100</C3>
   </Item>
   <Customer>
      <C1> Jersey WebInovaters</C1>
      <C2> ID46786</C2>
   </Customer>
</Container>
```

Defining Complex Condition in choiceCondition

The `choiceCondition` construct is used along with the conditionValue construct for records that are complex and may have fields delimited by multiple delimiter types. The other choiceCondition types available are `FixedLength`, `Variable`, and `Ad hoc`. The following example is for the variable choiceCondition type.

Native Data Format to Be Translated

```
Name1,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", YES
Name2,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", NO
```

Native Schema

```
<element name="LookAhead">
  <complexType>
    <choice maxOccurs="unbounded" nxsd:choiceCondition="${x}" nxsd:lookAhead="70"
nxsd:scanLength="3" nxsd:assignTo="${x}">
      <element name="Record1" type="string" nxsd:conditionValue="YES"
nxsd:style="terminated" nxsd:terminatedBy="," nxsd:skipMode="after" nxsd:skipUntil="$
{eol}" />
      <element name="Record2" type="string" nxsd:conditionValue="NO "
nxsd:style="terminated" nxsd:terminatedBy="," nxsd:skipMode="after" nxsd:skipUntil="$
{eol}" />
    </choice>
  </complexType>
</element>
```

Translated XML Using Native Schema

```
<LookAhead xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <Record1>Name1</Record1>
   <Record2>Name2</Record2>
</LookAhead>
```

Defining dataLines

If the requirement is to translate only a portion of the data and not the entire data, then you can specify the number of lines to be ignored from the beginning of the file and the number of lines to be translated from that point onwards by using the `dataLines` construct.

Native Data Format to Be Translated

```
Fred,addr,20-08-1954,0161-499-1718
Tam,addr,20-08-1954,0161-499-1718
Albert,addr,20-08-1954,0161-499-1718
Bill,addr,20-08-1954,0161-499-1718
Phil,addr,20-08-1954,0161-499-1718
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
```

```
        nxsd:version="NXSD"

        nxsd:headerLines="1"
        nxsd:dataLines="1">

<element name="terminated">
  <complexType>
    <sequence maxOccurs="unbounded">
      <element name="PersonName" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," />
      <element name="Address" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," />
      <element name="DOB" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," />
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using Native Schema

```
<terminated xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Tam</PersonName>
   <Address>addr</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>0161-499-1718</Telephone>
</terminated>
```

Defining Date Formats with Time Zone

In the translator, the date or time must be associated with a time zone. The translator supports the date formats with time zone for both, the date in native data and for the date in XML.

There are two parts when translating a date/time string. First, the format of the date in the native data (`dateformat`), second is the time zone to use when parsing that date (`timeZone` or `useTimeZone`). The translator uses these details while parsing the date/time string.

After the parsing, by default, the date string is converted to the ISO-8601 format in an XML. You can override the defaults by using `XMLDateFormat` and `XMLTimeZone`, or `useTimeZone`.

Native Data Format to Be Translated

```
11/16/0224/11/02
11-20-2002
23*11*2002
01/02/2003 01:02
01/02/2003 03:04:05
Thu, 26 May 2005 15:50:11 India Standard Time
Do, 26 Mai 2005 15:43:10 Indische Normalzeit
20063202
11/16/02
```

Native Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="dateformat">
  <complexType>
    <sequence>
      <element name="StartDate" type="date" nxsd:dateFormat="MM/dd/yy"
       nxsd:localeLanguage="en" nxsd:style="fixedLength" nxsd:length="8" />

      <element name="EndDate" type="date"   nxsd:dateFormat="dd/MM/yy"
       nxsd:style="terminated" nxsd:terminatedBy="${eol}" />

      <element name="Milestone" type="dateTime" nxsd:useTimeZone="UTC"
       nxsd:dateFormat="MM-dd-yyyy" nxsd:style="terminated"
nxsd:terminatedBy="${eol}" />

      <element name="DueDate" type="dateTime" nxsd:useTimeZone="UTC"
       nxsd:dateFormat="dd*MM*yyyy" nxsd:style="terminated" nxsd:terminatedBy="$
{eol}" />

      <element name="Date" type="dateTime" nxsd:useTimeZone="UTC"
       nxsd:dateFormat="MM/dd/yyyy hh:mm" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />

      <element name="Date" type="dateTime" nxsd:useTimeZone="UTC"
       nxsd:dateFormat="MM/dd/yyyy hh:mm:ss" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />

      <element name="LongDateInEnglish" type="dateTime"
       nxsd:displayTimeZone="true" nxsd:useTimeZone="IST" nxsd:dateFormat="EEE, d MMM
       yyyy HH:mm:ss zzzz" nxsd:localeLanguage="en" nxsd:localeCountry="US"
       nxsd:style="terminated" nxsd:terminatedBy="${eol}" />

      <element name="LongDateInGerman"  type="dateTime"
       nxsd:displayTimeZone="true" nxsd:useTimeZone="IST" nxsd:dateFormat="EEE, d MMM
       yyyy HH:mm:ss zzzz" nxsd:localeLanguage="de" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />

      <element name="InvalidDate" type="dateTime" nxsd:useTimeZone="UTC"
       nxsd:dateParsingMode="lax" nxsd:dateFormat="yyyyMMdd" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />

      <element name="MyFormatDate" type="string" nxsd:dateFormat="MM/dd/yy"
       nxsd:xmlDateFormat="dd-MM-yyyy" nxsd:localeLanguage="en"
nxsd:style="fixedLength"
       nxsd:length="8" />

    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using Native Schema

```
<dateformat xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <StartDate>2002-11-16</StartDate>
   <EndDate>2002-11-24</EndDate>
   <Milestone>2002-11-20T00:00:00</Milestone>
   <DueDate>2002-11-23T00:00:00</DueDate>
   <Date>2003-01-02T01:02:00</Date>
   <Date>2003-01-02T03:04:05</Date>
   <LongDateInEnglish>2005-05-26T15:50:11+05:30</LongDateInEnglish>
   <LongDateInGerman>2005-05-26T15:43:10+05:30</LongDateInGerman>
   <InvalidDate>2008-08-02T00:00:00</InvalidDate>
   <MyFormatDate>16-11-2002</ MyFormatDate >
</dateformat>
```

Implementing Validation During Translation

You must configure Oracle JCA Adapters to implement validation during translation. Validation helps ensure that Oracle JCA Adapters do not publish invalid messages during translation.

You can implement either one or both of the following types of validation:

- Payload Validation
- Schema Validation

Payload Validation

Payload validation involves validating the input and output XML messages that are processed by Oracle JCA Adapters. You can set payload validation at one of the these levels:

- Top-Level Validation
- Field-Level Validation

  You can enable validation either at the NXSD layer (by using the nxsd:validate top-level flag) or at the Service Engine layer. See " Configuring BPEL Process Service Components and Engines" in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite* for more information on Service Engine layer validation.

Top-Level Validation

In top-level validation, the DOMResult (result in the form of a Document Object Model) is validated against the XML schema. This form of validation is implemented on both inbound and outbound payloads. This form of validation can control the publishing of invalid records and provide information about XML validation errors. However, it does not provide translation context. For example, information about the line and column in the native stream where the error was encountered is not provided by top-level validation.

To implement top-level validation of XML messages:

- The nxsd namespace in the message must be set to the following:

  ```
  xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
  ```

- The validation flag must be set to `true` as follows:

  ```
  nxsd:validation="true"
  ```

For example:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
     xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
         attributeFormDefault="unqualified"
        nxsd:stream="chars"
         nxsd:version="NXSD"
        nxsd:validation="true"
        >
```

Field Level Validation

In field-level validation, the individual fields are validated against the XML schema. This form of validation is implemented only on inbound payloads, not on outbound payloads.

If the XML message does not conform to the XML schema, then information about the exact line and character where the error was encountered is displayed.

To implement field-level validation of XML messages:

- The nxsd namespace in the message must be set to the following:

  ```
  xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
  ```

- The validation flag must be set to `true` as follows:

  ```
  nxsd:fieldValidation="true"
  ```

For example:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
elementFormDefault="qualified"
attributeFormDefault="unqualified"
 nxsd:stream="chars"
  nxsd:version="NXSD"
   nxsd:fieldValidation="true"
   >
```

Schema Validation

Schema validation involves validating the schema (native schemas or XML schemas) that you define for the native or XML data formats to be translated by the Oracle JCA Adapters.

To enable schema validation:

- The nxsd namespace in the message must be set to the following:

  ```
  xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
  ```

- The validate nxsd flag must be set to `true` as follows:

  ```
  nxsd:validateNxsd="true"
  ```

For example:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
     xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
         attributeFormDefault="unqualified"
        nxsd:stream="chars"
```

```
 nxsd:version="NXSD"
nxsd: validateNxsd ="true"
>
```

Note:

The `nxsd:validateNxsd="true"` validation flag *does not* affect payload level validations.

Processing Files with BOM

The byte order mark (BOM) is a special U+FEFF Unicode character that describes the encoding of a byte sequence. The Native Format Translator can be configured to use BOM for determining the character encoding of the native input data. By default, BOM is not used. If your input data uses BOM, then set the `nxsd:parseBom` attribute to `true` in the native schema. Otherwise, the translator throws a parsing error.

The following is a sample nxsd file:

```
<?xml version= '1.0' encoding= 'UTF-8' ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            xmlns:tns="http://TargetNamespace.com/InboundService"
            targetNamespace="http://TargetNamespace.com/InboundService"
            elementFormDefault="qualified" attributeFormDefault="unqualified"
            nxsd:parseBom="true" nxsd:version="NXSD" nxsd:stream="chars"
 nxsd:encoding="UTF8">
    <xsd:element name="Root-Element">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="mydata" minOccurs="1" maxOccurs="unbounded"
                    nxsd:style="array"
                    nxsd:cellSeparatedBy="${eol}">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="C1" type="xsd:string"
                               nxsd:style="fixedLength"
                               nxsd:length="3"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
 </xsd:schema>
```

Multi-Byte Translation for Inbound and Outbound Native Data

The Native Format Builder processes inbound fixed-length native data in terms of characters. The translator internally uses *character* data. With support for Multi-Byte streaming in 12c, you can specify the length of a field in terms of bytes instead of characters. This enables the inbound Oracle translator to read the data as bytes and convert them into characters based on the encoding used. Similarly, the outbound Translator converts the XML Data into bytes (and not characters) and writes them to the output stream.

The Initial Problem

However, you might want to specify the field length as *bytes*, and to process fixed length data where length has to be specified in terms of *bytes*. For fixed length binary data, such specifying is very useful because specifying the length in characters would not be as helpful.

This not a problem with a single byte character set; however, it is a problem with a multi byte character set as the length in bytes is *not equal* to the length in characters.

Solution

To solve this problem and to enable you to specify the field length in terms of bytes, the NXSD translator supports specifying the length of the field in terms of bytes. Specifically, you can specify the length unit as either `Byte` or `Char`. The default value is `Char`. To specify this unit length, use the following convention:

```
nxsd:lengthUnits="Byte|Char"
```

`Byte` specifies that the length of the field is to be translated in terms of Bytes, and `Char` indicates that the length of the field is specified in terms of character.

Note that you must make *manual* changes to the files to provide for both inbound and outbound multi-byte translation. You cannot do this through the Native Format Builder User Interface.

Specifying Padded Data

You can also specify inbound data as padded data. When you do this, the translator strips off all the occurrences of padded data.

To use this feature, you must let the translator know the format of the padded Byte data it is stripping off. Use the following convention to do so:

```
nxsd:paddedDataType="decimal|binary|octal|hexbinary|string" nxsd:paddedBy="XYZ"
```

See the following table for a description of the padded Data Type formats that you can specify as inputs.

Format	Description
Decimal	Padded data specified is in decimal format. You must specify the decimal values corresponding to the padded bytes. Each decimal digit is expressed in 'nnn' format for example, 32 is expressed as '032'. If you wish to specify multiple digits, then each digit must follow the same format; for example, 001002123 would mean 3 bytes (1, 2 and 123) `nxsd:paddedDataType="decimal" nxsd:padStyle="head" nxsd:paddedBy='032032'`
Binary	Padded data specified is in Binary format. You must specify the binary values corresponding to the padded bytes. If you want to specify more than one byte as padded data, then the length corresponding to each byte must be eight. For example: `nxsd:paddedDataType="binary" nxsd:padStyle="head" nxsd:paddedBy='0010000000100000'/`

Format	Description
Oct	Padded data specified is in Octal format. You must specify the octal values corresponding to the padded bytes. If you want to specify more than one byte as padded data then the length corresponding to each byte must be three. For example, `nxsd:paddedDataType="octal" nxsd:padStyle="head" nxsd:paddedBy='040040'`
hexbinary	Padded data specified is in Hexadecimal format. You have to specify the hexbinary values corresponding to the padded bytes. If you want to specify more than one byte as padded data, then the length corresponding to each byte must be two. For example, `nxsd:paddedDataType="hexBinary" nxsd:padStyle="head" nxsd:paddedBy='2020'`
string	Padded data specified is in String format. Translator would use the encoding specified at schema level (nxsd:encoding). For example, `nxsd:paddedDataType="string" nxsd:padStyle="head"nxsd:paddedBy='`

Note:

Note that the SJIS character can be used in padding only when `paddedDataType`="string"

Specifying a Prefix or a Suffix

You can also specify the prefix and suffix which could be appended to the incoming data before publishing to the XML To do, use the following convention:

```
nxsd:prefixWith="XXXX" nxsd:suffixWith="YYYY"
```

Note:

While reading/writing data in byte mode in FixedLength style, the translator does not support specifying a prefix and suffix with base64Binary and hexBinary data.

Translator Behavior with Multi-Stream Data

The behavior of the Translator depends upon the type of data to be published to the XML. For example ,you might specify the type as

`type="xsd:string|xsd:hexBinary|xsd:base64Binary"`

If data type is specified as `string`, the Translator reads the specified number of bytes for the input stream.

However, if the input stream does not contain sufficient data, the translator throws an exception.

Once the Translator has read the specified number of bytes, it strips off all padded data from the read bytes. After removing the padded data, Translator converts remaining bytes to Character.

If the remaining bytes do not match the character boundary, or if the remaining bytes cannot be converted to Character for any reason, the Translator throws an exception.

If the data is successfully converted to the Characters, these Characters are published to the XML.

If data type you specify is either `hexBinary` or `base64Binary`, after removing the padded data, the Translator converts the remaining bytes to the specified data type and they are published to the XML.

Outbound Translation Behavior

For outbound translation, you can specify the field length in terms of bytes. The Translator generates the bytes corresponding to the data specified in XML

Once data is converted to bytes, the Translator first removes any prefix or suffix (if present).

For string types (that is, elements that are not defined as `xsd:hexBinary` or `xsd:base64Binary` in the schema) the prefix and suffix are removed before the value is converted to bytes for padding and writing out.

For binary types, prefix and suffix are not allowed, as previously mentioned.

The translator checks for the number of bytes generated along with the length specified in the schema. If the number of bytes generated are more than that of specified in Schema, translator throws an exception.

If the generated bytes are less than that of specified in length, the translator adds the `paddedBy` data bytes at the start or end of generated bytes as specified in schema.

Examples

The following examples provide XSD, XML and text files representing Translation.

Base 64 Binary Padded Data

The following are examples of Base 64 Binary Padded Data conversion. In the XSD, the translator is told to strip off strings using `nxsd:paddedDataType=`

```
<?xml version="1.0" encoding="SJIS"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            targetNamespace="http://xmlns.oracle.com/test"
            xmlns:tns="http://xmlns.oracle.com/test"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
              nxsd:stream="bytes" nxsd:version="NXSD"
                   nxsd:encoding="SJIS" >
  <xsd:element name="root">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1" type="xsd:base64Binary" nxsd:style="fixedLength"
nxsd:length="14"
           nxsd:lengthUnit="byte" nxsd:paddedDataType="string"
            nxsd:padStyle="head" nxsd:paddedBy=",Ì"/>
      <xsd:element name="field2" type="xsd:base64Binary"
              nxsd:style="fixedLength" nxsd:length="13"
```

```
              nxsd:lengthUnit="byte"
                  nxsd:paddedDataType="string"
                   nxsd:padStyle="tail"
                    nxsd:paddedBy=",Ì"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The XML is

```
<root xmlns="http://xmlns.oracle.com/test">
   <field1>g32Di4Ngg2+DQ4Nn</field1>
   <field2>IILMg1SDfIFbg2c=</field2>
</root>
```

Binary

This example shows conversion of multiple field binary data.

The XSD follows. Note that `nxsd:lengthUnits="Byte"` which indicates the length of the field is to be translated in terms of Bytes.

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            targetNamespace="http://style.onron.com/pcbpel/demo"
            xmlns:tns="http://style.onron.com/pcbpel/demo"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"              nxsd:stream="bytes"
nxsd:version="NXSD"
            nxsd:encoding="SJIS"
            >
  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1" type="xsd:base64Binary" nxsd:style="fixedLength"
nxsd:length="10"
               nxsd:lengthUnit="Byte"/>
                <xsd:element name="field2"
              type="xsd:base64Binary" nxsd:style="fixedLength"
                     nxsd:length="10"
                    nxsd:lengthUnit="Byte"/>
        <xsd:element name="field3" type="xsd:hexBinary"
                nxsd:style="fixedLength" nxsd:length="20"
                 nxsd:lengthUnit="Byte"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The XML is:

```
<Root-Element xmlns="http://style.onron.com/pcbpel/demo">
   <field1>gqCCooKkICAgIA==</field1>
   <field2>gqmCq4KtICAgIA==</field2>
   <field3>82BD82BF82C282C482C620202020202020202020
                   </field3>
</Root-Element>
```

Shift JIS Encoding

The following provide an example of shift JIS encoding in bytes.

The XSD is:

```
<?xml version="1.0" encoding="SJIS"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/
                                pcbpel/nxsd"
            targetNamespace="http://xmlns.oracle.com/
                               test"
            xmlns:tns="http://xmlns.oracle.com/test"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
              nxsd:stream="bytes"
               nxsd:version="NXSD"
                  nxsd:encoding="SJIS" >
  <xsd:element name="root">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1"
           type="xsd:hexBinary"
                  nxsd:style="fixedLength"
        nxsd:length="14" nxsd:lengthUnit="byte"
            nxsd:paddedDataType="string"
                     nxsd:padStyle="head"
                 nxsd:paddedBy=",Ì"/>
      <xsd:element name="field2"
                  type="xsd:hexBinary"
            nxsd:style="fixedLength"
                  nxsd:length="13"
            nxsd:lengthUnit="byte"
            nxsd:paddedDataType="string"
            nxsd:padStyle="tail"                    nxsd:paddedBy=",Ì"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The corresponding XML is

```
<root xmlns="http://xmlns.oracle.com/test">
   <field1>837D838B8360836F83438367</field1>
   <field2>2082CC8354837C815B8367</field2>
</root>
```

Identifier Length Example

The `IdentifierLength` construct indicates the number of characters and bytes in which the actual length of the data is stored, while `lengthUnit` enables you to specify the field length in bytes.

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            targetNamespace="http://style.onron.com/pcbpel/demo"
            xmlns:tns="http://style.onron.com/pcbpel/demo"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
               nxsd:stream="bytes" nxsd:version="NXSD"
            nxsd:encoding="SJIS"
            >
```

```
  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1" type="xsd:string"
              nxsd:style="fixedLength"
              nxsd:identifierLength="2"
              nxsd:lengthUnit="Byte"/>
        <xsd:element name="field3" type="xsd:string"
              nxsd:style="fixedLength"
              nxsd:identifierLength="2"
              nxsd:lengthUnit="Byte"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Identifier Example base64BInary

The following example shows a similar `Identifierlength` statement, except the type is base64Binary.

The XSD is:

```
<?xml version="1.0" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            targetNamespace="http://style.onron.com/pcbpel/demo"
            xmlns:tns="http://style.onron.com/pcbpel/demo"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified" nxsd:stream="bytes"
nxsd:version="NXSD"
            nxsd:encoding="SJIS"
            >

  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1" type="xsd:base64Binary"
nxsd:style="fixedLength" nxsd:identifierLength="2"
          nxsd:lengthUnit="Byte"/>
        <xsd:element name="field3" type="xsd:hexBinary"
          nxsd:style="fixedLength"
          nxsd:identifierLength="2" nxsd:lengthUnit="Byte"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The XML is:

```
<Root-Element xmlns="http://style.onron.com/pcbpel/demo">
   <field1>gqCCooKkICAgIA==</field1>
   <field3>82BD82BF82C282C482C6202020202020202020202020</field3>
</Root-Element>
```

Identifier-Padded Data with SJIS

The following shows a similar Identifier example, but with padded data (by '32') and shifted SJIS encoding.

The xsd is:

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            targetNamespace="http://style.onron.com/pcbpel/demo"
            xmlns:tns="http://style.onron.com/pcbpel/demo"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
              nxsd:stream="bytes" nxsd:version="NXSD"
              nxsd:encoding="SJIS"
            >
  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1" type="xsd:string"
           nxsd:style="fixedLength" nxsd:identifierLength="2" nxsd:lengthUnit="Byte"
nxsd:padStyle="head" nxsd:paddedBy='32'/>
        <xsd:element name="field3" type="xsd:string"
           nxsd:style="fixedLength"
           nxsd:identifierLength="2"
           nxsd:lengthUnit="Byte"
           nxsd:padStyle="tail"
           nxsd:paddedBy='32'/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Identifier-Padded Binary

Another identifier example shows padded binary data.

The xsd for this example is:

```
<?xml version="1.0" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            targetNamespace="http://style.onron.com/pcbpel/demo"
            xmlns:tns="http://style.onron.com/pcbpel/demo"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified" nxsd:stream="bytes"
nxsd:version="NXSD"
            nxsd:encoding="SJIS"
            >
  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1" type="xsd:base64Binary" nxsd:style="fixedLength"
nxsd:identifierLength="2"
          nxsd:lengthUnit="Byte" nxsd:padStyle="head"
           nxsd:paddedBy='32'/>
        <xsd:element name="field3" type="xsd:hexBinary"
           nxsd:style="fixedLength" nxsd:identifierLength="2" nxsd:lengthUnit="Byte"
nxsd:padStyle="tail"
           nxsd:paddedBy='32'/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The XML is:

```
<Root-Element xmlns="http://style.onron.com/pcbpel/demo">
   <field1>gqmCq4KC</field1>
   <field3>82BD82BF82C282C482C6</field3>
</Root-Element>
```

Padded Multibyte Binary Element

The XSD for padded multi-byte binary is:

```
<?xml version="1.0" ?>

 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
             targetNamespace="http://style.onron.com/pcbpel/demo"
             xmlns:tns="http://style.onron.com/pcbpel/demo"
             elementFormDefault="qualified"
             attributeFormDefault="unqualified"
                 nxsd:stream="bytes"
                 nxsd:version="NXSD"
             nxsd:encoding="SJIS"
             nxsd:validateNxsd="true">

  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1" type="xsd:base64Binary" nxsd:style="fixedLength"
nxsd:length="10" nxsd:lengthUnit="Byte" nxsd:paddedDataType="binary"
nxsd:padStyle="head" nxsd:paddedBy='0010000000100000'/>

        <xsd:element name="field2" type="xsd:hexBinary"
           nxsd:style="fixedLength" nxsd:length="20"
           nxsd:lengthUnit="Byte" nxsd:padStyle="tail"
          nxsd:paddedDataType="binary"
          nxsd:paddedBy='0010000000100000'/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The XML is:

```
<Root-Element xmlns="http://style.onron.com/pcbpel/demo">
   <field1>gqmCq4Kt</field1>
   <field2>82BD82BF82C282C482C6</field2>
</Root-Element>
```

Padded Multi-Byte Decimal

The padded multi-byte decimal XML is:

```
<?xml version="1.0" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
             targetNamespace="http://style.onron.com/pcbpel/demo"
             xmlns:tns="http://style.onron.com/pcbpel/demo"
             elementFormDefault="qualified"
             attributeFormDefault="unqualified"
                nxsd:stream="bytes"
                nxsd:version="NXSD"
```

```
                nxsd:encoding="SJIS">

  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1" type="xsd:base64Binary" nxsd:style="fixedLength"
nxsd:length="10"
         nxsd:lengthUnit="Byte"
         nxsd:paddedDataType="decimal"
         nxsd:padStyle="head"
         nxsd:paddedBy='032032'/>
        <xsd:element name="field2" type="xsd:hexBinary"
        nxsd:style="fixedLength" nxsd:length="20"
         nxsd:lengthUnit="Byte" nxsd:padStyle="tail"
         nxsd:paddedDataType="decimal"
         nxsd:paddedBy='032032'/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

SOSI Support

The Translator also provides support to escape bytes (SHIFT_OUT, SHIFT_IN) from the input stream.

This support is specifically provided to support SJIS encoding, where, in mixed Double Byte Character Set mode, SHIFT_OUT and SHIFT_IN bytes are inserted to the data to indicate the change of mode (from single byte to double byte).

To use SOSI support with the Translator, use the following convention:

`nxsd:escapeBytes="sosi"`

Note that the SJIS character can be used in padding only when `paddedDataType="string"`

Translator XPath Functions

The Translator XPath functions can translate data from a native format (such as CSV, fixed-length, tab-delimited, and COBOL Copybook formats) to an XML format and from an XML format to a native format. The translator XPath functions are of two types, streaming and non-streaming.

The difference is that the streaming translator XPath functions implement the batching transformation approach while the non-streaming XPath functions do not implement the batching transformation approach. With the batching transformation approach, files that are of the order of a few gigabytes (GB) can be processed without running into memory issues.

This section includes the following topics:

Terminologies

This section describes the terminologies that you must understand for using translation XPath functions.

Attachment Element

An attachment element unusually refers to the actual content elsewhere by using an `"href"` attribute. The actual content may be present in a file system or in a database table.

An attachment is usually represented by using the following schema construct:

```
<element name="hrefelement">
    <complexType>
        <attribute name="href" type="string"/>
    </complexType>
</element>
```

The `"href"` attribute contains the actual location of the data being referred to. It can contain the path to a file in the file system or a pointer (primary key) to a database entity.

Scalable DOM

Scalable DOM (SDOM), from Oracle XML Developer Kit (Oracle XDK), provides scalable and pluggable support for DOM. This removes problems of memory inefficiency, limited scalability, and lack of control over the DOM configuration.

Using the lazy materialization mechanism, Oracle XDK only creates nodes that are accessed and frees unused nodes from memory. Applications can process very large XML documents with improved scalability.

Translator XPath Functions

A translator may be required while reading and writing files. This section discusses the following translator XPath functions:

- doTranslateFromNative Function
- doTranslateToNative Function
- doStreamingTranslate Function

doTranslateFromNative Function

The `doTranslateFromNative` XPath function translates input data into XML. The input data can be a string, an attachment element, or a base64Binary element.

`ora:dotranslateFromNative('input','nxsdTemplate','nxsdRoot','targetType','attachment element?')`

The following table describes the parameters used in the syntax for using this function:

Parameter	Description
`input`	Input data for the XPath function; the data can either be a string data that must be translated, an Oracle File or FTP Adapter attachment, an attachment referring to an external file path, or a base64Binary element.
nxsdTemplate	NXSD schema to use to translate the input data into XML format.
nxsdRoot	Root element in the NXSD schema.

Parameter	Description
targetType	This parameter des how the XPath function translates the native data into XML. Must be set to either `'DOM'`, or `'ATTACHMENT'` or `'SDOM'`. If the `targetType` parameter is: • `'DOM'`, then the translated data is returned as a DOM. • `'ATTACHMENT'`, then the translated data is returned as an attachment. If the optional parameter (`attachmentElement`) is available to the XPath function, then the XPath function uses the corresponding `href` attribute to write the translated XML. However, if the parameter is absent, then the XPath function creates a new database-backed attachment and returns that. See the following example for more details. • `'SDOM'`, then the translated data is returned as SDOM. You must use this if the returned XML file is huge.
attachmentElement	This parameter is optional. This is the attachment for the returned XML file.

Example - Configuring the XPath Function When the Input Data Is of String Type and Must Be Converted to an XML DOM

```
<variables>
     <variable…/>
    <variable name="csv_data" type="xsd:string"/>
  </variables>
<assign name="assignCSVData">
      <copy>
        <from expression="'this, is, csv, data…'"/>
        <to variable="csv_data"/>
      </copy>
</assign>

<assign name="doTranslateFromNativeCall">
      <copy>
        <from expression="ora: doTranslateFromNative
(bpws:getVariableData('csv_data'),                  'xsd/address-
csv.xsd',                    'Root-Element','DOM')"/>
        <to variable="returnVariable" query="/ns1:Root-Element"/>
      </copy>
    </assign>
```

In this example:

- `csv_data` is a BPEL variable containing CSV data to be translated into XML.
- `xsd/address-csv.xsd` is relative path to the NXSD schema in the project.
- `Root-Element` is a root element in the NXSD schema (This is optional.)
- `returnVariable` is the returned XML data as DOM.

Example - When the Input data is an Attachment, Which Must Be Translated to a DOM

1. Define `attachmentElement` in the schema of the BPEL process, as follows:

```
<schema targerNamespace="…">
        <element name="attachmentElement">
```

```
            <complexType>
                <attribute name="href" type="string"/>
            </complexType>
        </element>
</schema>
```

2. Create a variable of type attachment element in the schema of the BPEL process, as follows:

```
  <variables>
     <variable…/>
    <variable name="attachmentVariable"
type="client:attachmentElement"/>
  </variables>
```

3. Assign the source file path that you must translate, as follows:

```
<assign name="AssignAttachmentReference">
  <copy>
    <from expression="'/tmp/xpath/in/address.csv'"/>
    <to variable="attachmentVariable"
            query="/client:attachmentElement/@href"/>
  </copy>
</assign>
```

4. Call the XPath function, as follows:

```
<assign name="xlateFromNative">
  <copy>
    <from expression="ora:doTranslateFromNative
(bpws:getVariableData('attachmentVariable'),          'xsd/address-csv.xsd',
'Root-Element', 'DOM')"/>
    <to variable="returnVariable" query="/ns1:Root-Element"/>
  </copy>
</assign>
```

In this example:

- `attachmentVariable` is an attachment variable in BPEL referring to the source file path.
- `xsd/address-csv.xsd` is the relative path to the NXSD schema in the project.
- `Root-Element` is a root element in the NXSD schema.
- `returnVariable` is the XML data returned as DOM.

Example - Configuring XPath Function When the Input Data Is Base64-encoded and Must Be Translated to DOM

1. Define the base64-encoded element in the schema of the BPEL process, as follows:

```
<schema targerNamespace="…">
  <element name="mtomElement" type="base64Binary"/>
</schema>
```

2. Create a variable of type mtom element in the schema of the BPEL process, as follows:

```
<variables>
     <variable…/>
    <variable name="encodedData" type="client:mtomElement"/>
</variables>
```

3. Assign the source file path that you must translate, as follows

```
<assign name="assignBase64EncodedData">
      <copy>
        <from expression="'b3JhY2xl'"/>
        <to variable="encodedData"              query="/client:mtomElement"/>
      </copy>
</assign>

<assign name="doTranslateFromNativeCall">
      <copy>
        <from expression="ora: doTranslateFromNative
(bpws:getVariableData('encodedData'), 'xsd/address-csv.xsd','Root-
Element','DOM')"/>
        <to variable="returnVariable"
                    query="/ns1:Root-Element"/>
      </copy>
    </assign>
```

In this example:

- `mtomElement` is a BPEL variable containing base64-encoded data to be translated into XML.
- `xsd/address-csv.xsd` is the relative path to the NXSD schema in the project.
- `Root-Element` is a root element in the NXSD schema.
- `returnVariable` is the XML data returned as DOM.

Example - Configuring XPath Function When the Input Data Is of String Type, Which Must Be Translated to an Attachment Referred to by a File-Path

1. Define `attachmentElement` in the schema of the BPEL process, as follows:

```
<schema targerNamespace="…">
        <element name="attachmentElement">
            <complexType>
                <attribute name="href" type="string"/>
            </complexType>
        </element>
</schema>
```

2. Create an input variable of type string and an output variable of type attachment in the schema of the BPEL process, as follows:

```
<variables>
    <variable…/>
    <variable name="csv_data" type="xsd:string"/>
    <variable name="returnAttachmentVariable"
            type="client:attachmentElement"/>
</variables>
```

3. Assign the CSV data that you must translate, as follows:

```
<assign name="assignCSVData">
    <copy>
      <from expression="'this, is, csv, data…'"/>
      <to variable="csv_data"/>
    </copy>
</assign>
```

4. Populate the attachment with the path of the file where you want the translated data to be stored, as follows:

```
<assign name="AssignAttachmentReferenceForOutput">
    <copy>
      <from expression="'/tmp/xpath/output/address.xml'"/>
      <to variable=" returnAttachmentVariable "
            query="/client:attachmentElement/@href"/>
    </copy>
</assign>
```

5. Call the XPath function as follows:

```
<assign name="doTranslateFromNativeCall">
    <copy>
      <from expression="ora: doTranslateFromNative
      (bpws:getVariableData('csv_    data'),'xsd/address-csv.xsd','Root-
Element','ATTACHMENT',   bpws:getVariableData('returnAttachmentVariable'))"/>
      <to variable="returnAttachmentVariable"/>
    </copy>
</assign>
```

In this example:

- `csv_data` is a BPEL string variable containing CSV data to be translated into XML.
- `xsd/address-csv.xsd` is the relative path to the NXSD schema in the project.
- `Root-Element` is a root element in the NXSD schema.
- `returnAttachmentVariable` is the returned attachment.

Note: In this example, `targetType` is set to `ATTACHMENT`, and `returnAttachmentVariable` points to the file path where the translated XML is to be written.

However, the fifth parameter (`bpws:getVariableData('returnAttachmentVariable')`) is optional. If this parameter is missing, then the XPath function creates a database-backed attachment and returns it. In such a case, the XPath function is configured, as follows:

1. Define `attachmentElement` in the schema of the BPEL process, as follows:

```
<schema targerNamespace="…">
        <element name="attachmentElement">
            <complexType>
                <attribute name="href" type="string"/>
            </complexType>
        </element>
</schema>
```

2. Create an input variable of type string and an output variable of type attachment in the schema of the BPEL process, as follows:

```
<variables>
    <variable…/>
    <variable name="csv_data" type="xsd:string"/>
    <variable name="returnAttachmentVariable"
        type="client:attachmentElement"/>
</variables>
```

3. Assign the CSV data that you must translate, as follows:

```
<assign name="assignCSVData">
    <copy>
      <from expression="'this, is, csv, data…'"/>
      <to variable="csv_data"/>
    </copy>
</assign>

<assign name="doTranslateFromNativeCall">
      <copy>
        <from expression="ora: doTranslateFromNative
(bpws:getVariableData('csv_data'),'xsd/address-csv.xsd','Root-
Element','ATTACHMENT')"/>
        <to variable="returnAttachmentVariable"/>
      </copy>
    </assign>
```

After the XPath call returns, the `returnAttachmentVariable` variable is populated with the `href` attribute pointing to the GUID representing the database-backed attachment.

Note: If the data being translated is huge, then you must use either `ATTACHMENT` or `SDOM` as the targetType parameter for the XPath function.

doTranslateToNative Function

The `doTranslateToNative` XPath function translates an input DOM into string data or an attachment.

Syntax

`ora:dotranslateToNative('`*`input`*`','`*`nxsdTemplate`*`','`*`nxsdRoot`*`','`*`targetType`*`','`*`attachmentElement`*`?')`

The following table describes the parameters used in the syntax for using this function:

Table 6-8 do TranslateToNative Function Parameters

Parameter	Description
input	Input data for the XPath function; the data can either be DOM or SDOM data that must be translated to a native format such as CSV.
nxsdTemplate	NXSD schema to be used to translate the input data into XML format.
nxsdRoot	Name of the root element in the NXSD schema.
targetType	This parameter des how the XPath function translates the XML data into native formats. Must be set to either `'STRING'`, or `'ATTACHMENT'`. If the `targetType` parameter is: • `STRING`, then the translated data is returned as a string. • `ATTACHMENT`, then the translated data is returned as an attachment. If the optional parameter (`attachmentElement`) is available to the XPath function, then the XPath function uses the corresponding `href` attribute to write the translated native data. However, if the parameter is absent, then the XPath function creates a new database-backed attachment and returns that. See the following example for more details.

Table 6-8 (Cont.) do TranslateToNative Function Parameters

Parameter	Description
attachmentElement	This parameter is optional. This is the attachment to which the translated data is written.

Example - Configuring the XPath Function When the Input Data Is of XML Format and Must Be Translated Into CSV String Format

```
<variables>
     <variable…/>
   <variable name="inputDOM" type="ns1:Root-Element"/>
          <!- - data that must be translated into native - ->
    <variable name="returned_csv_data"
       type="xsd:string"/>
  </variables>

<assign name="doTranslateToNativeCall">
      <copy>
        <from expression="ora: doTranslateToNative (bpws:getVariableData('inputDOM'),
        'xsd/address-csv.xsd','Root-Element','STRING')"/>
        <to variable="returned_csv_data"/>
      </copy>
    </assign>
```

In this example:

- `inputDOM` is a BPEL DOM variable containing XML data to be translated into string data representing the translated CSV.
- `xsd/address-csv.xsd` is the relative path to the NXSD schema in the project.
- `Root-Element` is a root element in the NXSD schema.
- `return_csv_data` is the string variable that contains the translated CSV data.

Example - Configuring XPath Function to Translate an Incoming XML DOM into an Attachment Representing the Target File-Path for the Translated CSV

1. Define `attachmentElement` in the schema of the BPEL process, as follows:

```
<schema targerNamespace="…">
        <element name="attachmentElement">
            <complexType>
                <attribute name="href"
                    type="string"/>
            </complexType>
        </element>
</schema>
```

2. Create an input variable of type `attachmentElement` in the schema of the BPEL process, as follows:

```
<variables>
     <variable…/>
    <variable name="inputDOM"
        type="ns1:Root-Element"/>
    <variable name="attachmentVariable"
```

```
        type="client:attachmentElement"/>
  </variables>
```

3. Assign the target file path where you want the translated CSV to be written, as follows:

```
<assign name="AssignAttachmentReference">
  <copy>
     <from expression=
                    "'/tmp/xpath/out/address.csv'"/>
     <to variable="attachmentVariable"
            query="/client:attachmentElement/@href"/>
  </copy>
</assign>
```

4. Call the XPath function, as follows:

```
<assign name="xlateToNative">
      <copy>
        <from expression="ora:doTranslateToNative
            (bpws:getVariableData('inputDOM'),'xsd/address-csv.xsd'        'Root-
Element', 'ATTACHMENT', bpws:getVariableData
           (' attachmentVariable'))"/>
        <to variable="attachmentVariable"/>
      </copy>
    </assign>
```

In this example:

- `inputDOM` is a BPEL DOM variable containing XML data to be translated into a CSV output file represented by `/tmp/xpath/out/address.csv`.
- `xsd/address-csv.xsd` is the relative path to the NXSD schema in the project.
- `Root-Element` is a root element in the NXSD schema.
- `AttachmentElement` points to the target output file path represented by `/tmp/xpath/out/address.csv`.

Note: In this example, `targetType` is set to `ATTACHMENT`, and `AttachmentVariable` points to the file path where the translated CSV file is to be written.

However, the fifth parameter (`bpws:getVariableData('attachmentVariable')`) is optional. If this parameter is missing, then the XPath function creates a database-backed attachment and returns it. In such a case, the XPath function is configured as follows:

1. Define `attachmentElement` in the schema of the BPEL process, as follows:

```
<schema targerNamespace="…">
        <element name="attachmentElement">
            <complexType>
                <attribute name="href" type="string"/>
            </complexType>
        </element>
</schema>
```

2. Create an input variable of type `attachmentElement` in the schema of the BPEL process, as follows:

```
<variables>
     <variable…/>
```

```
    <variable name="inputDOM" type="ns1:Root-Element"/>
    <variable name="attachmentVariable"
       type="client:attachmentElement"/>
  </variables>
```

3. Call the XPath function, as follows:

```
<assign name="xlateToNative">
      <copy>
        <from expression="ora:doTranslateToNative
        (bpws:getVariableData('inputDOM'),
         'xsd/address-csv.xsd', 'Root-Element',
           'ATTACHMENT')"/>
        <to variable="attachmentVariable"/>
      </copy>
    </assign>
```

After the XPath call returns, `attachmentVariable` is populated with the `href` attribute pointing to the GUID representing the database-backed attachment.

doStreamingTranslate Function

XPath functions implement the batching transformation approach. With this approach, files that are of the order of a few gigabytes (GB) can be processed without running into memory issues. Arbitrarily large payloads can be handled because the transformation engine does not store the result of the transformation in its memory. The transformation engine flushes its memory after a batch of elements of the large file is processed. The default batch size is 10000, which is the number of elements after which the transformation engine flushes its memory. This parameter is used internally and is optional.

Note:

Batching transformation approach is supported for XML documents that have repeating structures only.

Syntax

`ora:doStreamingTranslate('`*`input`*`','`*`streamingXpathContext`*`','`*`targetType`*`','`*`attachmentElement`*`?')`

Table 6-9 describes the parameters used in the syntax for using this function:

Table 6-9 doStreamingTranslate Parameters

Parameter	Description
input	Input data for the XPath function; the data can either be SDOM or an Attachment element.
`streamingXpathContext`	DOM representing the XPath context.
targetType	This parameter des how the XPath function translates the input data into an attachment. This must be set to either `SDOM` or `ATTACHMENT`.
attachmentElement	This parameter is optional. This is the attachment to which the data is streamed.

Table 6-9 (Cont.) doStreamingTranslate Parameters

Parameter	Description
sourceDocumentType	Set this to ATTACHMENT if the source element points to an `attachmentElement`.

The `streamingXPathContext` parameter specifies the context for the streaming transformation and, it must conform to the following schema element:

Example - Schema Element to Which streamingXPathContext Must Conform

```
<schema targetNamespace="…">
    <element name="streamingcontext">
        <complexType>
            <sequence>
                <element name="sourceSchema"
                    type="string"/>
                <element name="sourceRootElement"
                    type="string"/>
                <element name="sourceType"
                    type="string"/>
                <element name="sourceDocumentType"
                    type="string"/>
                <element name="xsl"
                    type="string"/>
                <element name="targetSchema"
                    type="string"/>
                <element name="targetRootElement"
                    type="string"/>
                <element name="targetType" type="string"/>
                <element name="batchSize" type="string"/>
            </sequence>
        </complexType>
    </element>
</schema>
```

Table 6-10 streamingXPathContext Path Parameters

Parameter	Description
sourceSchema	Source NXSD schema used to translate a native data to XML.
sourceRootElement	Source NXSD schema used to translate a native data to XML
sourceType	Set this to either `xml` or `native` depending on the input data.
xsl	Relative path of the xsl.
targetSchema	Target NXSD schema used to translate an XML into native data.
targetRootElement	Name of root element in target NXSD schema.
targetType	Set this to either `xml` or `native` depending on the output data.
batchSize	The number of elements after which the transformation engine flushes its memory.

Batching Transformation Features

This section discusses the following features of batching transformation:

Applicability

Batching transformation is applicable to:

- Documents with repeating structure
- XSLTs not requiring aggregation across entire document

Batched Invocation of XSLT Engine

The following procedure highlights the batched invocation of the XSLT engine:

1. Splitting the source document into multiple batches of one or more records
2. Performing the XSLT transformation one batch at a time
3. Combining the result of the XSLT invocation to a single target document

Splitting or Combining Performed on the Fly

The source documents are split and the results are combined into a target document:

- Without any intermediate memory or disk storage
- Through pipelining or intercepting SAX events

Low In-Memory Footprint

Batching transformation method uses low memory for the following tasks:

- Transforming arbitrarily large XML documents, which are constrained by the target system
- For standalone tests, 540 MB is transformed in less than 3 minutes

The example below implements the FlatStructure File Adapter sample using streaming transformation XPath functions. This sample use case translates the inbound native attachment from a CSV format to an XML format, and then applies the user-supplied XSL file to the resulting XML file. The transformed XML file is then translated into a fixed-length content represented by an attachment.

Example - Using Streaming Transformation XPath Function

1. Define attachmentElement, as shown in the example below.

   Example - Defining attachmentElement

```
<schema targerNamespace="…">
    <element name="attachmentElement">
        <complexType>
           <attribute name="href"
              type="string"/>
        </complexType>
    </element>
</schema>
```

2. Create a variable for the input attachment referring to the inbound csv file and the output attachment referring to the output fixed-length file. Create the variable corresponding to the streaming context. You must populate this variable before making a call to the XPath function.

Example - Creating the Variable for the Input Attachment

```
  <variables>
    <variable name="xlationContext"
            element="client:streamingcontext"/>
    <variable name="inputAttachment"
            element="client:attachmentElement"/>
    <variable name="returnAttachment"
            element="client:attachmentElement"/>
  </variables>

 <!- - Assign the input and output attachments - ->
    <assign name="assignValuesForAttachments">
      <copy>
        <from expression="'/tmp/xpath/in/address.csv' "/>
        <to variable="inputAttachment"
          query="/client:attachmentElement/@href"/>
      </copy>
      <copy>
        <from expression="
                '/tmp/xpath/out/address_fixedLength.txt' "/>
        <to variable="returnAttachment"
            query=
             "/client:attachmentElement/@href"/>
      </copy>
    </assign>
<!- - Assign the streaming context - ->
    <assign name="AssignStreamingContext">
      <copy>
        <from expression="'xsd/address-csv.xsd'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext
                      /client:sourceSchema"/>
      </copy>
      <copy>
        <from expression="'Root-Element'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/
                    client:sourceRootElement"/>
      </copy>
      <copy>
        <from expression="'native'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/client:
                  sourceType"/>
      </copy>
      <copy>
        <from expression=
                  "'xsd/address-fixedLength.xsd'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/
                   client:targetSchema"/>
      </copy>
      <copy>
        <from expression="'Root-Element'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/client:
                   targetRootElement"/>
      </copy>
      <copy>
        <from expression="'native'"/>
```

```
        <to variable="xlationContext"
            query="/client:streamingcontext/client:
                   targetType"/>
      </copy>
      <copy>
        <from expression="'xsl/addr1Toaddr2.xsl'"/>
        <to variable="xlationContext"
            query="/client
                   :streamingcontext/client:xsl"/>
      </copy>
      <copy>
        <from expression="'10000'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/client:
                   batchSize"/>
      </copy>
    </assign>
<!- - call the XPath function - ->
    <assign name="executeStreamingXPath">
      <copy>
        <from expression=
                 "ora:doStreamingTranslate
           (bpws:getVariableData('inputAttachment',
               '/client:attachmentElement'),
           bpws:getVariableData('xlationContext'),
                              'ATTACHMENT',
               bpws:getVariableData('returnAttachment'))"/>
        <to variable="returnAttachment"
                   query="/client:attachmentElement"/>
      </copy>
    </assign>
```

Use Cases for the Native Format Builder

This section describes the following use cases:

- Defining the Schema for a Delimited File Structure
- Defining the Schema for a Fixed Length File Structure
- Defining the Schema for a Complex File Structure
- Removing or Adding Namespaces to XML with No Namespace
- Defining the Choice Condition Schema for a Complex File Structure
- Defining Choice Condition With LookAhead for a Complex File Structure
- Defining Array Type Schema for a Complex File Structure
- Defining the Schema for a DTD File Structure
- Defining the Schema for a COBOL Copybook File Structure

Note:

Sampling the data with multi-character delimiter in Native Format Builder is not supported currently. The same can be achieved through hand coding the NXSD with the appropriate Delimited By string.

Defining the Schema for a Delimited File Structure

A comma-separated value (CSV) file is a common non-XML file structure.

Use the **Delimited** option in the Native Format Builder wizard, when creating the XML schema for this native file.

The `nxsd:headerLines="1"` schema attribute signifies that the first line must be treated as a header row and skipped in the native data before actually translating the rest of the data. The `nxsd:stream="chars"` schema attribute signifies that the data should be read as characters. If `nxsd:stream` is set as `bytes`, `nxsd:stream="bytes"`, then this schema attribute signifies that the native data should be read as bytes. For each of the element declarations, `Name`, `Street`, `City`, `State`, and `Country`, which have a corresponding scalar data, the `nxsd:style="terminated"` attribute defines that the corresponding data is stored in terminated style. The actual terminator is then defined by the `nxsd:terminatedBy=","` attribute specified at that construct. See Defining Terminated Data for details on the terminated style.

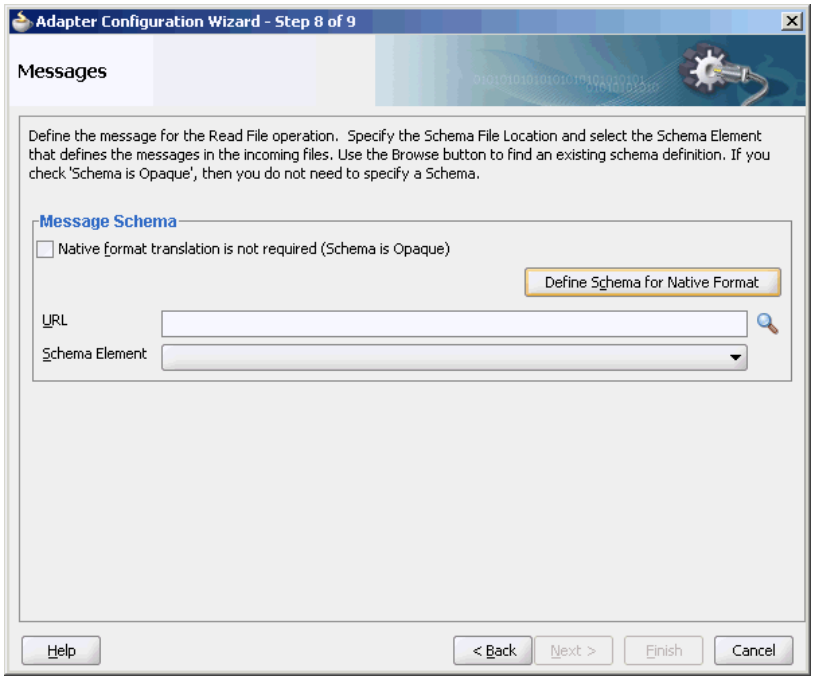
In this use case, the Native Format Builder uses a delimited sample file type that contains the address details, such as name, street, city, state, and country. Every element in this sample native file is delimited by a comma (,). You can generate the corresponding NXSD and also test it. Perform the following steps to run the use case:

1. The data in a sample text file, `address-csv.txt`, is:

```
Name,Street1,Street2,City,State,Country
Oracle India Private Limited, Lexington Towers Prestige
            St. John's Woods, 2nd Cross Road Chikka Audugodi,
               Bangalore,Karnataka, India
Intel Technology India Private Limited, Survey #23-56 P Devarabeesanahalli
Village, Outer Ring Road Varthur Hobli, Bangalore, Karnataka, India
```

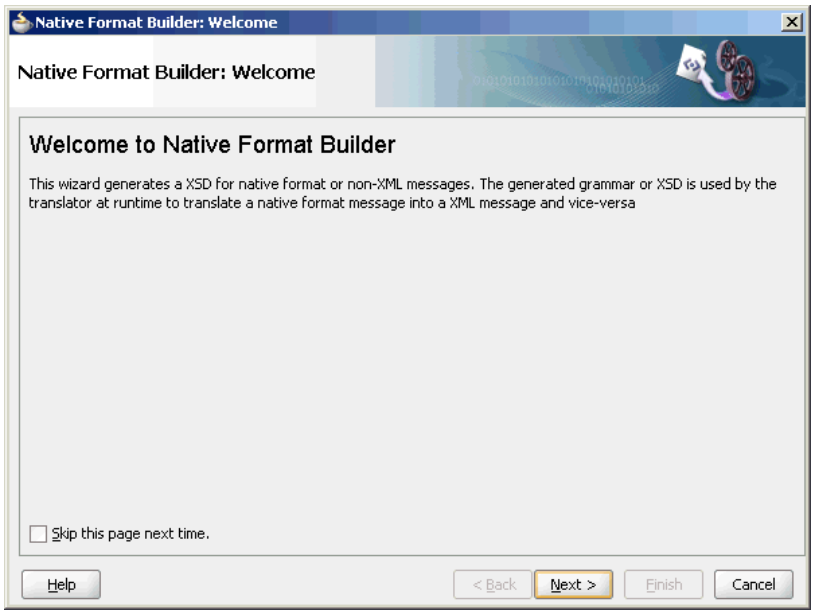
2. Navigate to the Adapter Configuration Wizard Messages page, as displayed in Figure 6-4, and click the **Define Schema for Native Format** button.

Figure 6-4 Starting the Native Format Builder Wizard



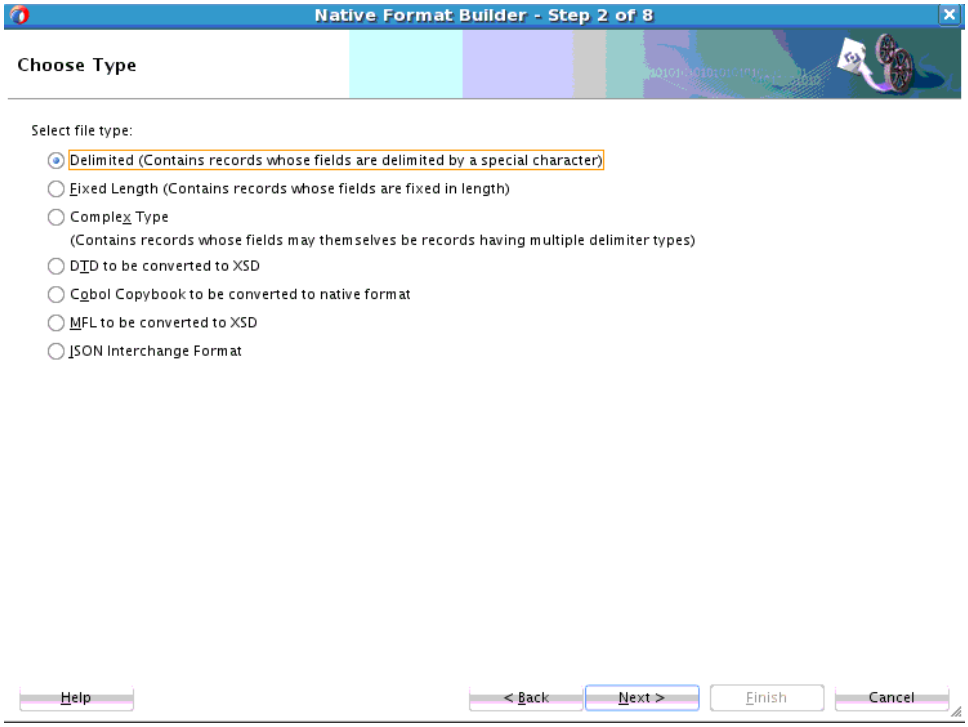
The Native Format Builder Welcome page is displayed, as shown in Figure 6-5.

Figure 6-5 Native Format Builder Wizard Welcome Page



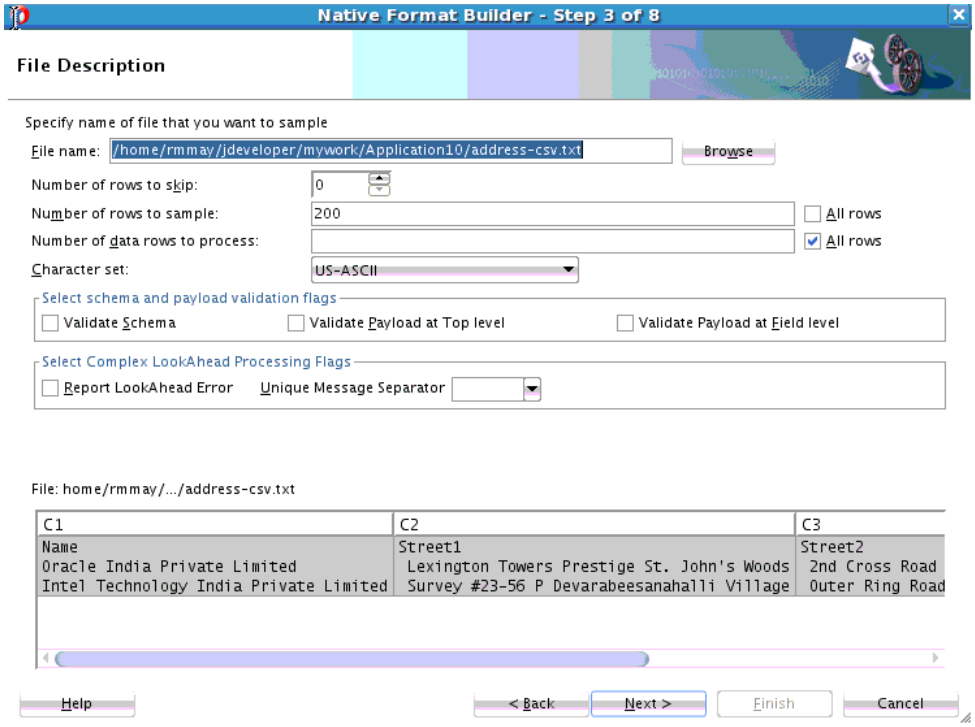
3. Click **Next**. The **Choose Type** page is displayed, as shown in Figure 6-6.

Figure 6-6 Native Format Builder Wizard Choose Type Page



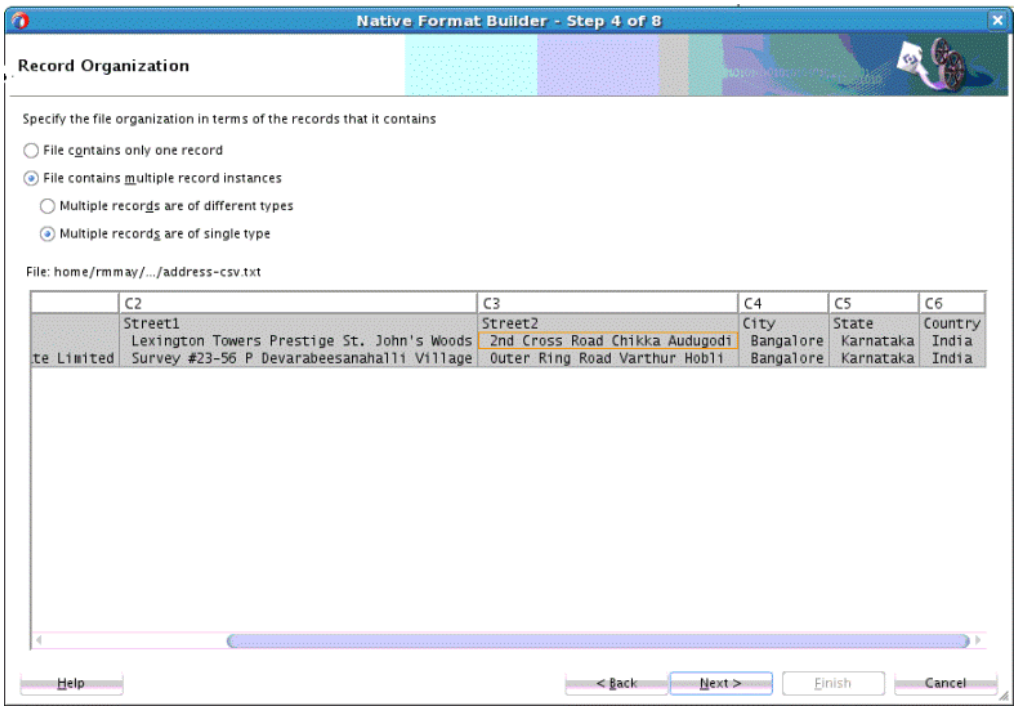
4. Click **Next**. The **Native Format Builder File Description** page is displayed.
5. Click **Browse** and select the `address-csv.txt` file, as shown in Figure 6-7.

Figure 6-7 Native Format Builder Wizard File Description Page



6. Click **Next**. The **Record Organization** page is displayed, as shown in Figure 6-8.

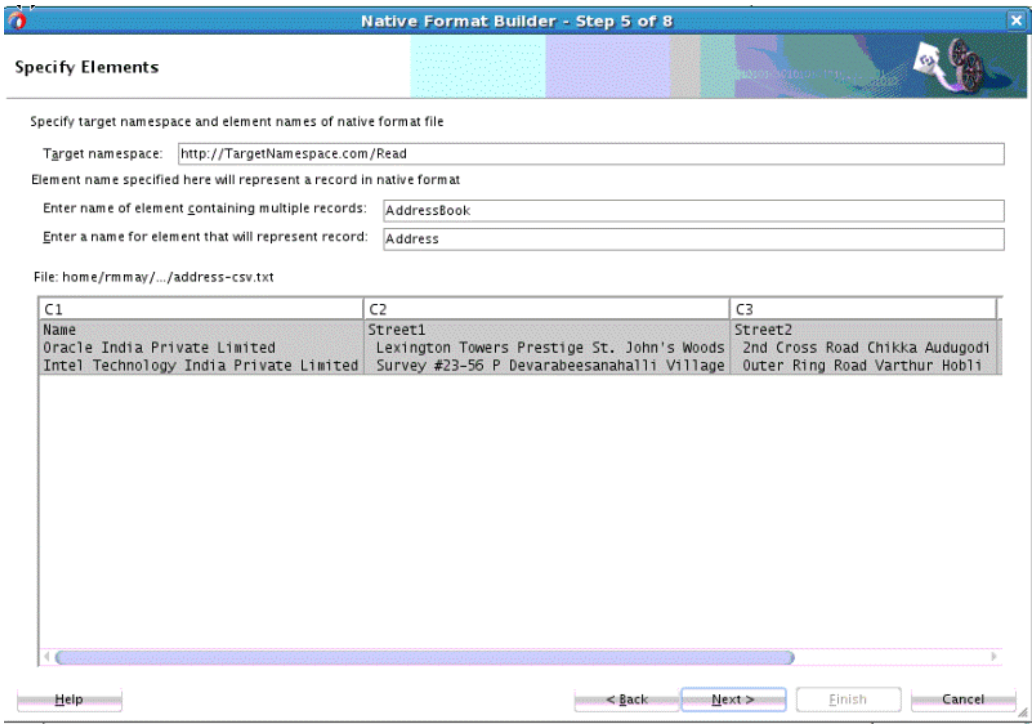
Figure 6-8 Native Format Builder Wizard Record Organization Page



7. Select **File contains multiple record instances**, then select **Multiple records are of single type**, and then click **Next**. The **Specify Elements** page is displayed.

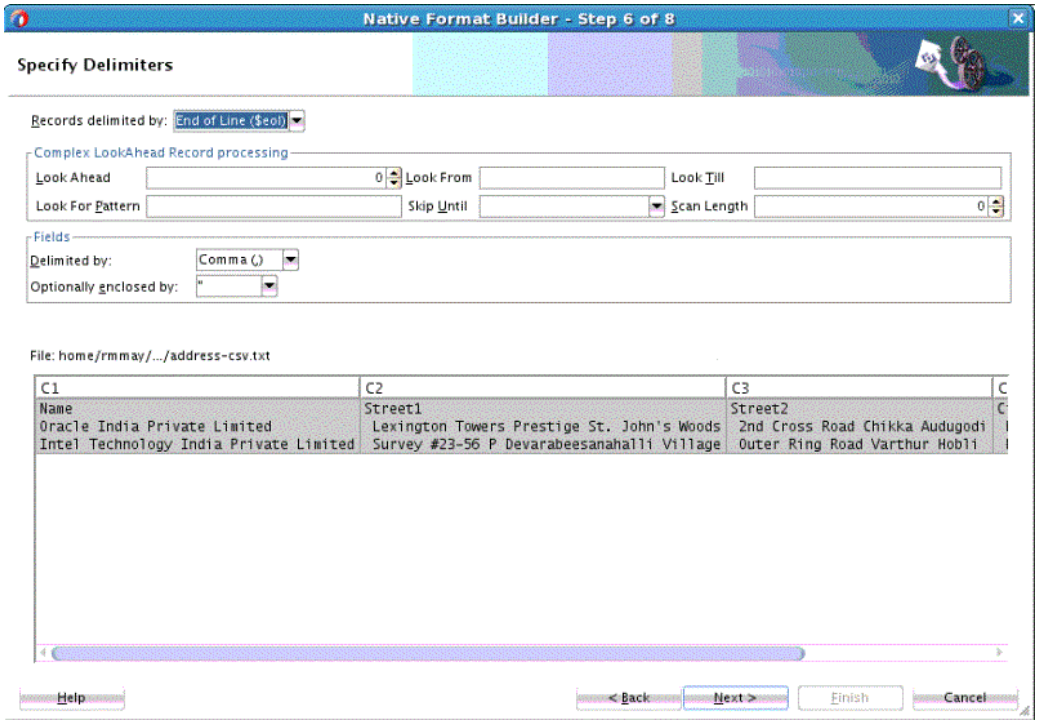
8. Enter `AddressBook` in the **Enter name of element containing multiple records** field and enter `Address` in the **Enter a name for element that represents record** field, as shown in Figure 6-9.

Figure 6-9 Native Format Builder Wizard Specify Elements Page



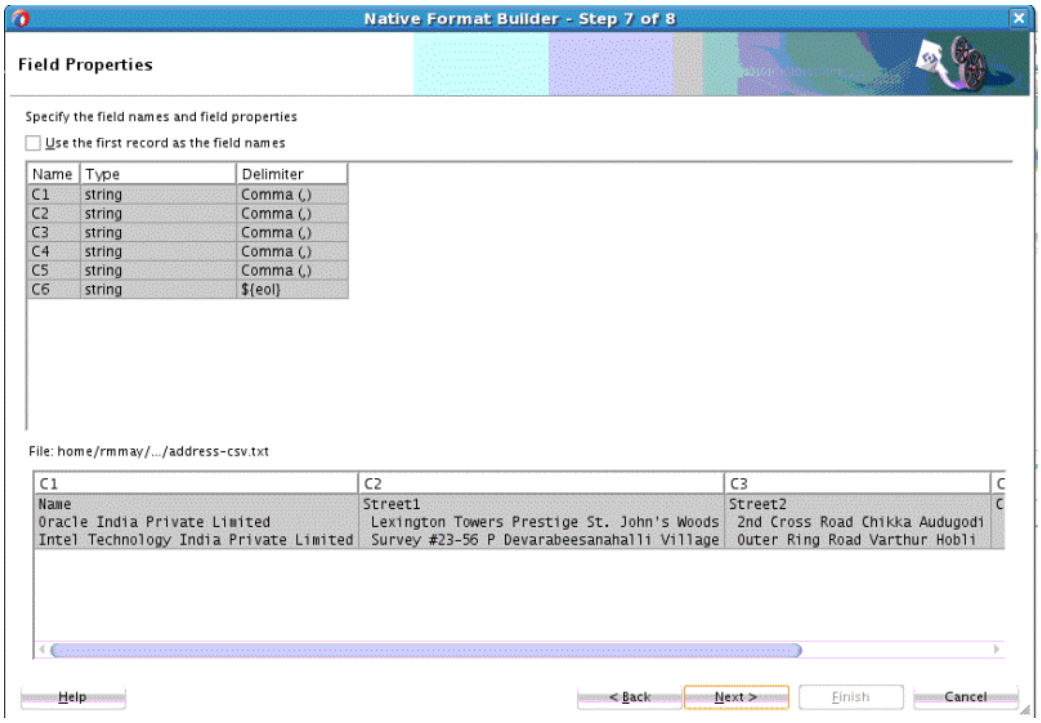
9. Click **Next**. The **Specify Delimiters** page is displayed, as shown in Figure 6-10. (For more information on Complex Lookahead strategies, see Defining Complex Look Ahead Strategies for Conditional Processing of Record Using Regular Expressions).

Figure 6-10 Native Format Builder Wizard Specify Delimiters Page



10. Ensure that the **Comma(,)** option is selected in the Delimited By field, and click **Next**. The Field Properties page is displayed, as shown in Figure 6-11.

Figure 6-11 Native Format Builder Wizard Field Properties Page

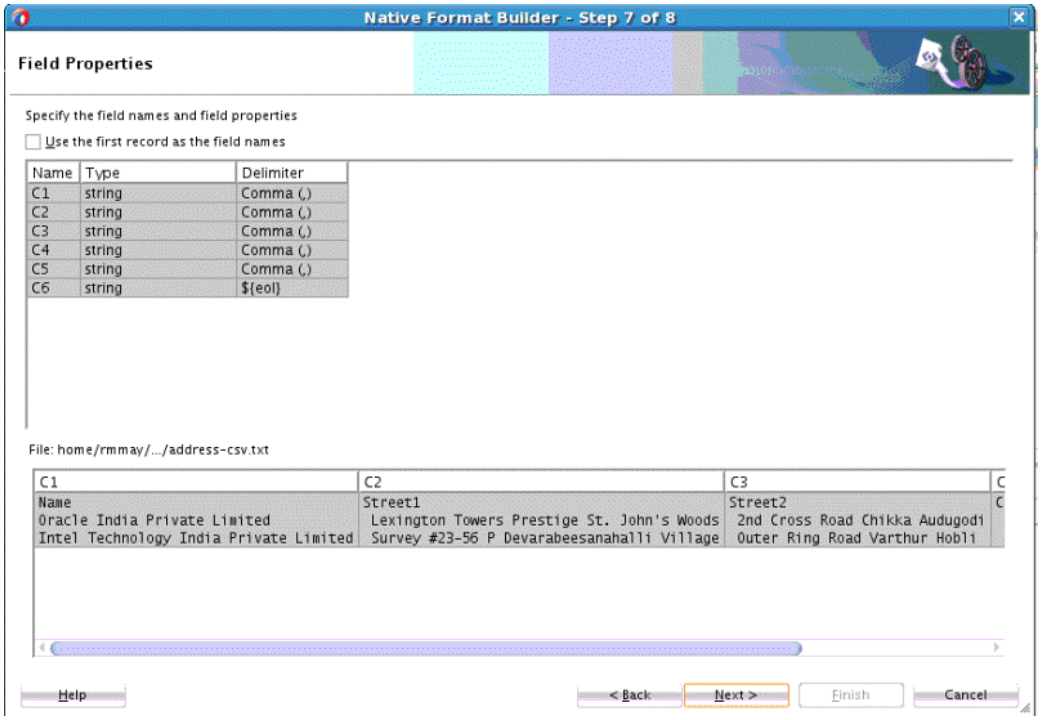


11. Select **Use the first record as the field names**, then click **Next**. The Generated Native Format File page is displayed, as shown in Figure 6-12.

Note:

The first record is used as the field name, is also treated as a header record, and is skipped during translation.

Figure 6-12 Native Format Builder Wizard Generated Native Format File Page



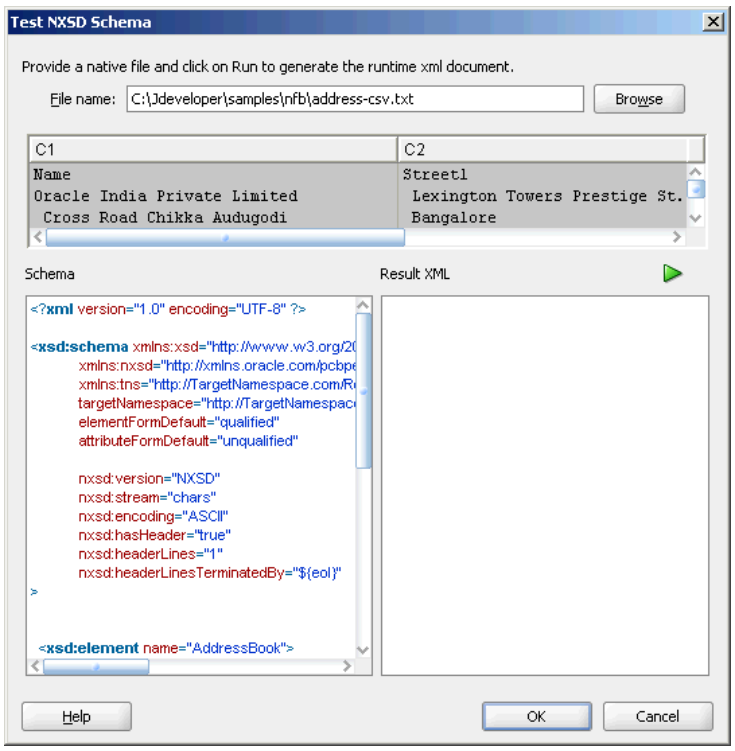
The corresponding native schema definition is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            xmlns:tns="http://TargetNamespace.com/Read"
            targetNamespace="http://TargetNamespace.com/Read"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            nxsd:version="NXSD"
            nxsd:stream="chars"
            nxsd:encoding="ASCII"
            nxsd:hasHeader="true"
            nxsd:headerLines="1"
            nxsd:headerLinesTerminatedBy="${eol}"
>
  <xsd:element name="AddressBook">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Address" minOccurs="1"
              maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Name"
                type="xsd:string"
                  nxsd:style="terminated"
                  nxsd:terminatedBy=","
                  nxsd:quotedBy="&quot;" />
              <xsd:element name="Street1"
          type="xsd:string"
               nxsd:style="terminated"
               nxsd:terminatedBy=","
               nxsd:quotedBy="&quot;" />
               <xsd:element name="Street2"
```

```
                type="xsd:string"
               nxsd:style="terminated"
               nxsd:terminatedBy=","
               nxsd:quotedBy="&quot;" />
              <xsd:element name="City" type="xsd:string"
               nxsd:style="terminated"
               nxsd:terminatedBy=","
               nxsd:quotedBy="&quot;" />
              <xsd:element name="State"
                 type="xsd:string"
               nxsd:style="terminated"
               nxsd:terminatedBy=","
               nxsd:quotedBy="&quot;" />
              <xsd:element name="Country"
                  type="xsd:string"
               nxsd:style="terminated"
               nxsd:terminatedBy="${eol}"
               nxsd:quotedBy="&quot;" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

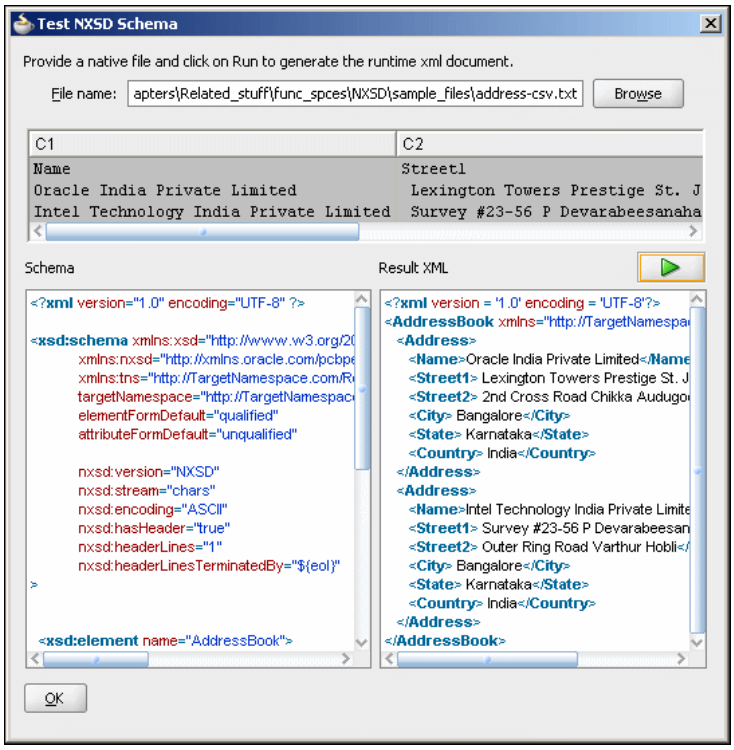
12. Click **Test**. The Test NXSD Schema dialog is displayed, as shown in Figure 6-13.

Figure 6-13 Test NXSD Schema Dialog



13. Click the **Generate XML** icon. The resultant XML is displayed on the Result XML pane of the Test NXSD Schema dialog, as shown in Figure 6-14.

Figure 6-14 Test NXSD Schema Dialog



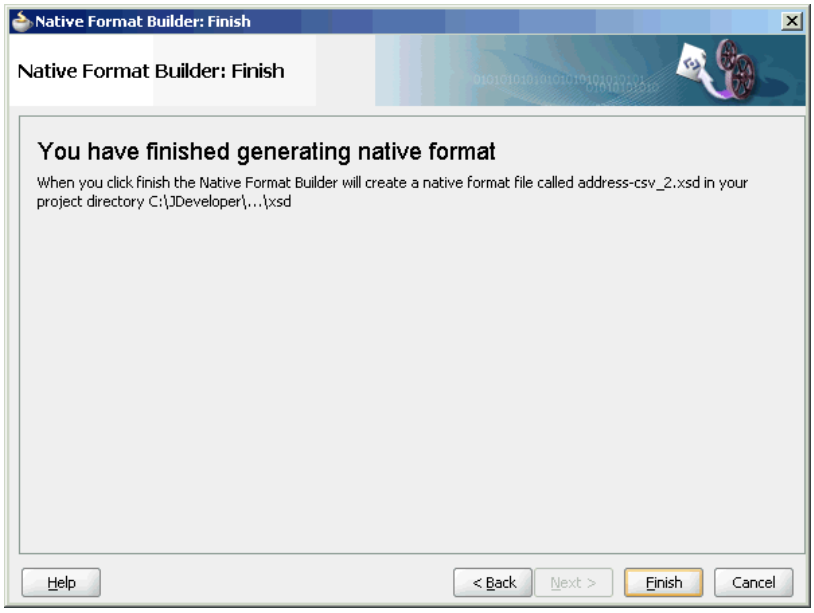
The native data using the corresponding native schema format is translated into the following XML:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<AddressBook xmlns="http://TargetNamespace.com/ReadFile">
   <Address>
      <Name>Oracle India Private Limited</Name>
      <Street1> Lexington Towers Prestige St. John's Woods</Street1>
      <Street2> 2nd Cross Road Chikka Audugodi</Street2>
      <City> Bangalore</City>
      <State> Karnataka</State>
      <Country> India</Country>
   </Address>
   <Address>
      <Name>Intel Technology India Private Limited</Name>
      <Street1> Survey #23-56 P Devarabeesanahalli Village</Street1>
      <Street2> Outer Ring Road Varthur Hobli</Street2>
      <City> Bangalore</City>
      <State> Karnataka</State>
      <Country> India</Country>
   </Address>
</AddressBook>
```

14. Click **OK**. The Generated Native Format Schema File page is displayed, as shown in Figure 6-12.

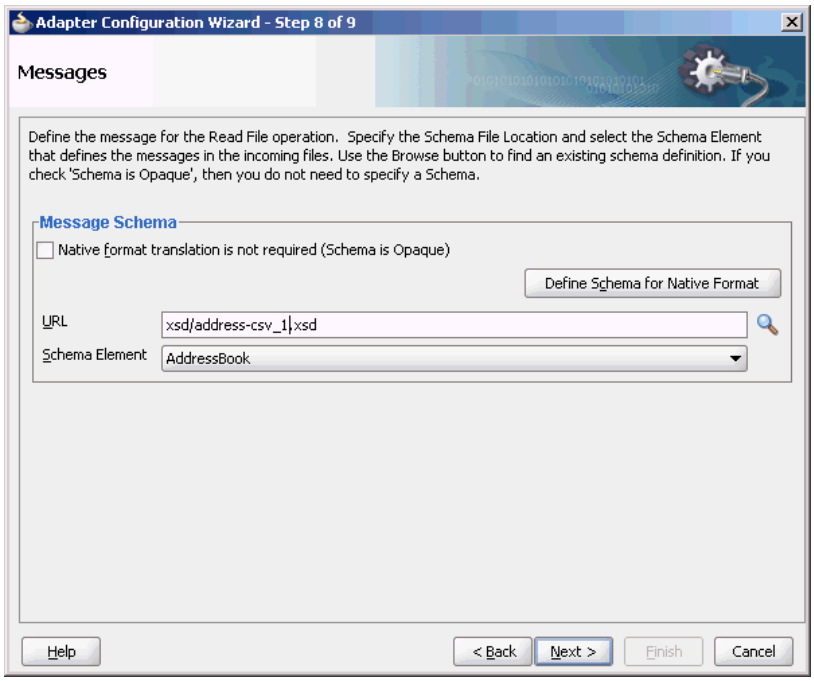
15. Click **Next**. The Native Format Builder Finish page is displayed, as shown in Figure 6-15.

Figure 6-15 Native Format Builder Wizard Finish Page



16. Click **Finish**. The Adapter Configuration Wizard Messages page is displayed, as shown in Figure 6-16, containing the generated NXSD.

Figure 6-16 Adapter Configuration Wizard Messages Page



Defining a Asterisk (*) Separated Value File Structure

The use case defined in the previous example is just one specific case of the *SV class, where the wildcard can be substituted by any character or string. For example, for the native data containing a plus (+) separated value, substitute the wildcard with the plus (+) character.

Use the **Delimited** type option in the Native Format Builder wizard when creating the XML schema for this native file.

Native Data Format to Be Translated

The following native data format is provided:

```
a+b+c+d+e
f+g+h+i+j
```

Native Schema

The corresponding native schema definition is similar to the one in the previous use case except that instead of `nxsd:terminatedBy=","` you now define the terminated by format as `nxsd:terminatedBy="+"`. See Defining Terminated Data for details about the terminated style.

Defining the Schema for a Fixed Length File Structure

In this example, the native data used is the same as in the CSV case, but the data used is of type fixed length and not CSV.

Use the Fixed Length option in the Native Format Builder wizard, to create the XML schema for this native file.

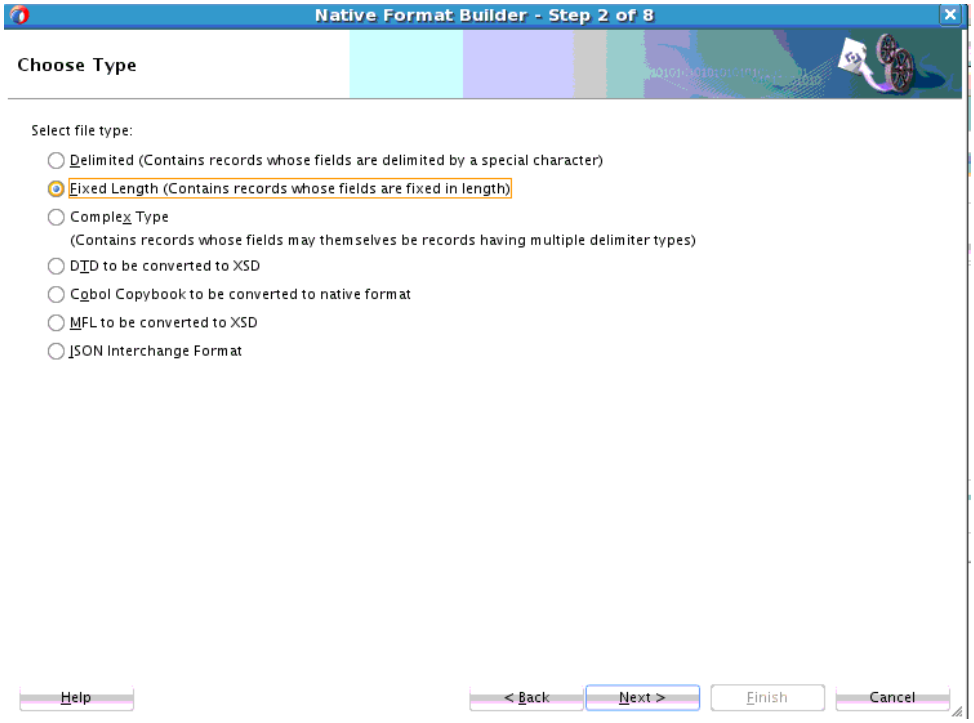
In this use case, the Native Format Builder uses a fixed-length file type called `address` that contains the address details such as name, street, city, state, and country. Every element in this `address` native file has a fixed length. You can generate the corresponding NXSD and also test it. Perform the following steps to run the use case:

1. The data in a sample text file, `address.txt`, is:

```
Name                 Street    City      State     Country
ABC Private Limited  Street1   Bangalore Karnataka India
XYZ Private Limited  Street1   Bangalore Karnataka India
```

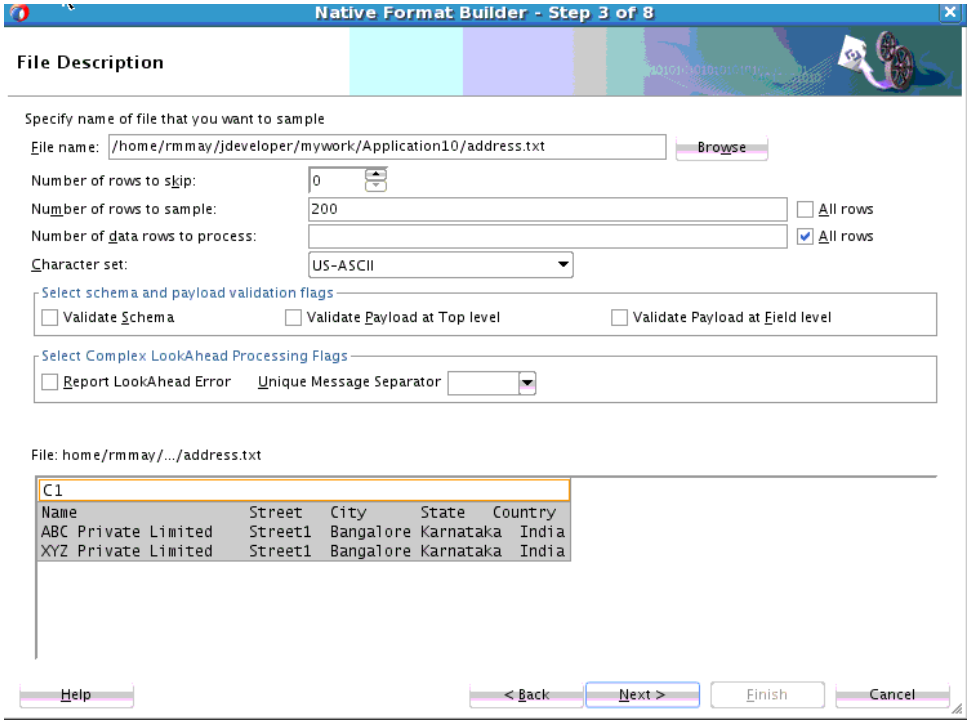
2. Launch the Adapter Configuration Wizard and navigate to the Messages page, as displayed in Figure 6-4, and click **Define Schema For Native Format**. The Native Format Builder Welcome page is displayed, as shown in Figure 6-5.
3. Click **Next**. The Choose Type page is displayed.
4. Select **Fixed Length** as the file type, as shown in Figure 6-17.

Figure 6-17 Native Format Builder Wizard Choose Type Page



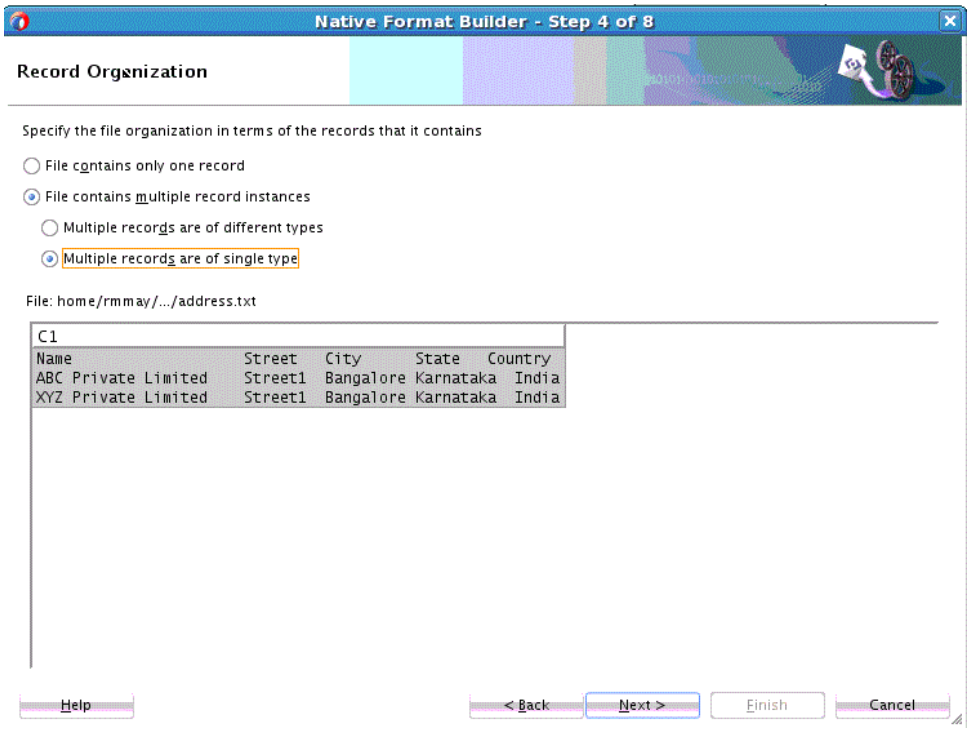
5. Click **Next**. The Native Format Builder File Description page is displayed.
6. Click **Browse** and select the `address.txt` file, as displayed in Figure 6-18.

Figure 6-18 Native Format Builder Wizard File Description Page



7. Click **Next**. The Record Organization page is displayed, as shown in Figure 6-19.

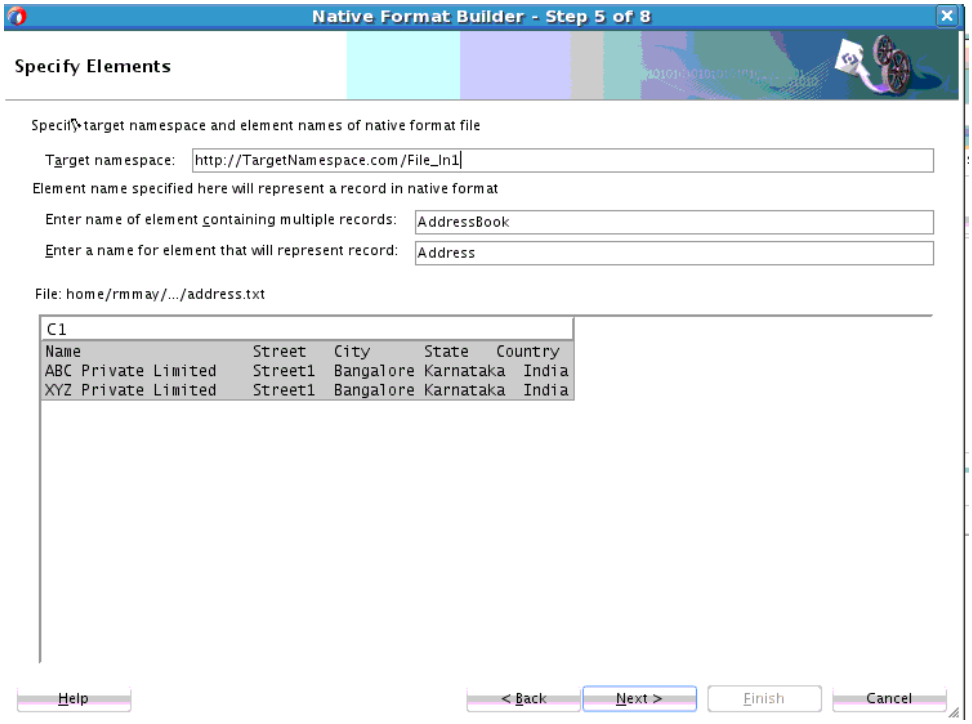
Figure 6-19 Native Format Builder Wizard Record Organization Page



8. Select **Multiple records are of single type**, and click **Next**. The Specify Elements page is displayed.

9. Enter `AddressBook` in the **Enter name of element containing multiple records** field, and enter `Address` in the **Enter a name for element that represents record** field, as shown in Figure 6-20.

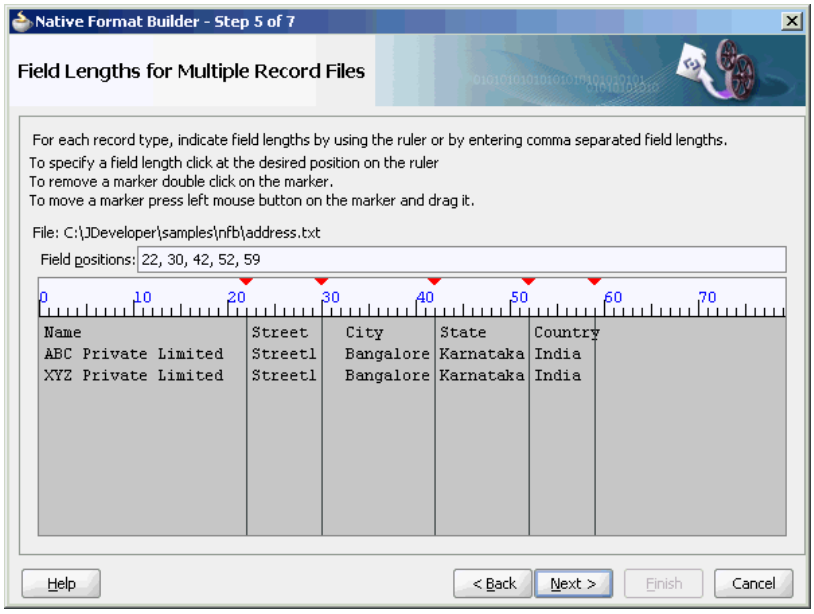
Figure 6-20 Native Format Builder Wizard Specify Elements Page



10. Click **Next**. The Field Lengths for Multiple Record Files page is displayed.

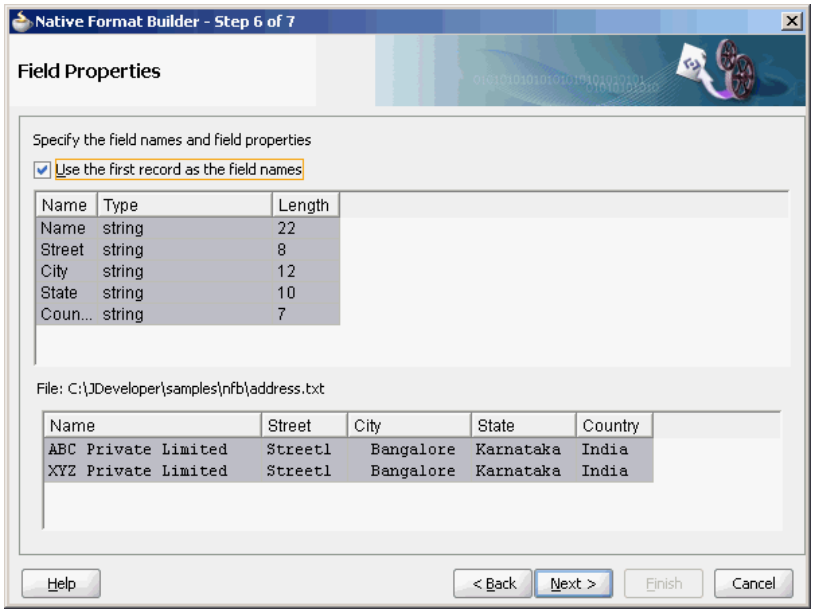
11. Click the ruler at the desired position to mark fields on the sample text area, as shown in Figure 6-21 and click **Next**. The Field Properties page is displayed.

Figure 6-21 Native Format Builder Wizard Field Lengths for Multiple Record Files Page



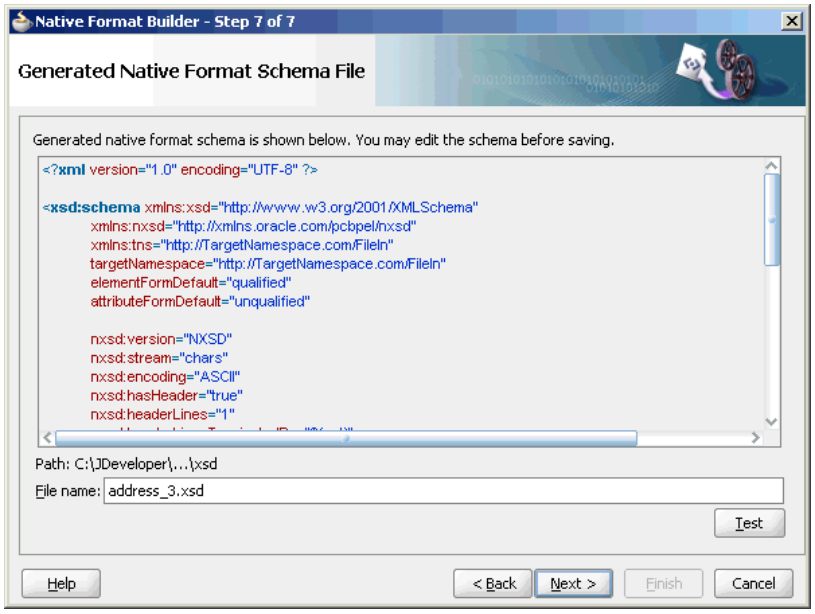
12. Check **Use the first record as the field names**, as shown in Figure 6-22.

Figure 6-22 Native Format Builder Wizard Field Properties Page



13. Click **Next**. The Generated Native Format Schema File page is displayed, as shown in Figure 6-23.

Figure 6-23 Native Format Builder Wizard Native Format Schema File Page



The corresponding native schema definition is similar to the definition of the CSV, file but `style` changes from `nxsd:style="terminated"` to `nxsd:style="fixedLength"` along with the relevant attributes for the fixed-length style. For the fixed-length style, the one mandatory attribute is the length: `nxsd:length`. The value of `nxsd:length` is the actual length of the data to be read.

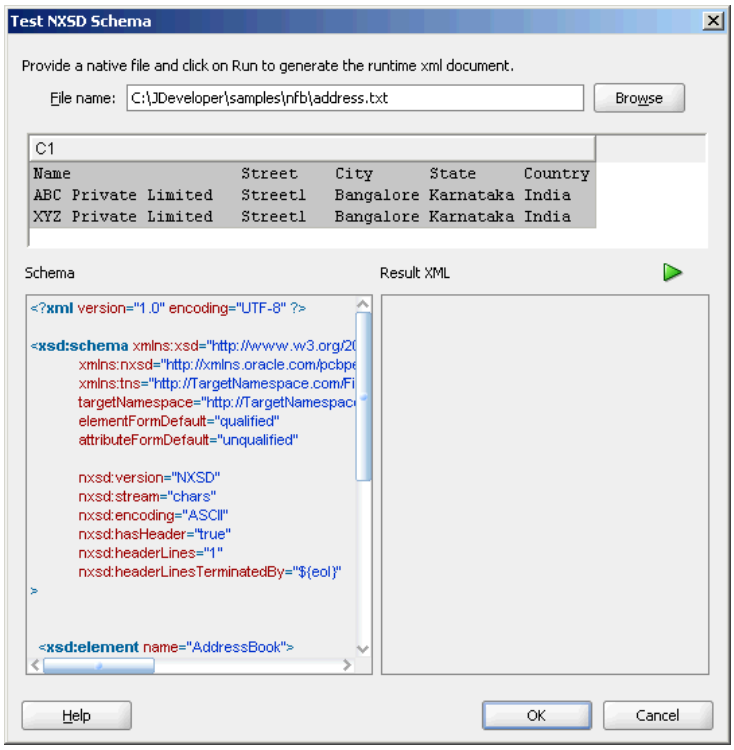
Example - Native Schema Definition Example with Fixed Length Style

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            xmlns:tns="http://TargetNamespace.com/FileIn_1"
            targetNamespace="http://TargetNamespace.com/FileIn_1"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"

            nxsd:version="NXSD"
            nxsd:stream="chars"
            nxsd:encoding="ASCII"
            nxsd:hasHeader="true"
            nxsd:headerLines="1"
            nxsd:headerLinesTerminatedBy="${eol}"

  <xsd:element name="AddressBook">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Address" minOccurs="1"
           maxOccurs="unbounded"
           nxsd:style="array"
           nxsd:cellSeparatedBy="${eol}">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Name"
               type="xsd:string"
               nxsd:style="fixedLength"
               nxsd:length="22" />
              <xsd:element name="Street"
                type="xsd:string"
                nxsd:style="fixedLength"
                nxsd:length="8" />
              <xsd:element name="City"
               type="xsd:string"
               nxsd:style="fixedLength" nxsd:length="12" />
              <xsd:element name="State"
                type="xsd:string" nxsd:style="fixedLength"
                           nxsd:length="10" />
              <xsd:element name="Country" type="xsd:string"
nxsd:style="fixedLength" nxsd:length="7" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

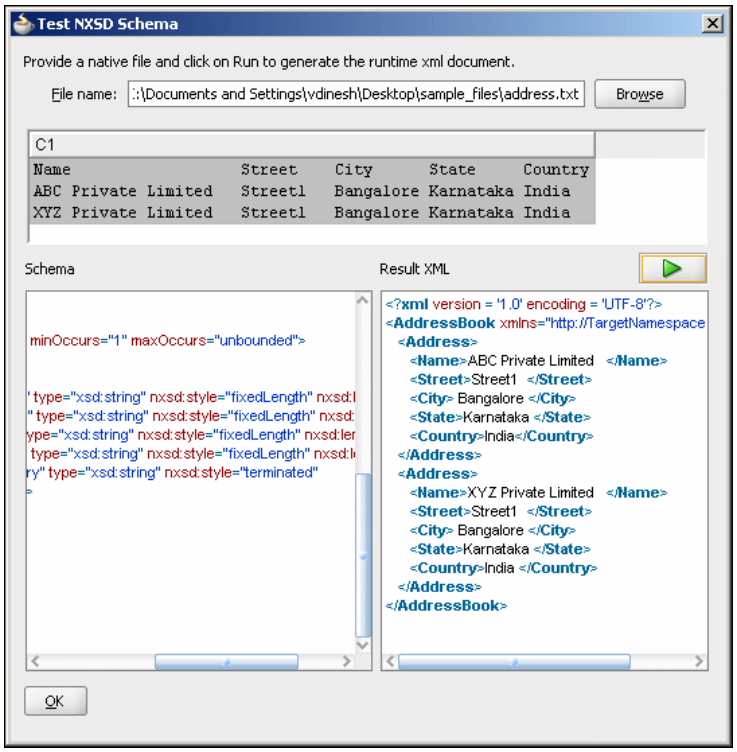
14. Click **Test**. The Test NXSD Schema dialog is displayed, as shown in Figure 6-24.

Figure 6-24 Test NXSD Schema Dialog



15. Click the **Generate XML** icon. The resultant XML is displayed on the Result XML pane of the Test NXSD Schema dialog, as shown in Figure 6-25.

Figure 6-25 Test NXSD Schema Dialog



The native data using the corresponding native schema format is translated into XML as shown in the following example.

Example - The Native Data Using the Native Schema FormatTranslated XML

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<AddressBook xmlns="http://TargetNamespace.com/Read">
   <Address>
      <Name>ABC Private Limited</Name>
      <Street>Street1  </Street>
      <City> Bangalore </City>
      <State>Karnataka </State>
      <Country>India</Country>
   </Address>
   <Address>
      <Name>XYZ Private Limited</Name>
      <Street>Street1  </Street>
      <City> Bangalore </City>
      <State>Karnataka </State>
      <Country>India </Country>
   </Address>
</AddressBook>
```

16. Click **OK**. The Generated Native Format File page is displayed, as shown in Figure 6-23.

17. Click **Next**. The Native Format Builder Finish page is displayed, as shown in Figure 6-15.

18. Click **Finish**. The Adapter Configuration Wizard Messages page is displayed, as shown in Figure 6-16, that contains the generated NXSD.

Defining the Schema for a Complex File Structure

The file structure of an invoice is more complex than the structure of CSV, *SV, and fixed-length files discussed in the preceding use cases. An invoice usually contains buyer information, seller information, and line items. Each of these elements, in turn, can be of complex type. For example, the buyer element can be defined as a partner-type, where partner-type consists of three elements - id, name, and address.

Use the Complex Type option in the Native Format Builder wizard when creating the XML schema for this native file.

In this use case, the Native Format Builder uses `invoice.txt`, a complex file type called invoice, which contains multiple records such as buyer, seller, and items. Also, using this use case, you can generate the NXSD and test it. Perform the following steps to run this use case:

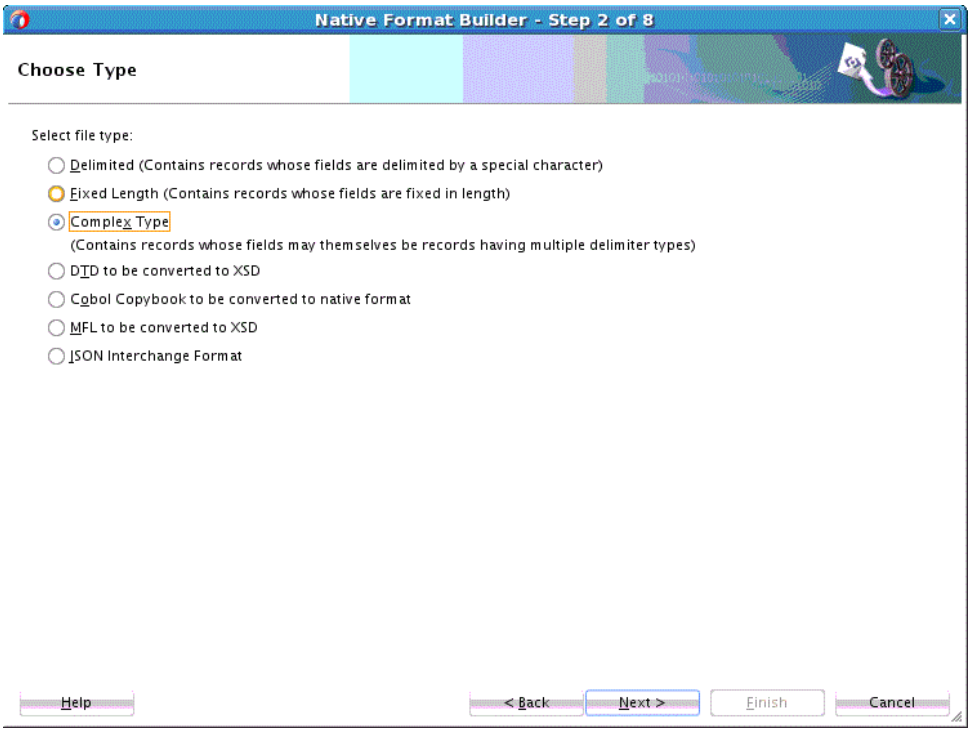
1. The data in a sample text file, `invoice.txt`, is:

```
6335722^Company OnêFirst Street   999       San Jose       95129USCA650-801-6250
 ^OraclêBridge Parkway 1600      Redwood Shores 94065USCA650-506-7000
001|BPEL Process Manager Enterprise Edition|20000,2,+40000+
002|BPEL Process Manager Standard Edition|10000,5,+50000+
003|BPEL Process Manager Developer Edition|1000,20,+20000+#110000
```

2. Launch the Adapter Configuration Wizard and navigate to the Messages page, as displayed in Figure 6-4, and click **Define Schema For Native Format**. The **Native Format Builder Welcome** page is displayed, as shown in Figure 6-5.

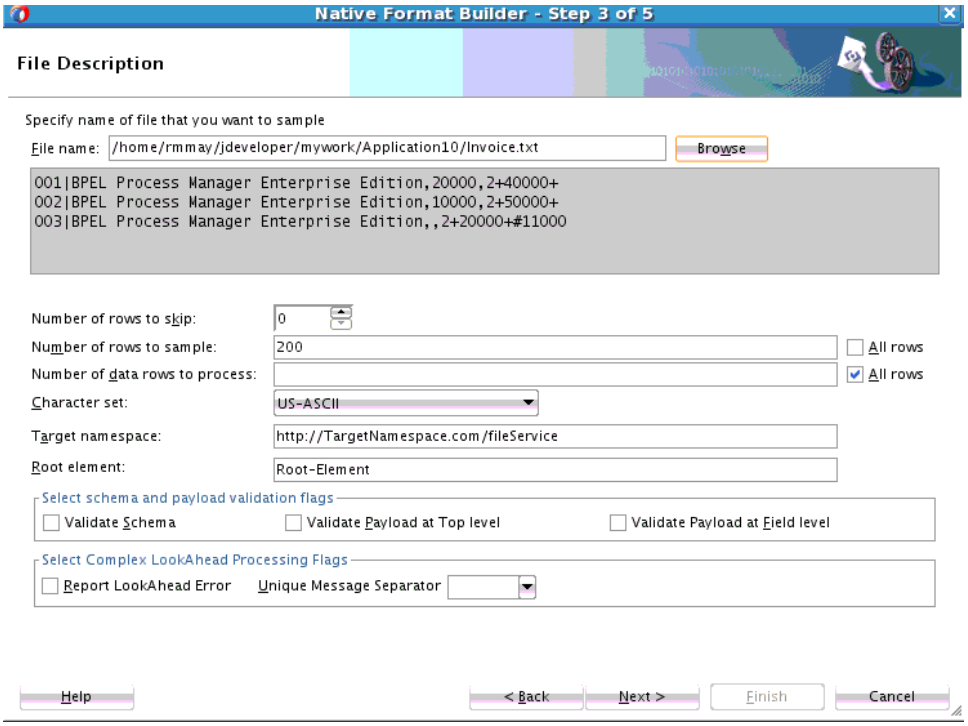
3. Click **Next**. The **Choose Type** page is displayed, as shown in Figure 6-26.

Figure 6-26 Native Format Builder Wizard Choose Type Page



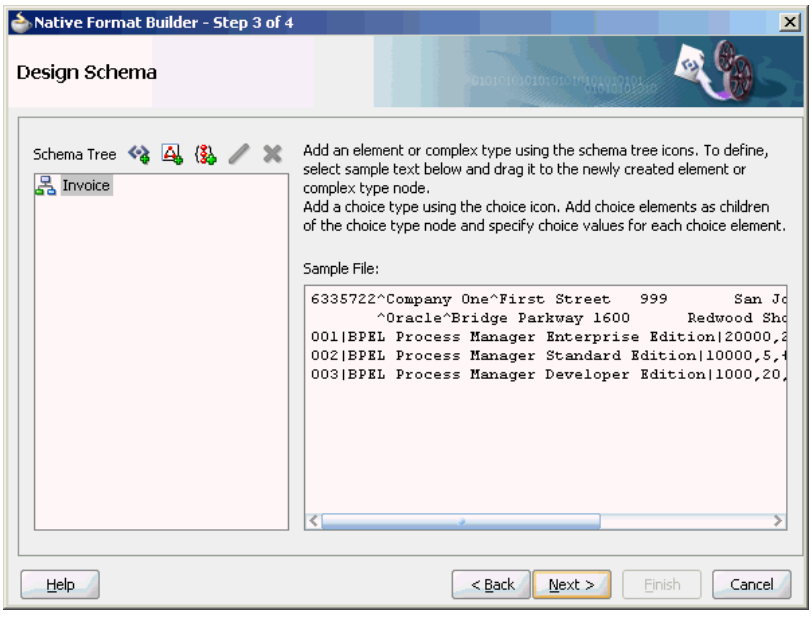
4. Select **Complex Type (Contains records whose fields may themselves be records having multiple delimiter types)**.
5. Click **Next**. The **Native Format Builder File Description** page is displayed.
6. Click **Browse** and select the `invoice.txt` file, and enter `Invoice` in the Root Element field, as displayed in Figure 6-27.

Figure 6-27 Native Format Builder Wizard File Description Page



7. Click **Next**. The **Native Format Builder Design Schema** is displayed, as shown in Figure 6-28.

Figure 6-28 Native Format Builder Wizard Design Schema Page



Create the partner-type Complex Type

The schema structure that you can build using the `invoice.txt` sample is as follows:

Invoice

Buyer => partner-type

Seller => partner-type

Items => item-type

Invoice-total => double

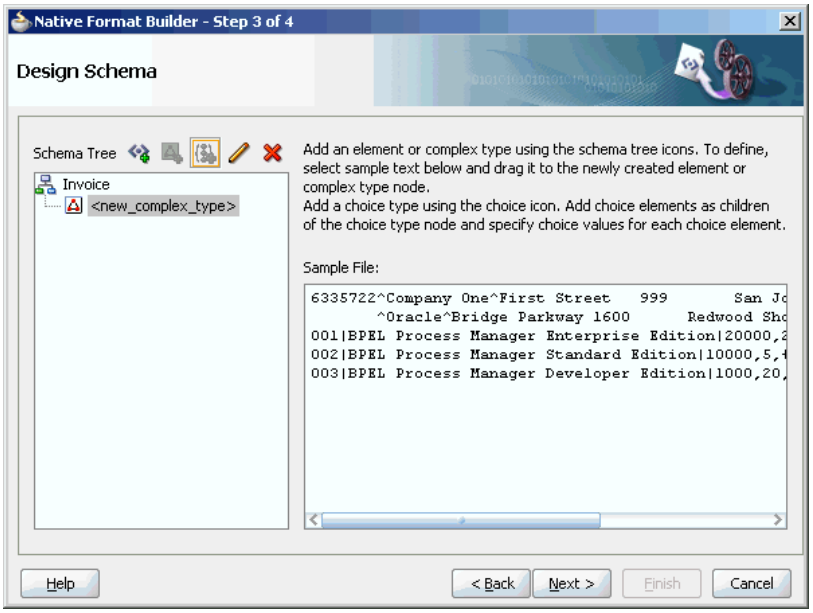
The first line in the native data consists of buyer details, followed by seller details, followed by line items, and finally the total for the line items. Both buyer and seller elements have the same complex structure, as follows:

- The first seven characters are the UID
- This is followed by the buyer/seller name surrounded by " ^".
- This is followed by the address until the end of the line.

To create this in the Native Format Builder:

1. Click the **Add Complex Type** icon. A Complex Type, `<new_complex_type>` is created in the Schema Tree under `Invoice`, as shown in Figure 6-29.

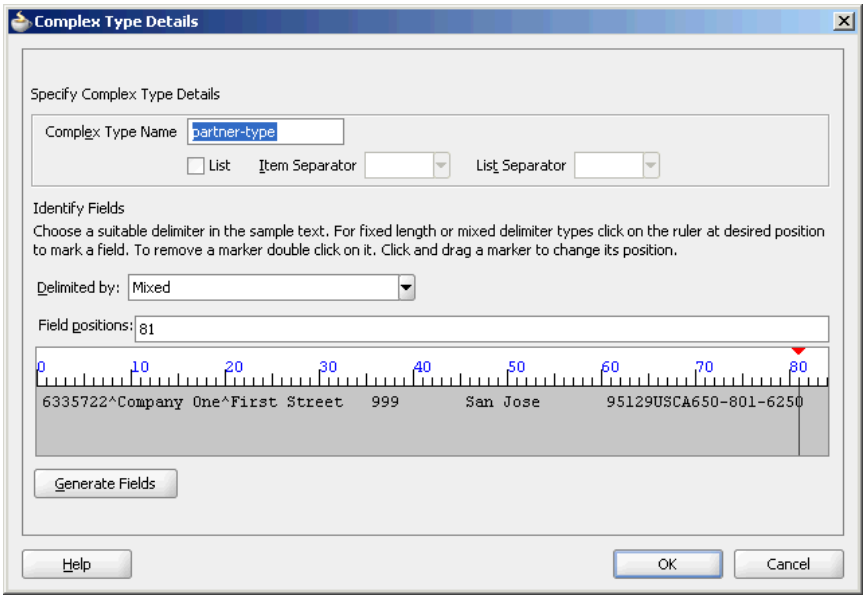
Figure 6-29 Native Format Builder Wizard Design Schema Page



2. Select the first row of the sample text from the right-hand pane of the Sample File section, and drag and drop it on the **<new_complex_type>** node. The **Complex Type Details** dialog is displayed.

3. Enter `partner-type` in the **Complex Type Name** field, as shown in Figure 6-30.

Figure 6-30 Native Format Builder Wizard Design Schema Page - Complex Type Details Dialog



4. Click the ruler at the desired position to mark fields on the sample text area, and then click the **Generate Fields** button. The system interprets the style of data for the defined fields.

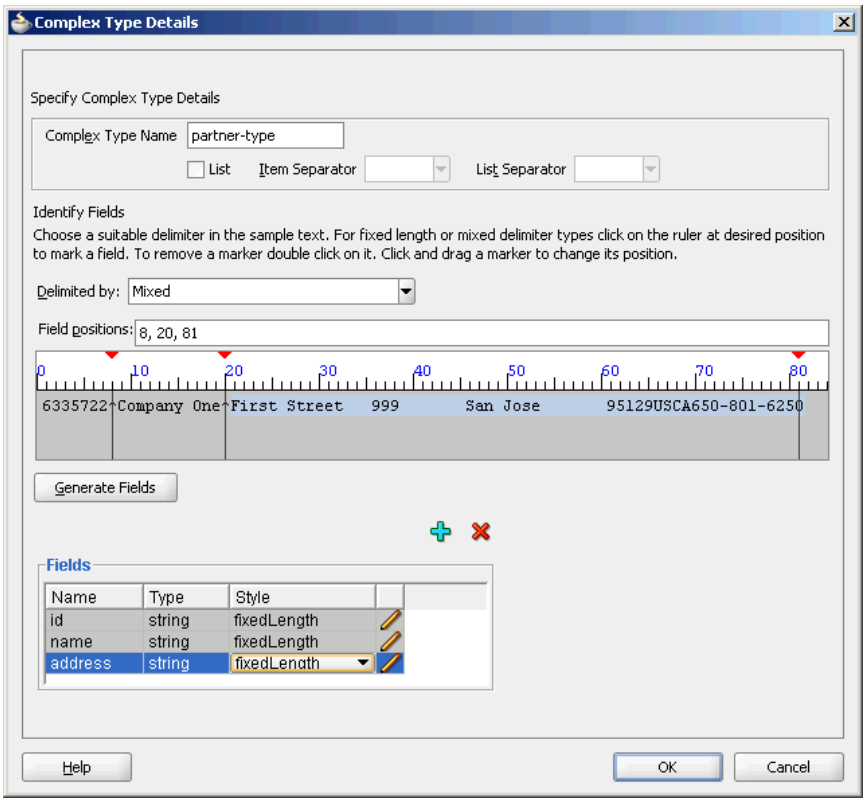
Note:

For the Fixed Length or Mixed Delimiter type options, a ruler-based text area is displayed. You have to use the rulers to identify fields within the sample text. In case of delimited data, select or enter the appropriate delimiter in the **Delimited By** field.

---

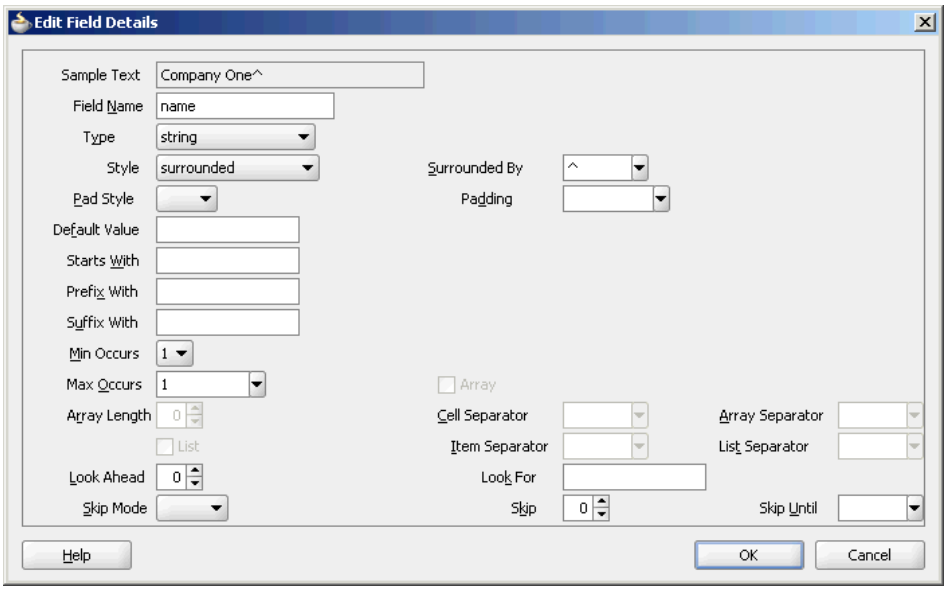
5. Enter the **id**, **name**, and **address** in the Name field, as shown in Figure 6-31.

Figure 6-31 Complex Type Details Dialog



6. Click the pencil icon adjacent to each field to display the corresponding **Edit Field Details** dialog that enables you to edit the field properties. For example, click the pencil icon adjacent to the **Name** field. The **Edit Field Details** dialog is displayed, as shown in Figure 6-32.

Figure 6-32 Edit Field Details Dialog



7. Edit the following field properties, as shown in Figure 6-32.

   - **Type**: The data type of the sample text. Select **String** from the Type list.

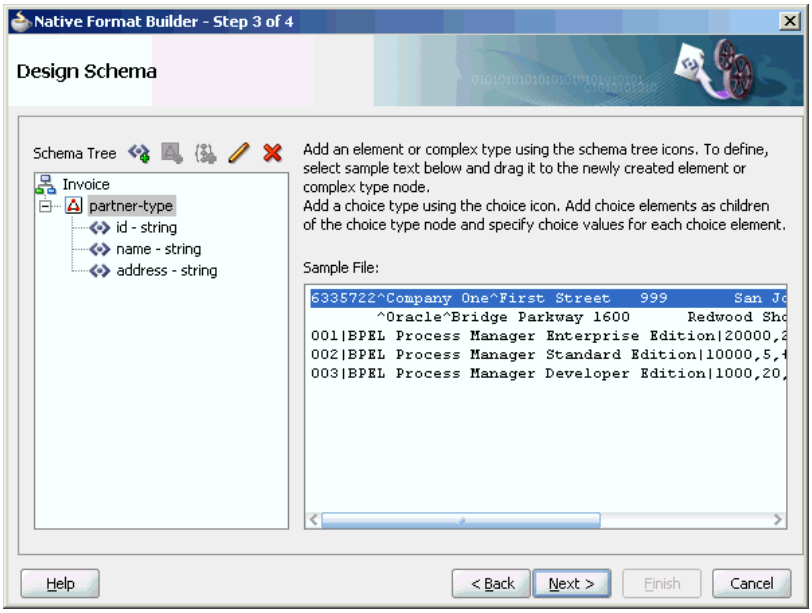
- **Style**: Represents the style of the complex type element. You can select any of the following four options:
  - fixed length
  - surrounded
  - terminated
  - left/right surrounded

  In this example, select **surrounded**.
- **Surrounded By**: This option is displayed when you select surrounded in the Style option. In this example, enter caret (^) in the Surrounded By field.

The field properties displayed on this panel correspond to the NXSD attributes used in the schema.

8. Click **OK**. The **Complex Type Details** dialog is displayed with the field properties that you selected.
9. Verify or edit the field properties for `id` and `address` **Name** fields.
10. Click **OK** in the **Complex Type Details** dialog. The **Native Format Builder Design Schema** page is displayed, as shown in Figure 6-33.

Figure 6-33 Native Format Builder Wizard Design Schema Page - partner-type Complex Type



Create an address-type Complex Type

The address element can be further defined as another complex-type that contains a fixed-length street, city, and additional fields.

To do so:

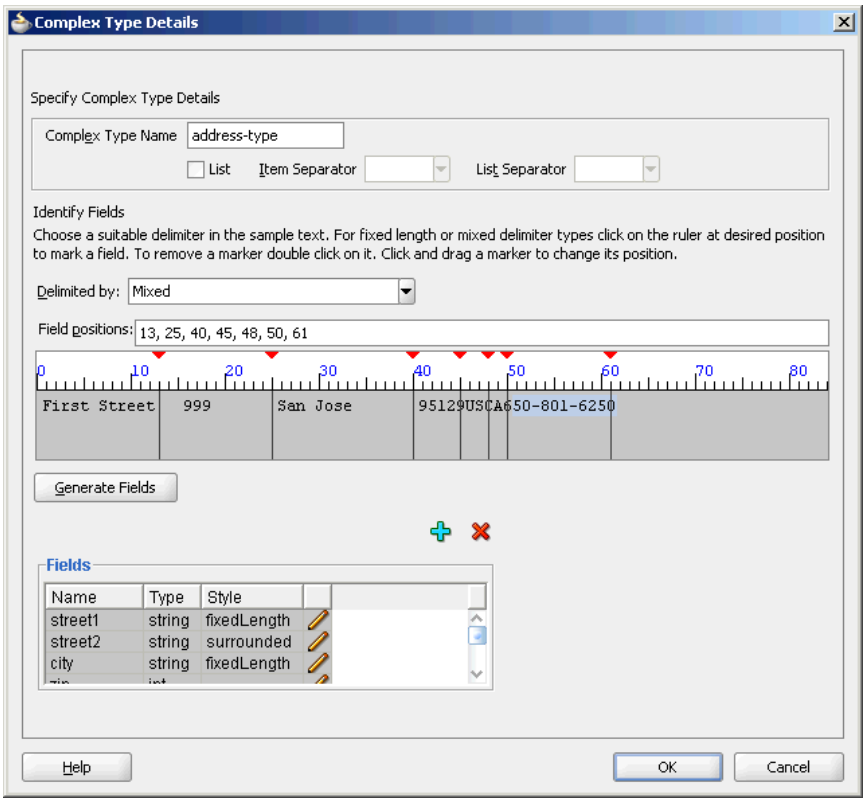
1. Create another <noncompliant> node in the Schema Tree. See Step 1 in Create the partner-type Complex Type.

2. Drag and drop the address part in the first row of the sample text to the Complex Type, `<noncompliant>`. The **Complex Type Details** dialog is displayed.

3. Enter **address-type** in the **Complex Type Name** field.

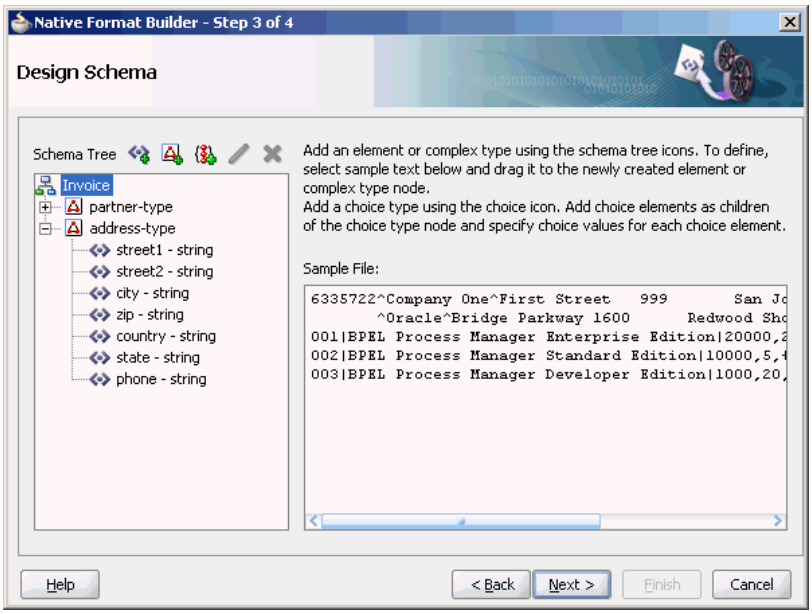
4. Click the ruler to mark fields on the sample text area, and then click the **Generate Fields** button. Now, enter **street1**, **street2**, **city**, **zip**, **country**, **state**, and **phone** in the **Name** field, as shown in Figure 6-34.

Figure 6-34 Native Format Builder Wizard Design Schema Page - Complex Type Details Dialog



5. Click **OK**. The **Native Format Builder Design Schema** page is displayed, as shown in Figure 6-35.

Figure 6-35 Native Format Builder Wizard Design Schema Page



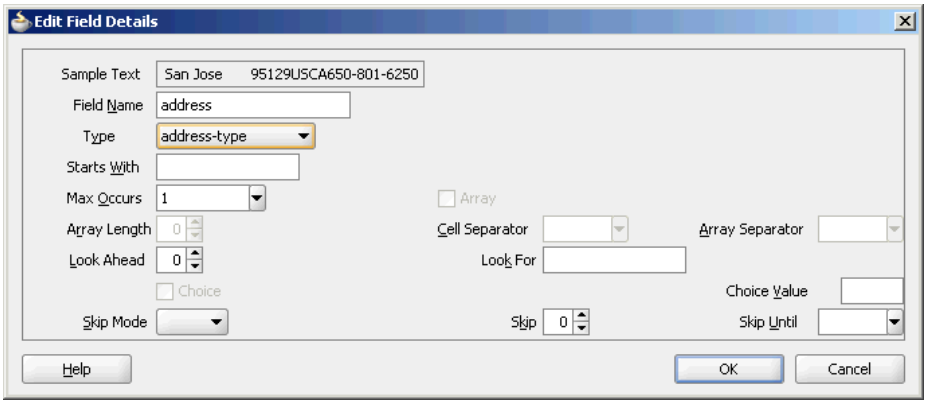
Assign the address-type Complex Type to the address field of partner-type Complex Type

You must assign the address-type complex type to the address field of the partner-type complex type. You can assign a complex type to an element by using one of the following methods:

1.
   - Drag and drop the address-type node on the address field node of the partner-type complex type. This instantly assigns address-type to the address field element, or
   - Select the address field node of the partner-type complex type and then click the pencil icon.

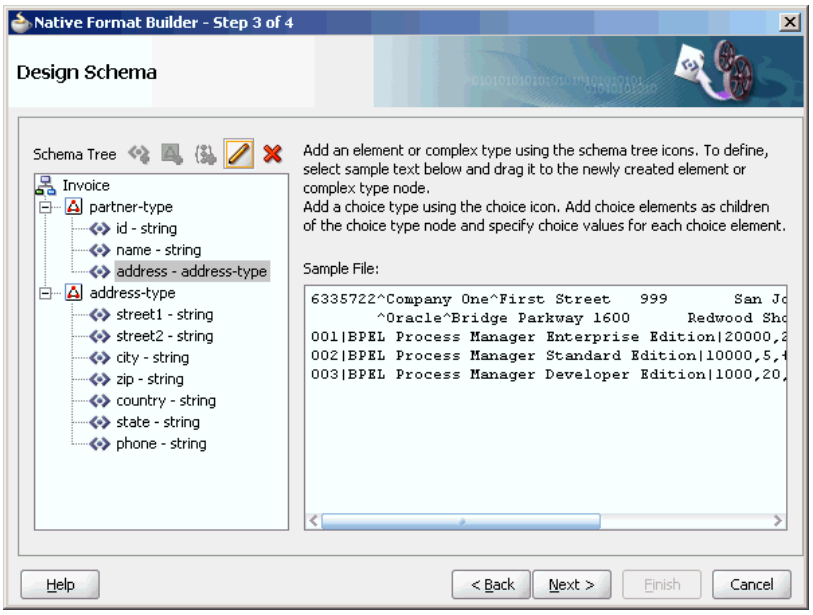
     The **Edit Field Details** dialog is displayed, as shown in Figure 6-36.

Figure 6-36 Edit Field Details Dialog



2. Select the **address-type** option in the Type list, and click **OK**. The address-type option is assigned to the address field element in the **Native Format Builder Design Schema** page, as shown in Figure 6-37.

Figure 6-37 Native Format Builder Wizard Design Schema Page

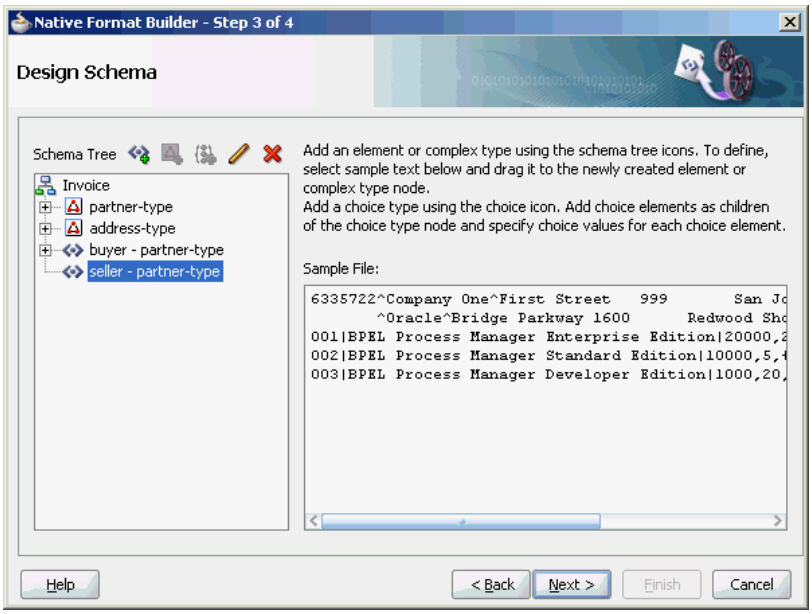


Create 'buyer' and 'seller' Global Elements

1. Select **Invoice**, and click the **Add Element** icon. An element, `<new_element>`, is created in the Schema Tree under the root element, **Invoice**.
2. Rename it to **buyer**.
3. Again, select **Invoice**, and click the **Add Element** icon. An element, `<new_element>`, is created in the Schema Tree under **Invoice**.
4. Rename it to **seller**.

   Drag and drop the partner-type node on each of the buyer and seller nodes, to assign the partner-type complex type to these nodes. The Schema Tree appears, as shown in Figure 6-38.

Figure 6-38 Native Format Builder Wizard Design Schema Page

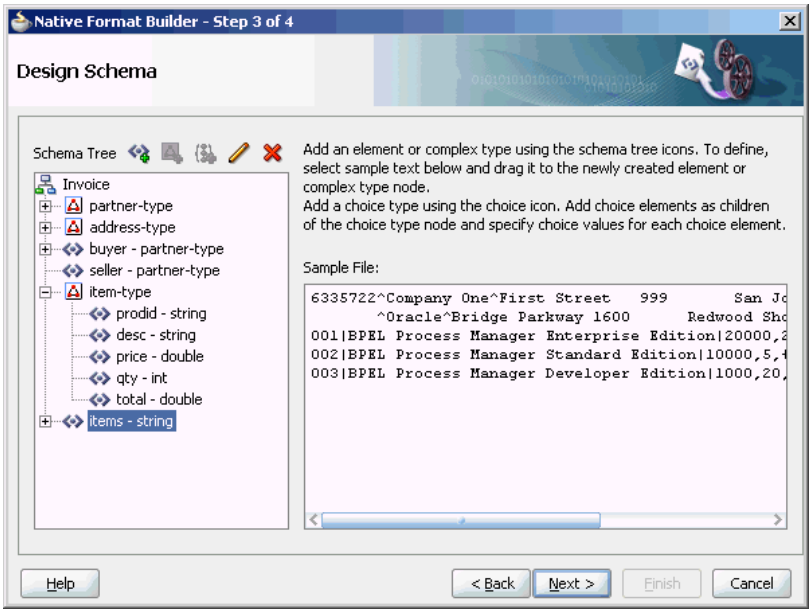


Create item-type Complex Type, and items and invoice-total Element Nodes

The items element can be considered an array of item-types. The last line item in the native file ends with the number sign (#), followed by the line-item total.

1. Select **Invoice**, and click the **Add Element** icon. An element, `<new_element>`, is created in the Schema Tree under **Invoice**.
2. Rename it to **items**.
3. Create the item-type complex type and define the field properties, as shown in Figure 6-39.

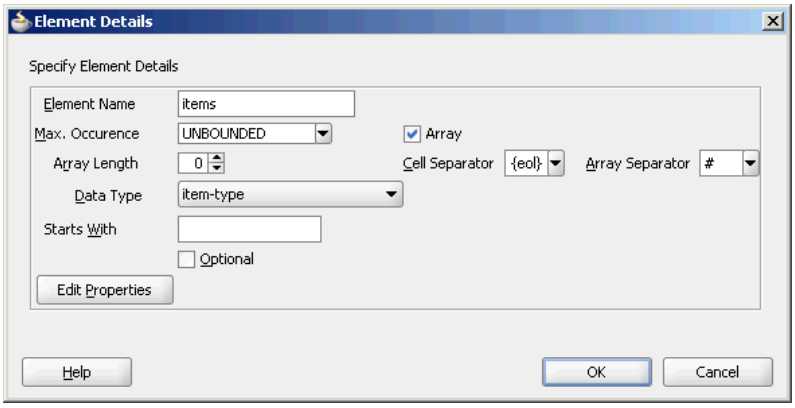
Figure 6-39 Native Format Builder Wizard Design Schema Page



4. Drag and drop **item-type** complex type to the **items** element to assign item-type to this element.

5. Select **items - item-type** and click the pencil icon. The Element Details dialog is displayed.

Figure 6-40 Element Details Dialog



6. Set the following properties in the **Element Details** dialog, as shown in Figure 6-40:

   a. Set Max. Occurrence - UNBOUNDED

   b. Select **Array**. The Cell Separator and Array Separator are enabled.

   c. Set Cell Separator - ${eol}

   d. Set Array Separator - #

   Note:

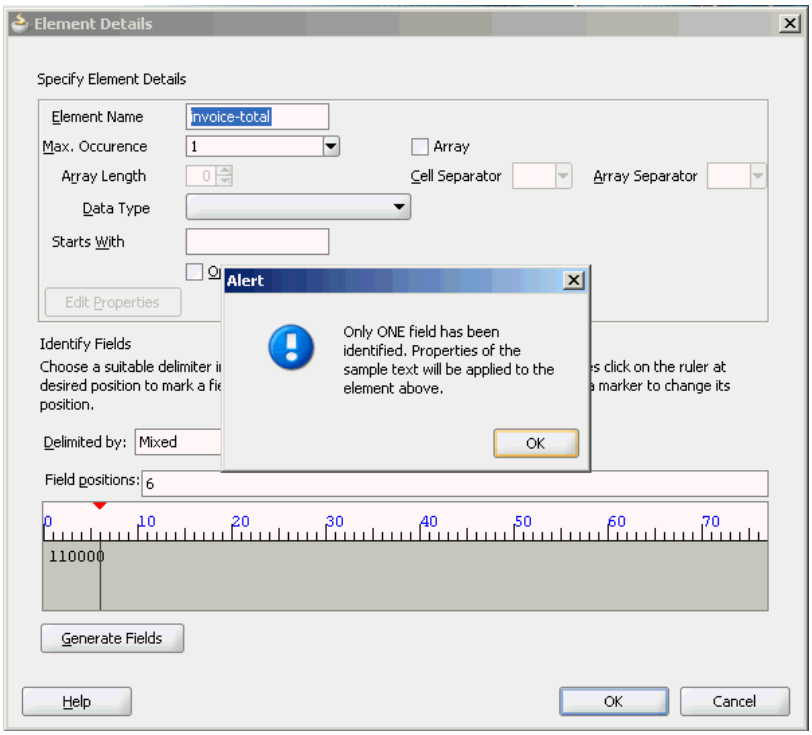
   The element `items` is defined as an array of `item-type`.

7. Click **OK**.

8. Create the invoice-total element, and drag and drop the sample text (110000) on the <new_element> node. The **Element Details** dialog is displayed.

9. Enter invoice-total in the **Element Name** field, and click **Generate Fields**. The Alert message is displayed, as shown in Figure 6-41.

Figure 6-41 Element Details Dialog - Alert Message

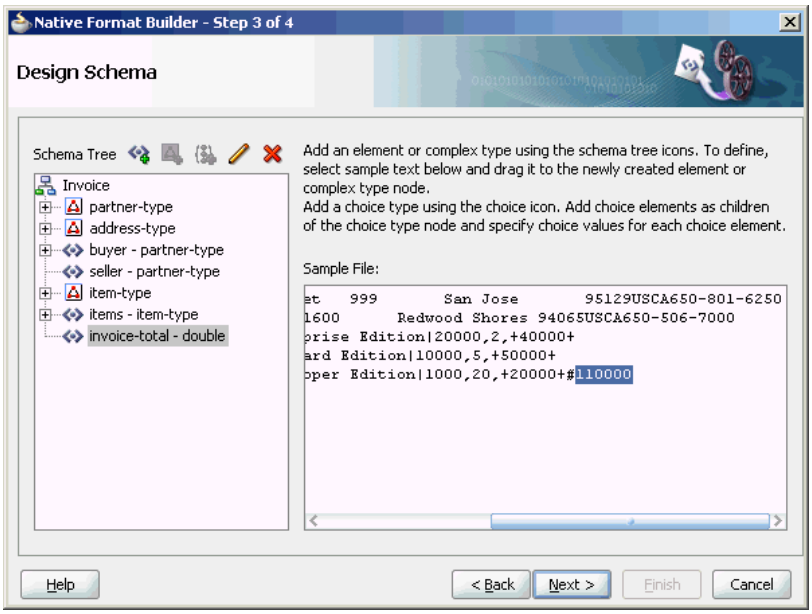


If a single field is identified in the sampled data for a global element, then the properties of this data are applied to the global element itself.

10. Click **OK** in the Alert message. The **Element Details** dialog is displayed.

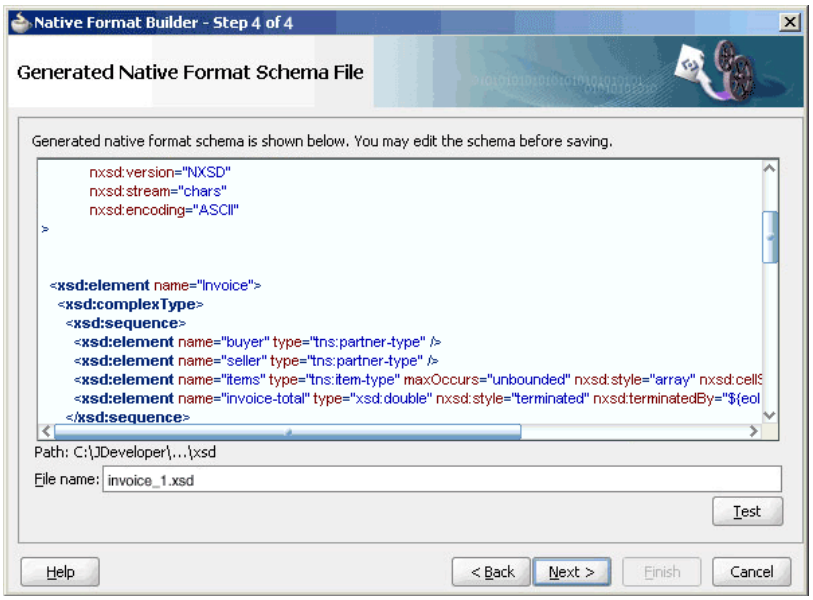
11. Select **double** in the Data Type list, and click **OK**. The **Native Format Builder Design Schema** page is displayed, as shown in Figure 6-42.

Figure 6-42 Native Format Builder Wizard Design Schema Page - Complete Schema Tree



12. Click **Next**. The **Generated Native Format Schema File** page is displayed, as shown in Figure 6-43, which displays the native format file.

Figure 6-43 Native Format Builder Wizard Generated Native Format File Page



The native schema definition corresponding to the preceding native data can be defined as follows:

Example - Native Schema Definition

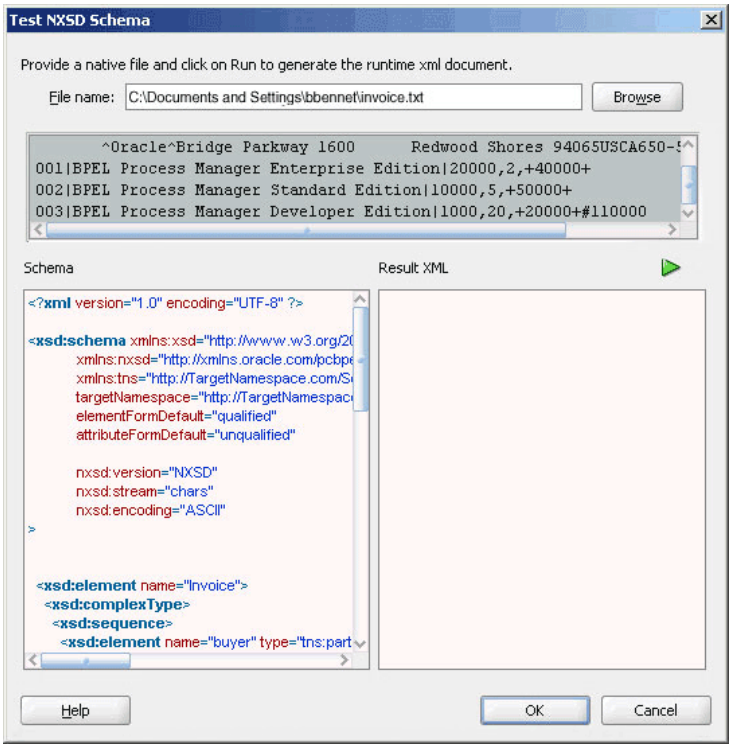
```
<schema attributeFormDefault="qualified"
        elementFormDefault="qualified"
               targetNamespace=
               "http://xmlns.oracle.com/ias/pcbpel
                /fatransschema/demo"
               xmlns:tns="http://xmlns.oracle.com
                             /ias/pcbpel/
                         fatransschema/demo"
               xmlns="http://www.w3.org/2001/XMLSchema"
               xmlns:nxsd="http://xmlns.oracle.com/pcbpel/
                                          nxsd"
            nxsd:version="NXSD" nxsd:stream="chars">

    <element name="invoice" type="tns:invoiceType" />

    <complexType name="invoiceType">
       <sequence>
          <element name="purchaser"
                        type="tns:partnerType" />
          <element name="seller"
                        type="tns:partnerType" />
          <element name="line-item"
                       type="tns:line-itemType"
              maxOccurs="unbounded" nxsd:style="array"
                nxsd:cellSeparatedBy="${eol}"
                nxsd:arrayTerminatedBy="#"/>
          <element name="total" type="double"
                nxsd:style="terminated"
                nxsd:terminatedBy="${eol}"/>
       </sequence>
    </complexType>
```

```
<complexType name="partnerType">
   <sequence>
      <element name="uid" type="string"
         nxsd:style="fixedLength"
         nxsd:length="7" nxsd:padStyle="tail"
         nxsd:paddedBy=" "/>
      <element name="name" type="string"
         nxsd:style="surrounded"
         nxsd:surroundedBy="^"/>
      <element name="address"
                type="tns:addressType" />
   </sequence>
</complexType>


<complexType name="addressType">
   <sequence>
      <element name="street1"
                       type="string"
         nxsd:style="fixedLength"
         nxsd:length="15"
                 nxsd:padStyle="tail"
         nxsd:paddedBy=" "/>
      <element name="street2" type="string"
         nxsd:style="fixedLength"
         nxsd:length="10"
                  nxsd:padStyle="tail"
         nxsd:paddedBy=" "/>
      <element name="city" type="string"
         nxsd:style="fixedLength"
         nxsd:length="15" nxsd:padStyle="tail"
         nxsd:paddedBy=" "/>
      <element name="postal-code" type="string"
         nxsd:style="fixedLength"
         nxsd:length="5"
         nxsd:padStyle="none"/>
      <element name="country" type="string"
         nxsd:style="fixedLength"
         nxsd:length="2"
                   nxsd:padStyle="none"/>
      <element name="state" type="string"
         nxsd:style="fixedLength"
         nxsd:length="2" nxsd:padStyle="none"/>
      <element name="phone" type="string"
         nxsd:style="terminated"
         nxsd:terminatedBy="${eol}"/>
   </sequence>
</complexType>

<complexType name="line-itemType">
   <sequence>
      <element name="uid" type="string"
         nxsd:style="fixedLength"
         nxsd:length="3" nxsd:padStyle="none"/>
      <element name="description" type="string"
         nxsd:style="surrounded"
         nxsd:surroundedBy="|"/>
      <element name="price" type="double"
         nxsd:style="terminated"
           nxsd:terminatedBy=","/>
```

```
            <element name="quantity" type="integer"
                 nxsd:style="terminated"
                 nxsd:terminatedBy=","/>
            <element name="line-total" type="double"
                 nxsd:style="surrounded"
                 nxsd:surroundedBy="+"/>
        </sequence>
    </complexType>

</schema>
```

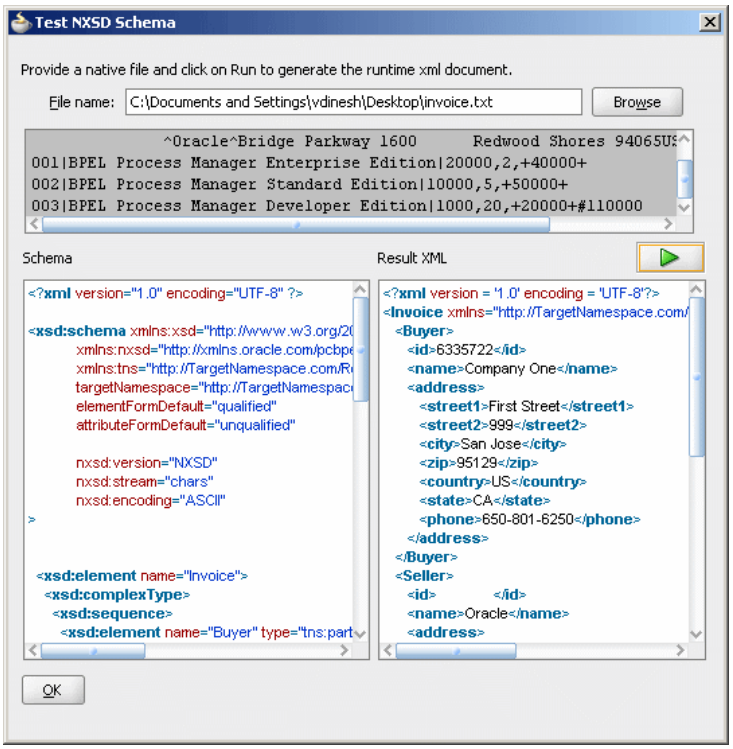
13. Click **Test**. The **Test NXSD Schema** dialog is displayed, as shown in Figure 6-44.

Figure 6-44 Test NXSD Schema Dialog



14. Click the **Generate XML** icon. The Result XML is displayed on the right pane of the **Test NXSD Schema** dialog, as shown in Figure 6-45.

Figure 6-45 Test NXSD Schema Dialog - Result XML



The translated XML is shown in the following example.

Example - Translated XML

```
<invoice xmlns="http://xmlns.oracle.com/pcbpel/
                demoSchema/invoice-nxsd">
  <purchaser>
    <uid>6335722</uid>
    <name>Company One</name>
    <address>
      <street1>First Street</street1>
      <street2>999</street2>
      <city>San Jose</city>
      <postal-code>95129</postal-code>
      <country>US</country>
      <state>CA</state>
      <phone>650-801-6250</phone>
    </address>
  </purchaser>
  <seller>
    <uid/>
    <name>Oracle</name>
    <address>
      <street1>Bridge Parkway</street1>
      <street2>1600</street2>
      <city>Redwood Shores</city>
      <postal-code>94065</postal-code>
      <country>US</country>
      <state>CA</state>
      <phone>650-506-7000</phone>
    </address>
  </seller>
  <line-item>
    <uid>001</uid>
```

```
    <description>BPEL Process Manager Enterprise
                Edition</description>
    <price>20000</price>
    <quantity>2</quantity>
    <line-total>40000</line-total>
  </line-item>
  <line-item>
    <uid>002</uid>
    <description>BPEL Process Manager Standard
                 Edition</description>
    <price>10000</price>
    <quantity>5</quantity>
    <line-total>50000</line-total>
  </line-item>
  <line-item>
    <uid>003</uid>
    <description>BPEL Process Manager Developer
                 Edition</description>
    <price>1000</price>
    <quantity>20</quantity>
    <line-total>20000</line-total>
  </line-item>
  <total>110000</total>
</invoice>
```

15. Click **OK**. The Generated Native Format File page is displayed, as shown in Figure 6-43.
16. Click **Next**. The Native Format Builder Finish page is displayed, as shown in Figure 6-15.
17. Click **Finish**. The Adapter Configuration Wizard Messages page is displayed, containing the generated NXSD, as shown in Figure 6-16.

Removing or Adding Namespaces to XML with No Namespace

When the native data is XML and that XML has no namespace, you can use the Native Format Translator to add a namespace to an inbound XML document and remove the namespace from an outbound XML document.

The XML has no namespace when either of the following is true:

- The XML has a corresponding XML schema, and there is no target namespace specified in that XML schema.
- The XML has a corresponding DTD, which was converted to the XML schema.

In both cases, you must create a wrapper schema with `targetNamespace` specified, and the wrapper schema must include the actual schema. In addition, the wrapper schema must also have the `nxsd:version` attribute set to DTD. For example:

```
--wrapper.xsd
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="myNamespace"
   xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
   nxsd:version="DTD">
   <include schemaLocation="actual.xsd"/>
</schema>
```

Note:

Ensure that `elementFormDefault="qualified"` is specified in the actual schema.

Using this `wrapper.xsd` file for the original `.xsd` file would add the `myNamespace` namespace to the inbound XML and would remove the `myNamespace` namespace from the outbound XML.

Defining the Choice Condition Schema for a Complex File Structure

In this use case, the Native Format Builder uses order.txt, a complex type file, which contains multiple record types such as order, customer, and items. Also, using this use case you can generate the NXSD and test it. Perform the following steps to run this use case:

1. The data in a sample text file, order.txt, is:

```
Order, ID41678, 20May2000
Item1, GigaWidget, 60, $75
Item2, MegaBucket, 48, $125
Cust1, Hopkins Associates, ID26490
Order, ID41680, 20May2000
Item3, Rt.Clopper, 40, $100
Item4, Lt.Clopper, 50, $100
Cust2, Jersey WebInovaters, ID46786
```

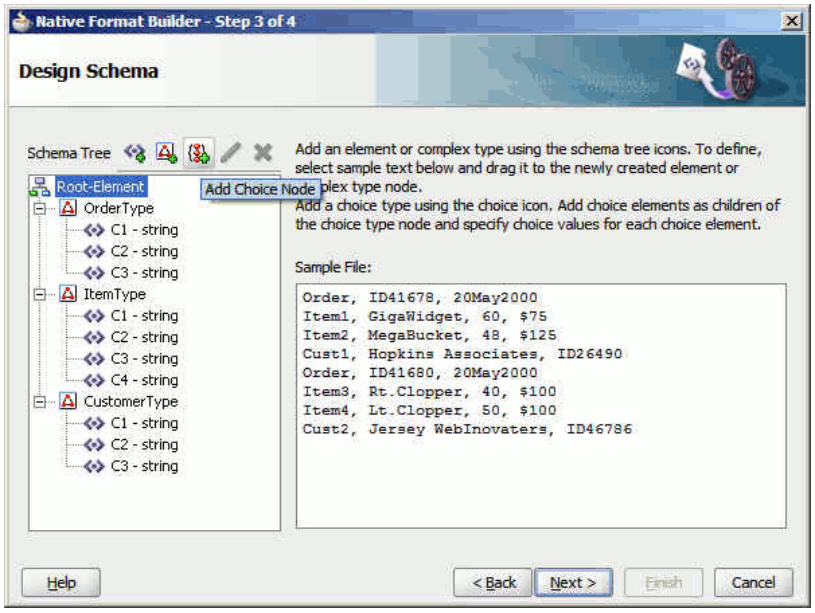
2. Create the following complex types by dragging one row each of order, customer, and item native data:

   - OrderType
   - ItemType
   - CustomerType

   For more information about creating a complex type, see Defining the Schema for a Complex File Structure.

   The Native Format Builder Design Schema page is displayed, as shown in Figure 6-46.

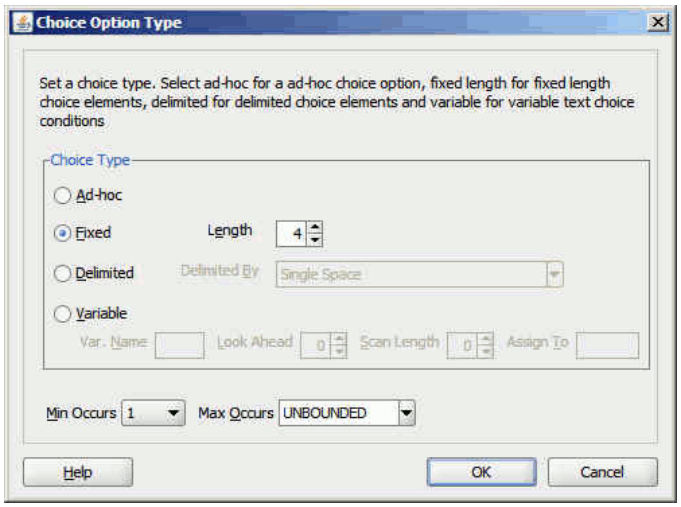
Figure 6-46 Native Format Builder Design Schema Page



3. Click **Add Choice Node**. The Choice Option Type dialog is displayed.

4. Set the options in the Choice Option Type dialog, as shown in Figure 6-47, and then click **OK**.

Figure 6-47 The Choice Option Type Dialog



5. Select **choice** and click the **Add Element** icon. A <new_element> is added to the choice node.

6. Rename the newly added element to OrderChoice, and then drag and drop the OrderType complex type element to OrderChoice.

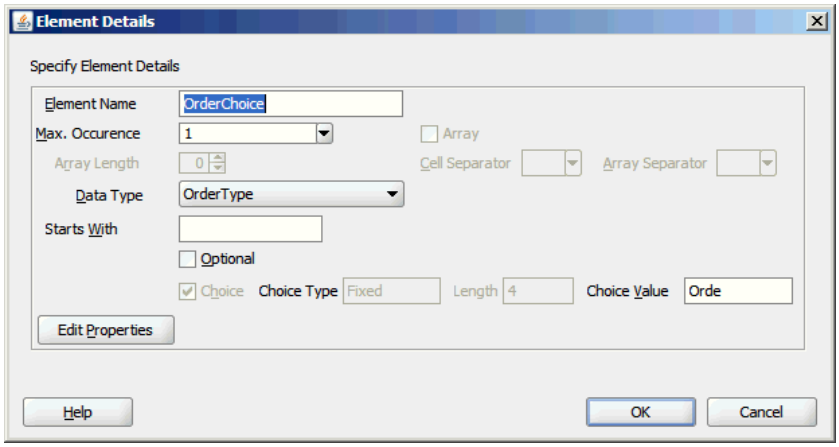
7. Select **OrderChoice - string** and click the **Edit Node** icon. The Element Details dialog is displayed.

8. Enter `Order` in the **Choice Value** field, as shown in Figure 6-48, and then click **OK**.

Note:

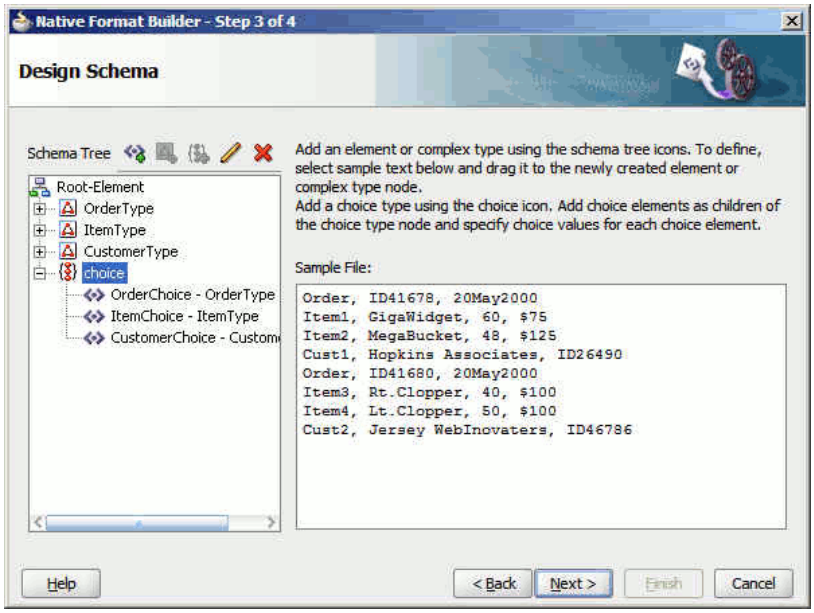
You should specify four characters in Choice Value field as the Length field has the value 4 in it.

Figure 6-48 The Element Details Dialog



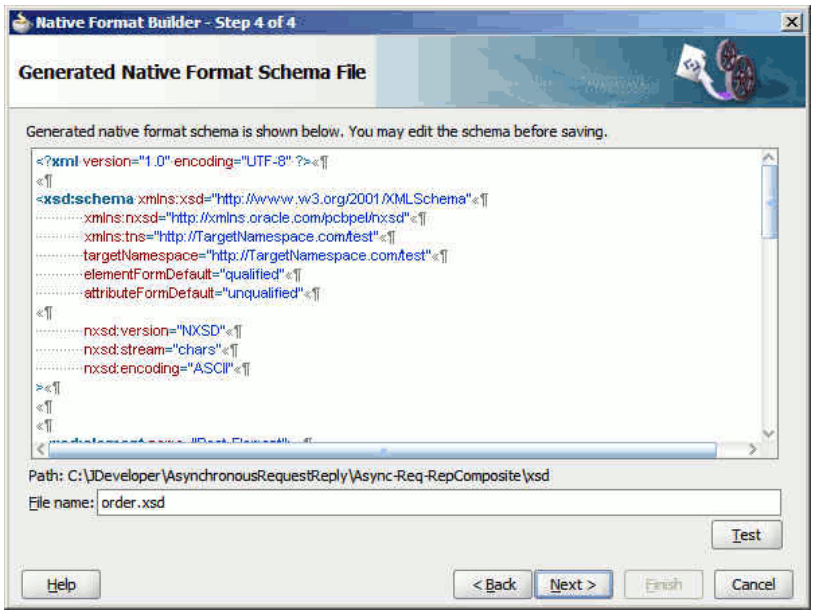
9. Follow Step 5 to 8 to create the ItemChoice choice complex type with ItemType data type and CustomerChoice choice complex type with CustomerType data type. The Native Format Builder Design Schema dialog is displayed, as shown in Figure 6-49.

Figure 6-49 Native Format Builder Design Schema Page



10. Click **Next**. The **Generated Native Format Schema File** page is displayed, as shown in Figure 6-50, which displays the native format file.

Figure 6-50 Generated Native Format Schema File Page



Native Schema

The native schema definition corresponding to the preceding native data can be defined in the following example.

Example - Native Schema Definition

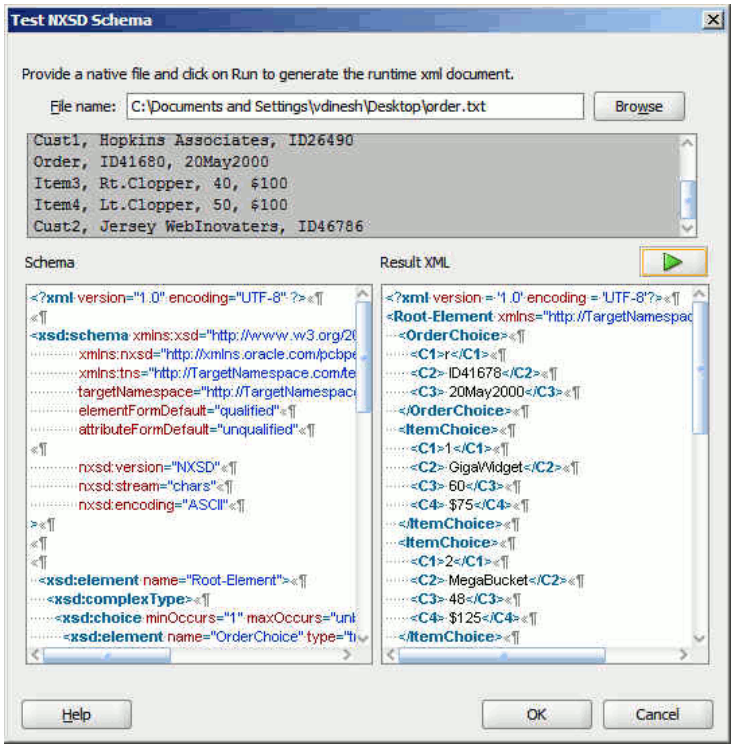
```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd=
             "http://xmlns.oracle.com/pcbpel/nxsd"
            xmlns:tns="http://TargetNamespace.com/test"
            targetNamespace=
                   "http://TargetNamespace.com/test"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            nxsd:version="NXSD"
            nxsd:stream="chars"
            nxsd:encoding="ASCII"
>
  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:choice minOccurs="1"
                  maxOccurs="unbounded"
 nxsd:choiceCondition="fixedLength"
                    nxsd:length="4">
        <xsd:element name="OrderChoice"
             type="tns:OrderType"
 nxsd:conditionValue="Orde" />
        <xsd:element name="ItemChoice"
              type="tns:ItemType"
 nxsd:conditionValue="Item" />
        <xsd:element name="CustomerChoice"
             type="tns:customerType"
 nxsd:conditionValue="Cust" />
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="customerType">
```

```
   <xsd:sequence>
     <xsd:element name="C1"
                     type="xsd:string"
          nxsd:style="terminated"
nxsd:terminatedBy="," />
     <xsd:element name="C2" type="xsd:string"
          nxsd:style="terminated"
nxsd:terminatedBy="," />
     <xsd:element name="C3" type="xsd:string"
          nxsd:style="terminated"
nxsd:terminatedBy="${eol}" />
   </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="ItemType">
   <xsd:sequence>
     <xsd:element name="C1"
           type="xsd:string"
           nxsd:style="terminated"
nxsd:terminatedBy="," />
     <xsd:element name="C2"
                    type="xsd:string"
           nxsd:style="terminated"
nxsd:terminatedBy="," />
     <xsd:element name="C3"
                   type="xsd:string"
           nxsd:style="terminated"
nxsd:terminatedBy="," />
     <xsd:element name="C4"
                  type="xsd:string"
       nxsd:style="terminated"
nxsd:terminatedBy="${eol}" />
   </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="OrderType">
   <xsd:sequence>
     <xsd:element name="C1" type="xsd:string"
           nxsd:style="terminated"
nxsd:terminatedBy="," />
     <xsd:element name="C2" type="xsd:string"
           nxsd:style="terminated"
nxsd:terminatedBy="," />
     <xsd:element name="C3" type="xsd:string"
           nxsd:style="terminated"
nxsd:terminatedBy="${eol}" />
   </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

11. Click **Test**. The **Test NXSD Schema** dialog is displayed.

12. Click the **Generate XML** icon. The Result XML is displayed on the right pane of the **Test NXSD Schema** dialog, as shown in Figure 6-51.

Figure 6-51 Test NXSD Schema Dialog



Translated XML Using the Native Schema

The translated XML is shown in the example below.

Example - Translated XML Using the Native Schema

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<Root-Element xmlns="http://TargetNamespace.com/test">
   <OrderChoice>
      <C1>r</C1>
      <C2> ID41678</C2>
      <C3> 20May2000</C3>
   </OrderChoice>
   <ItemChoice>
      <C1>1</C1>
      <C2> GigaWidget</C2>
      <C3> 60</C3>
      <C4> $75</C4>
   </ItemChoice>
   <ItemChoice>
      <C1>2</C1>
      <C2> MegaBucket</C2>
      <C3> 48</C3>
      <C4> $125</C4>
   </ItemChoice>
   <CustomerChoice>
      <C1>1</C1>
      <C2> Hopkins Associates</C2>
      <C3> ID26490</C3>
   </CustomerChoice>
   <OrderChoice>
      <C1>r</C1>
      <C2> ID41680</C2>
      <C3> 20May2000</C3>
```

```
   </OrderChoice>
   <ItemChoice>
      <C1>3</C1>
      <C2> Rt.Clopper</C2>
      <C3> 40</C3>
      <C4> $100</C4>
   </ItemChoice>
   <ItemChoice>
      <C1>4</C1>
      <C2> Lt.Clopper</C2>
      <C3> 50</C3>
      <C4> $100</C4>
   </ItemChoice>
   <CustomerChoice>
      <C1>2</C1>
      <C2> Jersey WebInovaters</C2>
      <C3> ID46786</C3>
   </CustomerChoice>
</Root-Element>
```

13. Click **OK**. The **Generated Native Format File** page is displayed.
14. Click **Next**. The **Native Format Builder Finish** page is displayed.

Defining Choice Condition With LookAhead for a Complex File Structure

In this use case, the Native Format Builder uses `address.txt`, a complex type file, which contains multiple records with different addresses. In this use case, you would build a schema which has 2 record types. The RecOne record takes data for records ending with text "YES" and the RecTwo record takes data for records ending with text "NO ".

Also, using this use case you can generate the NXSD and test it.

Perform the following steps to run this use case:

1. The data in a sample text file, address.txt is:

```
Name1,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", YES
Name2,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", NO
Name3,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", NO
Name4,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", YES
```

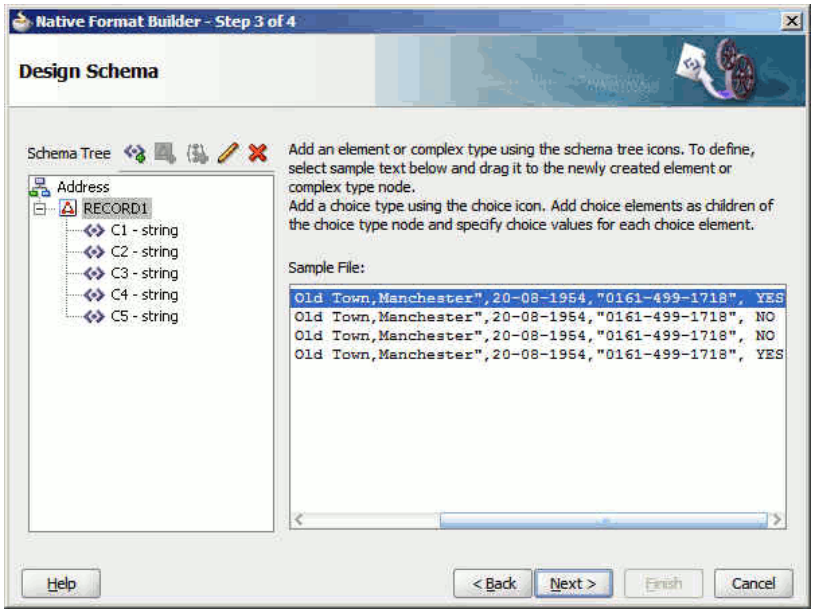
2. Launch the Adapter Configuration Wizard and navigate to the Messages page, and click **Define Schema For Native Format**. The Native Format Builder Welcome page is displayed.
3. Click **Next**. The **Choose Type** page is displayed.
4. Select **Complex Type** and click **Next**. The **Native Format Builder File Description** page is displayed.
5. Click **Browse** and select the address.txt file, and enter **Address** in the **Root Element** field.
6. Click **Next**. The **Native Format Builder Design Schema** page is displayed.
7. Click the **Add Complex Type** icon. A Complex Type, <new_complex_type> is created in the Schema Tree under Address.

8. Select the first row of the sample text from the right-hand pane of the Sample File section, and drag and drop it on the <new_complex_type> node. The **Complex Type Details** dialog is displayed.

9. Enter `RECORD1` in the **Complex Type Name** field and select **Comma (,)** in the **Delimited By** list.

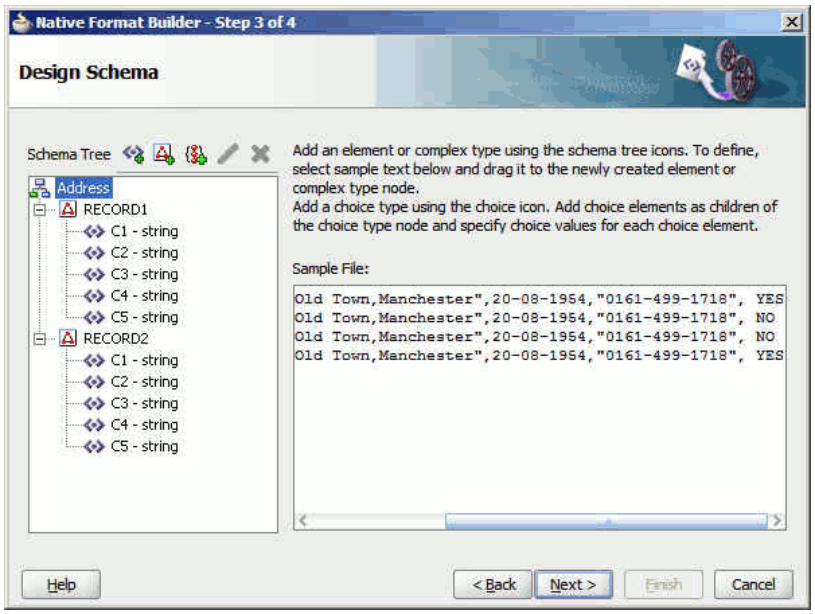
10. Click **OK**. The **Native Format Builder Design Schema** page is displayed, as shown in Figure 6-52.

Figure 6-52 Native Format Builder Design Schema Page



11. Similarly, create another complex type node called RECORD2. The **Native Format Builder Design Schema** page is displayed, as shown in Figure 6-53.

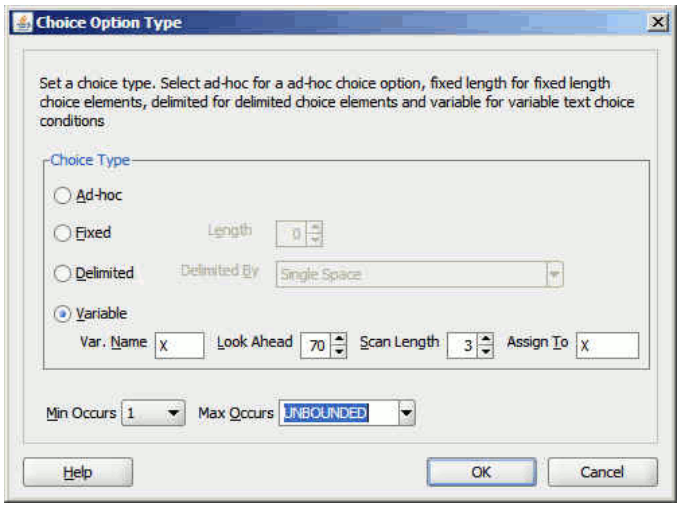
Figure 6-53 Native Format Builder Design Schema Page



12. Click **Add Choice Node**. The Choice Option Type dialog is displayed.

13. Set the options in the Choice Option Type dialog, as shown in Figure 6-54, and then click **OK**.

Figure 6-54 The Choice Option Type Dialog



14. Select **choice** and click the **Add Element** icon. A <new_element> is added to the choice node.

15. Click the **Edit Node** icon. The **Element Details** dialog is displayed.

16. Enter `RECONE` in the Element Name field and select **RECORD1** as the Data Type set choice condition as `YES`, and then click **OK**.

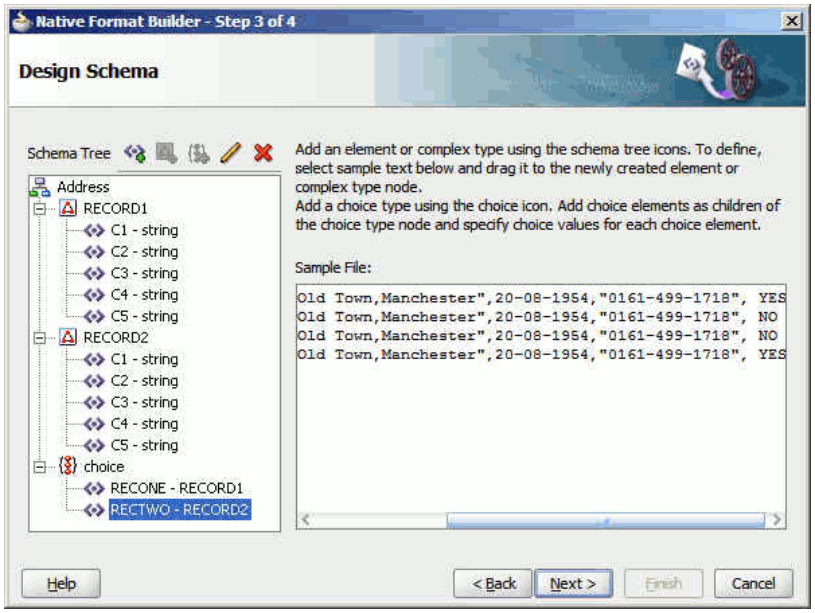
17. Follow Step 14 to 16 to create the `RECTWO` choice element for the choice node and set choice condition as `NO`.

Note:

There is one space after characters `NO`, because you must match the total number of characters to three.

The **Native Format Builder Design Schema** dialog is displayed, as shown in Figure 6-55.

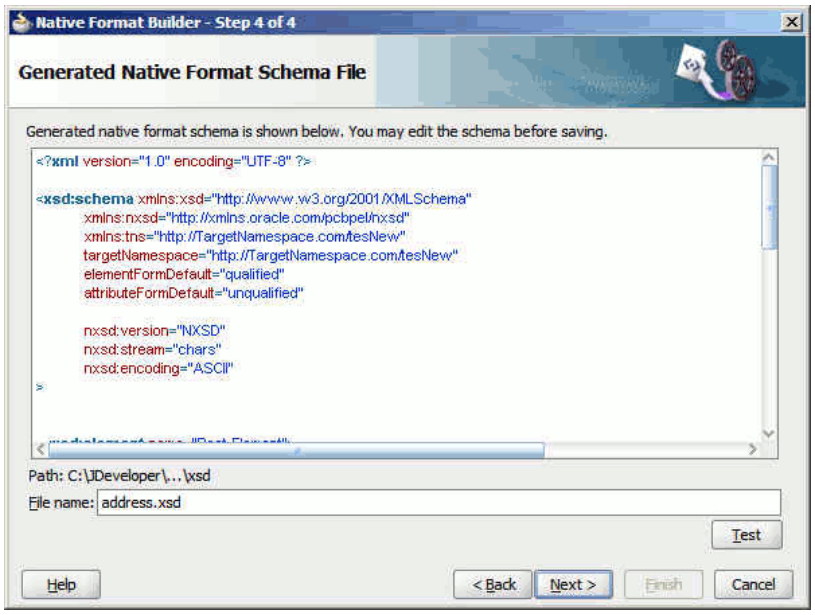
Figure 6-55 Native Format Builder Design Schema Page



18. Drag and drop the RECORD1 complex type to the RECONE element under choice and the RECORD2 complex type to the RECTWO element under choice. The **Native Format Builder Design Schema** dialog is displayed.

19. Click **Next**. The **Generated Native Format Schema File** page is displayed, as shown in Figure 6-56, which displays the native format file.

Figure 6-56 Generated Native Format Schema File Page



Native Schema

The native schema definition corresponding to the preceding native data can be defined as follows:

Example - Native Schema for Defining Choice Condition With LookAhead for a Complex File Structure

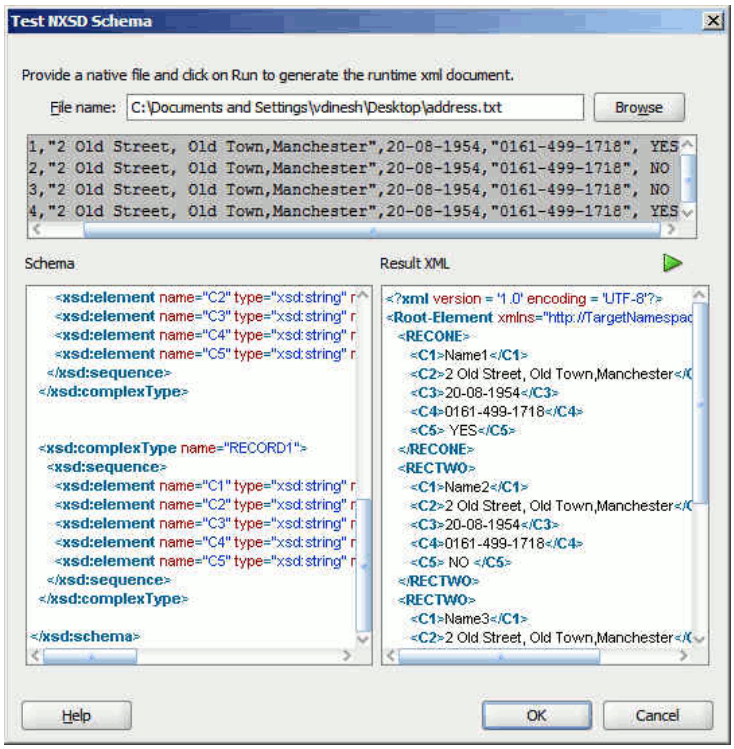
```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd=
                  "http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd=
                  "http://xmlns.oracle.com/pcbpel/nxsd"
            xmlns:tns="http://TargetNamespace.com/tesNew"
            targetNamespace=
                  "http://TargetNamespace.com/tesNew"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            nxsd:version="NXSD"
            nxsd:stream="chars"
            nxsd:encoding="ASCII"
>
  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:choice minOccurs="1"
                  maxOccurs="unbounded"
 nxsd:choiceCondition="${X}" nxsd:lookAhead="70"
 nxsd:scanLength="3"
 nxsd:assignTo="${X}">
        <xsd:element name="RECTWO"
         type="tns:RECORD2"
              nxsd:conditionValue="NO "
 />
        <xsd:element name="RECONE"
         type="tns:RECORD1"
           nxsd:conditionValue="YES"
 />
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="RECORD2">
    <xsd:sequence>
      <xsd:element name="C1"
          type="xsd:string"
       nxsd:style="terminated"
       nxsd:terminatedBy="," />
      <xsd:element name="C2"
           type="xsd:string"
       nxsd:style="terminated"
       nxsd:terminatedBy="," />
      <xsd:element name="C3"
               type="xsd:string"
       nxsd:style="terminated"
       nxsd:terminatedBy="," />
      <xsd:element name="C4"
                  type="xsd:string"
       nxsd:style="terminated"
       nxsd:terminatedBy="," />
      <xsd:element name="C5"
                  type="xsd:string"
       nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="RECORD1">
    <xsd:sequence>
      <xsd:element name="C1"
                   type="xsd:string"
        nxsd:style="terminated"
```

```
        nxsd:terminatedBy="," />
      <xsd:element name="C2"
                type="xsd:string"
        nxsd:style="terminated"
        nxsd:terminatedBy="," />
      <xsd:element name="C3"
                 type="xsd:string"
        nxsd:style="terminated"
        nxsd:terminatedBy="," />
      <xsd:element name="C4"
                    type="xsd:string"
        nxsd:style="terminated"
        nxsd:terminatedBy="," />
      <xsd:element name="C5"
                  type="xsd:string"
        nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

20. Click **Test**. The **Test NXSD Schema** dialog is displayed.

21. Click the **Generate XML** icon. The Result XML is displayed on the right pane of the **Test NXSD Schema** dialog, as shown in Figure 6-57.

Figure 6-57 Test NXSD Schema Dialog



Translated XML Using the Native Schema

The translated XML is shown in the example below.

Example - Translated XML Using the Native Schema

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<Root-Element xmlns="http://TargetNamespace.com/tesNew">
   <RECONE>
      <C1>Name1</C1>
      <C2>2 Old Street, Old Town,Manchester</C2>
      <C3>20-08-1954</C3>
      <C4>0161-499-1718</C4>
      <C5> YES</C5>
   </RECONE>
   <RECTWO>
      <C1>Name2</C1>
      <C2>2 Old Street, Old Town,Manchester</C2>
      <C3>20-08-1954</C3>
      <C4>0161-499-1718</C4>
      <C5> NO </C5>
   </RECTWO>
   <RECTWO>
      <C1>Name3</C1>
      <C2>2 Old Street, Old Town,Manchester</C2>
      <C3>20-08-1954</C3>
      <C4>0161-499-1718</C4>
      <C5> NO </C5>
   </RECTWO>
   <RECONE>
      <C1>Name4</C1>
      <C2>2 Old Street, Old Town,Manchester</C2>
      <C3>20-08-1954</C3>
      <C4>0161-499-1718</C4>
      <C5> YES</C5>
   </RECONE>
</Root-Element>
```

Note:

There are 2 recordtypes: RECONE and RECTWO. RECONE takes records that end with character YES and RECTWO takes records that end with character NO.

22. Click **OK**. The Generated Native Format File page is displayed.

23. Click **Next**. The Native Format Builder Finish page is displayed.

Defining Array Type Schema for a Complex File Structure

In this use case, the Native Format Builder uses array.txt, a complex type file, which contains an array of items. The sample data has four names which are separated by a semicolon and ending with a period. In this use case, you would create a schema with array type which has member names separated by a semicolon and array terminated by a period. Also, using this use case you can generate the NXSD and test it.

Perform the following steps to run this use case:

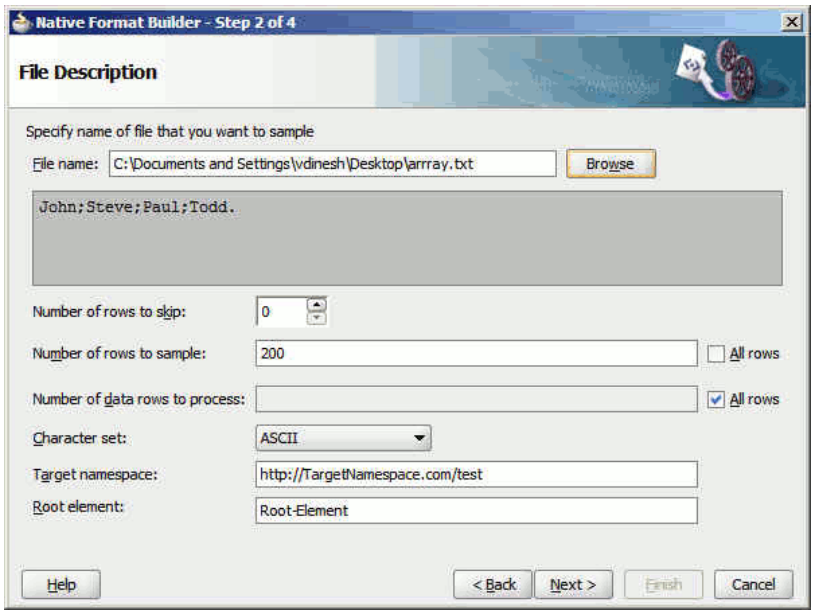
1. The data in a sample text file, array.txt, is:

   ```
   John;Steve;Paul;Todd.
   ```

2. Launch the Adapter Configuration Wizard and navigate to the Messages page and click **Define Schema For Native Format**. The Native Format Builder Welcome page is displayed.

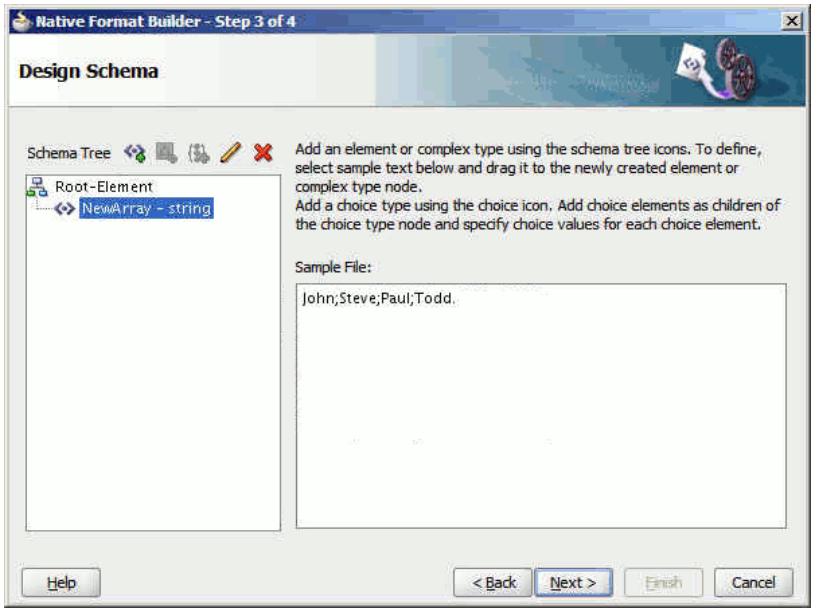
3. Click **Next**. The Choose Type page is displayed.
4. Select **Complex Type** and click **Next**. The Native Format Builder File Description page is displayed.
5. Click **Browse** and select the array.txt file, as shown in Figure 6-58. The Native Format Builder File Description page is displayed.

Figure 6-58 Native Format Builder File Description Page



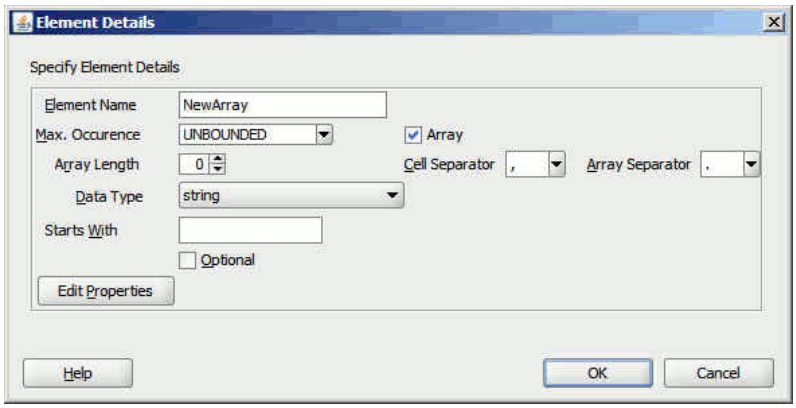
6. Click **Next**. The Native Format Builder Design Schema Page is displayed.
7. Create a global element called NewArray and drag and drop the native data to the newly created global element.
8. Select **NewArray**, as shown in Figure 6-59, and click the **Edit Node** icon. The **Element Details** dialog is displayed.

Figure 6-59 Native Format Builder Design Schema Page



9. Set the options in the **Element Details** dialog, as shown in Figure 6-60, and then click **OK**.

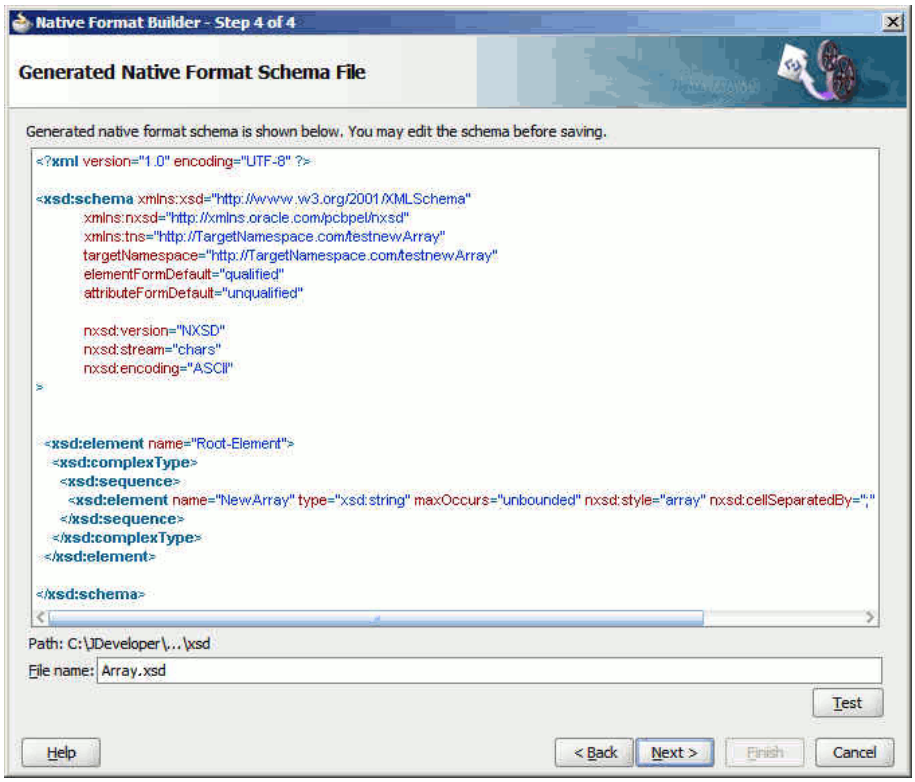
Figure 6-60 Element Details Dialog



The **Native Format Builder Design Schema** dialog is displayed.

10. Click **Next**. The **Generated Native Format Schema File** page is displayed, as shown in Figure 6-61, which displays the native format file.

Figure 6-61 Generated Native Format Schema File Page



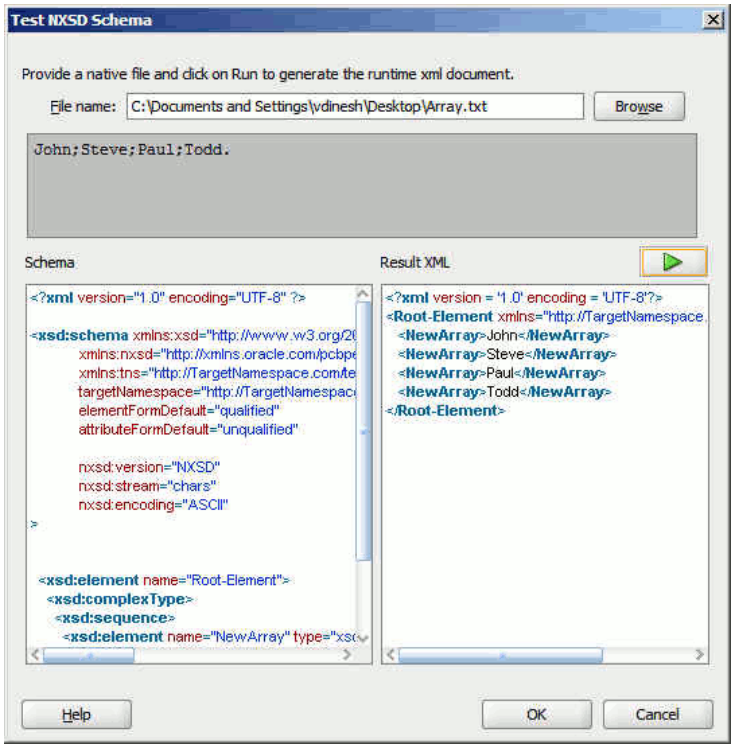
Native Schema

The native schema definition corresponding to the preceding native data can be defined as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            xmlns:tns="http://TargetNamespace.com/testnewArray"
            targetNamespace="http://TargetNamespace.com/testnewArray"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            nxsd:version="NXSD"
            nxsd:stream="chars"
            nxsd:encoding="ASCII"
>
  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="NewArray" type="xsd:string" maxOccurs="unbounded"
 nxsd:style="array" nxsd:cellSeparatedBy=";" nxsd:arrayTerminatedBy="." />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

11. Click **Test**. The Test NXSD Schema dialog is displayed.
12. Click the **Generate XML** icon. The Result XML is displayed on the right pane of the Test NXSD Schema dialog, as shown in Figure 6-62.

Figure 6-62 Test NXSD Schema Dialog



Translated XML Using the Native Schema

The translated XML looks as follows:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<Root-Element xmlns="http://TargetNamespace.com/testnewArray">
   <NewArray>John</NewArray>
   <NewArray>Steve</NewArray>
   <NewArray>Paul</NewArray>
   <NewArray>Todd</NewArray>
</Root-Element>
```

13. Click **OK**. The Generated Native Format File page is displayed.
14. Click **Next**. The Native Format Builder Finish page is displayed.

Defining the Schema for a DTD File Structure

This use case takes you through the procedure for defining the schema for the native data type, DTD file.

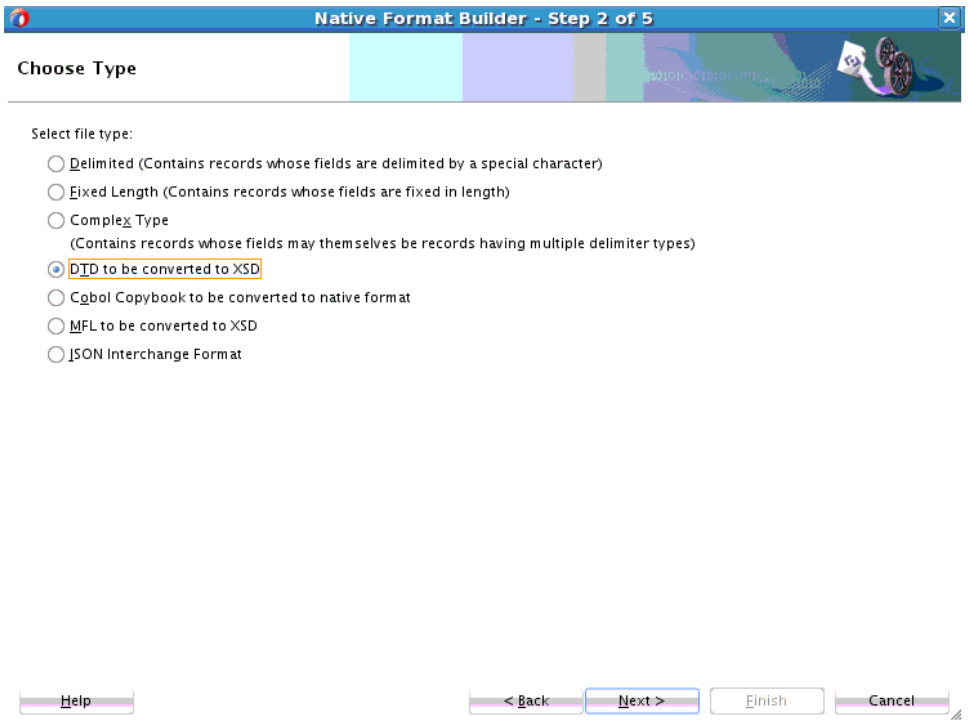
Use the **DTD to be converted to XSD** option in the Native Format Builder wizard when creating the XML schema for this native file.

In this use case, the Native Format Builder uses a DTD file type `*.dtd`. You can generate the corresponding NXSD and also test it. Perform the following steps to run the use case:

1. Use any DTD file.
2. Launch the Adapter Configuration Wizard and navigate to the Messages page, as displayed in Figure 6-4, and click **Define Schema For Native Format**. The Native Format Builder Welcome page is displayed, as shown in Figure 6-5.

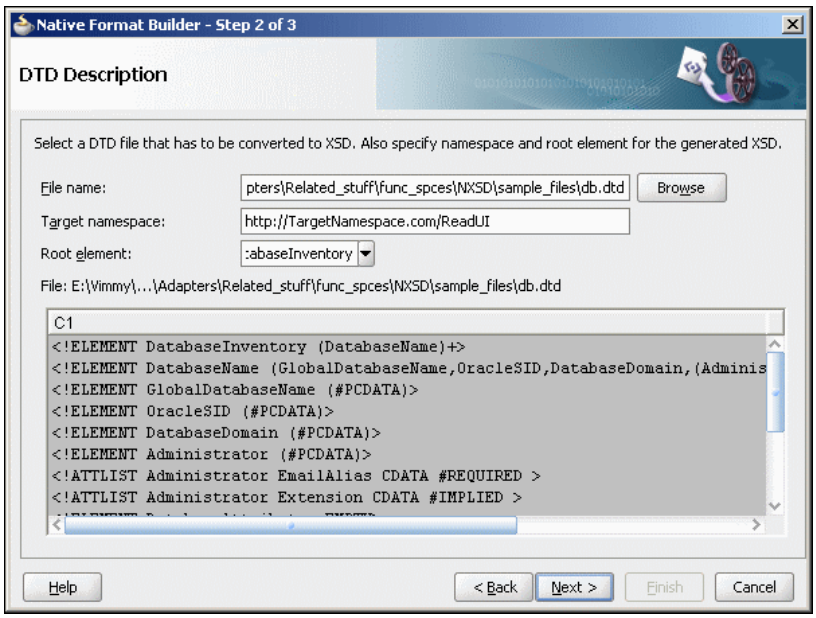
3. Click **Next**. The Choose Type page is displayed, as shown in Figure 6-26.
4. Select **DTD to be converted to XSD**. The Choose Type page is displayed, as shown in Figure 6-63.

Figure 6-63 Native Format Builder Wizard Choose Type Page



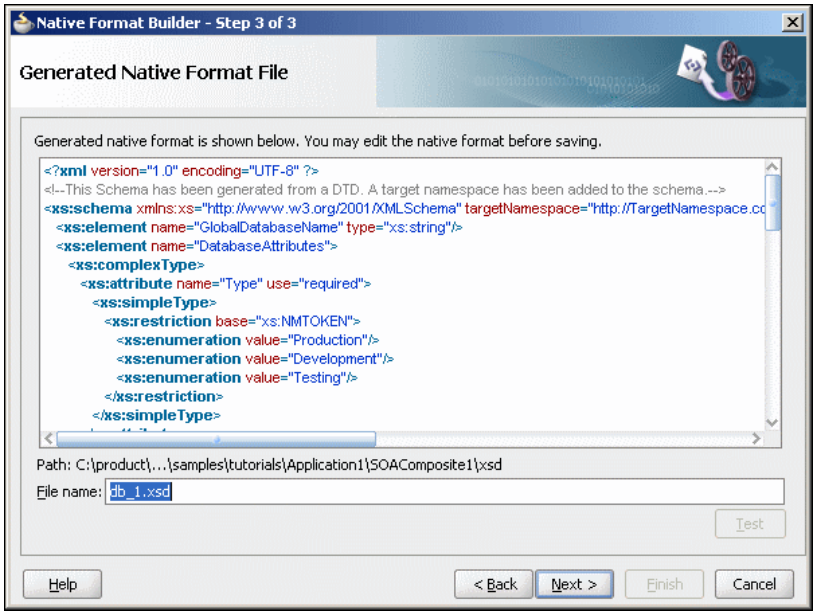
5. Click **Next**. The Native Format Builder DTD Description page is displayed.
6. Click **Browse** and select the `db.dtd` file, and select **DatabaseInventory** from the Root Element list, as displayed in Figure 6-64.

Figure 6-64 Native Format Builder Wizard File Description Page



7. Click **Next**. The Generated Native Format File page is displayed, as shown in Figure 6-65.

Figure 6-65 Native Format Builder Wizard Field Properties Page



The resulting generated Native Format file is shown in the example below.

Example - Native Format Generated File

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--This Schema has been generated from a DTD.
A target namespace has beenadded to the schema.-->
<xs:schema xmlns:
          xs="http://www.w3.org/2001/XMLSchema"
targetNamespace=
              "http://TargetNamespace.com/ReadUI"
xmlns="http://TargetNamespace.com/ReadUI"
                nxsd:version="DTD"
 xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd">
   <xs:element name="GlobalDatabaseName"
                     type="xs:string"/>
   <xs:element name="DatabaseAttributes">
      <xs:complexType>
         <xs:attribute name="Type"
                        use="required">
            <xs:simpleType>
               <xs:restriction base="xs:NMTOKEN">
                  <xs:enumeration
                            value="Production"/>
                  <xs:enumeration
                            value="Development"/>
                  <xs:enumeration
                            value="Testing"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="Version"
                    use="optional" default="9i">
            <xs:simpleType>
               <xs:restriction base=
```

```
                              "xs:NMTOKEN">
                     <xs:enumeration value="7"/>
                     <xs:enumeration value="8"/>
                     <xs:enumeration value="8i"/>
                     <xs:enumeration value="9i"/>
                  </xs:restriction>
               </xs:simpleType>
            </xs:attribute>
         </xs:complexType>
      </xs:element>
      <xs:element name="Comments" type="xs:string"/>
      <xs:element name="Administrator">
         <xs:complexType>
            <xs:simpleContent>
               <xs:extension base="xs:string">
                  <xs:attribute name="EmailAlias"
                       use="required"
    type="xs:string"/>
                  <xs:attribute name="Extension"
                      use="optional" type="xs:string"/>
               </xs:extension>
            </xs:simpleContent>
         </xs:complexType>
      </xs:element>
      <xs:element name="OracleSID"
                    type="xs:string"/>
      <xs:element name="DatabaseName">
         <xs:complexType>
            <xs:sequence>
               <xs:element
                    ref="GlobalDatabaseName"/>
               <xs:element
                    ref="OracleSID"/>
               <xs:element
                    ref="DatabaseDomain"/>
               <xs:element
                 maxOccurs="unbounded" ref="Administrator"/>
               <xs:element
                 ref="DatabaseAttributes"/>
               <xs:element
                 ref="Comments"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:element name="DatabaseDomain"
                    type="xs:string"/>
      <xs:element name="DatabaseInventory">
         <xs:complexType>
            <xs:sequence>
               <xs:element
                      maxOccurs="unbounded"
                       ref="DatabaseName"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
   </xs:schema>
```

8. Click **Next**. The Native Format Builder Finish page is displayed.

9. Click **Finish**. The Adapter Configuration Wizard Messages page is displayed containing the generated NXSD.

Defining the Schema for a COBOL Copybook File Structure

This use case shows how the Oracle File and FTP Adapters process a file in COBOL Copybook format (through use of the Native Format Builder wizard) to create a native schema file for translation.

The following COBOL Copybook examples are provided:

- Multiple Root Levels
- Single Root Level_ Virtual Decimal_ Fixed-Length Array
- Variable Length Array
- Numeric Types

Multiple Root Levels

A COBOL Copybook can have multiple root levels. If all root levels are at `01` level, then each such group implicitly redefines the other.

In this use case, the Native Format Builder uses a fixed-length file type, `po-ccb.cpy`, that contains the purchase order details such as buyer name, address, and items. Every element in this `po-ccb.cpy` native file has a fixed length. The data in the sample text file, `po-ccb.cpy`, appears as follows:

```
05 PO-RECORD.
10 PO-BUYER.
15 PO-UID PIC 9(7).
15 PO-NAME PIC X(15).
15 PO-ADDRESS.
20 PO-STREET PIC X(15).
20 PO-CITY PIC X(10).
20 PO-ZIP PIC 9(5).
20 PO-STATE PIC X(2).
10 PO-ITEM.
15 POITEM OCCURS 3 TIMES.
20 PO-LINE-ITEM.
25 PO-ITEM-ID PIC 9(3).
25 PO-ITEM-NAME PIC X(40).
25 PO-ITEM-QUANTITY PIC 9(2).
25 PO-ITEM-PRICE PIC 9(5)V9(2).
10 PO-TOTALPIC 9(7)V9(2).
```

You can generate the corresponding NXSD and also test it. Perform the following steps to run the use case:

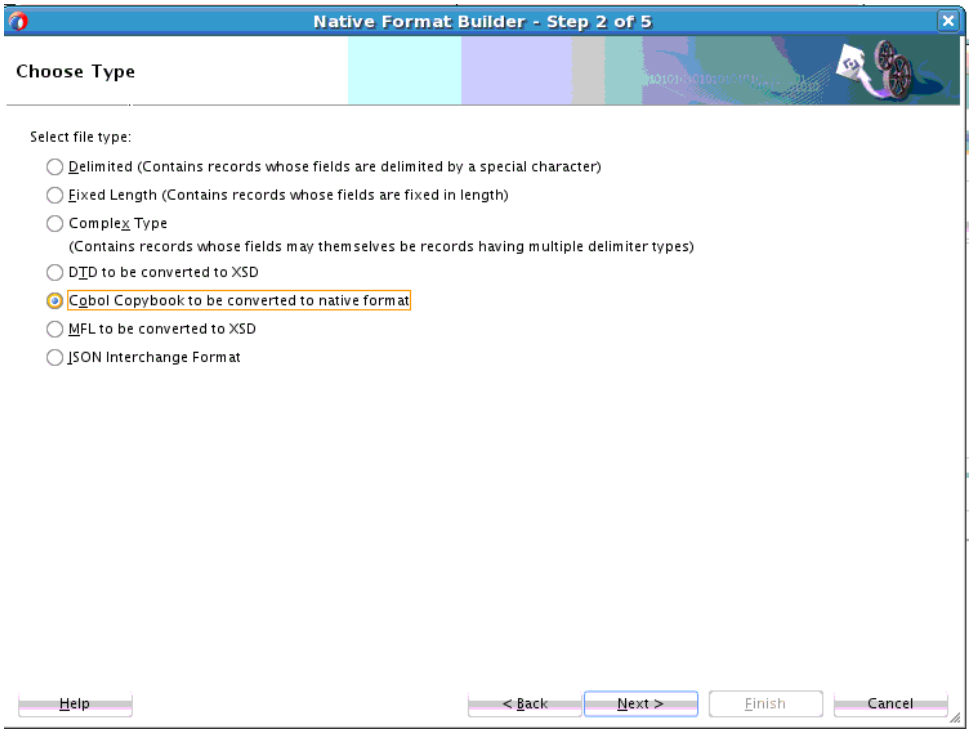
1. Get the following files from the `artifacts.zip` file contained in the `Adapters-105CobolCopyBook` sample.
   - `artifacts/samples/po-ccb.cpy`
   - `artifacts/samples/po-ebcdic.data`

   You can obtain the `Adapters-105CobolCopyBook` sample by accessing the Oracle SOA Sample Code site.

   Copy these files to your samples directory.

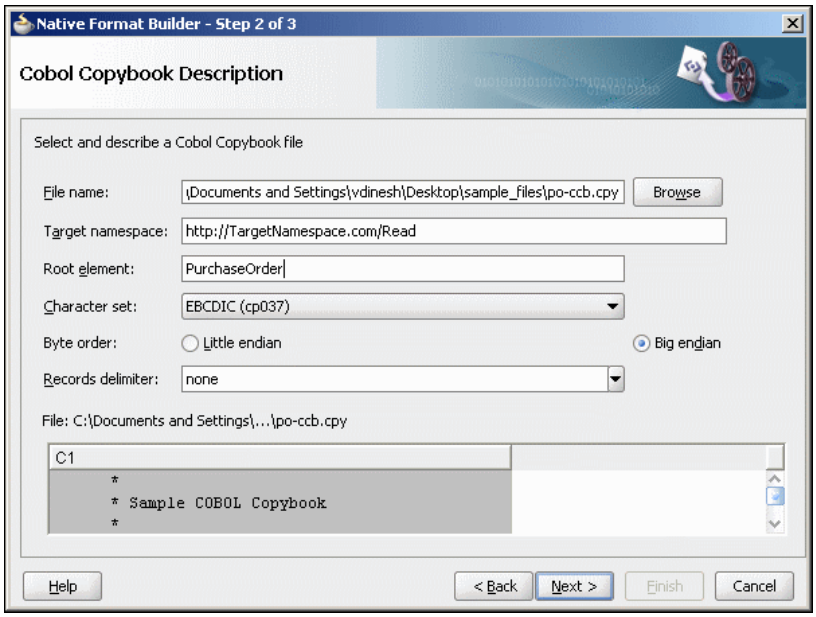
2. Launch the Adapter Configuration Wizard and navigate to the Messages page, as displayed in Figure 6-4, and click **Define Schema For Native Format**. The **Native Format Builder Welcome** page is displayed, as shown in Figure 6-5.
3. Click **Next**. The **Choose Type** page is displayed, as shown in Figure 6-26.
4. Select **Cobol Copybook to be converted to native format**. The **Choose Type** page is displayed, as shown in Figure 6-66.

Figure 6-66 Native Format Builder Wizard Choose Type Page



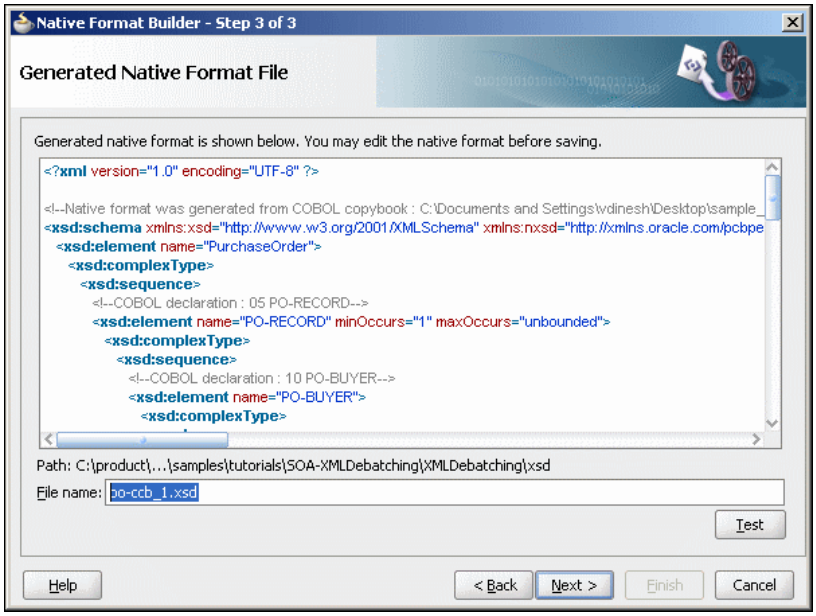
5. Click **Next**. The **Native Format Builder Cobol Copybook Description** page is displayed.
6. Click **Browse** and select the `po-ccb.cpy` file, as shown in Figure 6-67.

Figure 6-67 Native Format Builder Wizard File Description Page



7. Enter `PurchaseOrder` in the **Root-Element** field, and click **Next**. The **Generated Native Format File** page is displayed, as shown in Figure 6-68.

Figure 6-68 Native Format Builder Wizard Generated Native Format File Page



The top level payroll records are enclosed in a choice model group. Each payroll record also has two attributes, `nxsd:lookAhead` and `nxsd:lookFor` that help identify the type of record during runtime processing of the data file. So, you must add values for these attributes. For example, assume PAYROLL-F-RECORD occurs when the PAYROLL-F-TRANS-CODE field has a value of FR. The record element then looks as follows:

```
<xsd:element name="PAYROLL-F-RECORD" nxsd:lookAhead="10" nxsd:lookFor="FR">
```

The value 10 indicates the position of the lookahead field. The following COBOL Copybook has multiple root elements at the 05 level:

```
05 ORG-NUM           PIC 99.
05 EMP-RECORD.
  10 EMP-SSN       PIC 9(4)V(6).
  10 EMP-WZT       PIC 9(6).
```

The native schema is shown in the example below:

Example - Native Schema

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--Native format was generated from COBOL copybook : C:\Documents and
 Settings\vdinesh\Desktop\sample_files\po-ccb.cpy-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
 xmlns:extn="http://xmlns.oracle.com/pcbpel/nxsd/extensions"
 targetNamespace="http://TargetNamespace.com/Read"
xmlns:tns="http://TargetNamespace.com/Read"
                          elementFormDefault="qualified"
 attributeFormDefault="unqualified" nxsd:version="NXSD"
                          nxsd:encoding="cp037"
 nxsd:byteOrder="bigEndian" nxsd:stream="chars">
   <xsd:element name="PurchaseOrder">
      <xsd:complexType>
         <xsd:sequence>
            <!--COBOL declaration : 05 PO-RECORD-->
            <xsd:element name="PO-RECORD"
                           minOccurs="1"
                              maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <!--COBOL declaration : 10 PO-BUYER-->
                     <xsd:element name="PO-BUYER">
                        <xsd:complexType>
                           <xsd:sequence>
                              <!--COBOL declaration :
                                          15 PO-UID PIC 9(7)-->
                              <xsd:element name="PO-UID"
                                       type="xsd:long"
                                nxsd:style="fixedLength"
                                          nxsd:padStyle="head"
                                nxsd:paddedBy="0"
                                nxsd:length="7"/>
                              <!--COBOL declaration :
                                          15 PO-NAME PIC X(15)-->
                              <xsd:element name="PO-NAME"
                                          type="xsd:string"
                                nxsd:style="fixedLength"
                                          nxsd:padStyle="tail"
                                nxsd:paddedBy=" "
                                nxsd:length="15"/>
                              <!--COBOL declaration :
                                            15 PO-ADDRESS-->
                              <xsd:element name="PO-ADDRESS">
                                 <xsd:complexType>
                                    <xsd:sequence>
                                       <!--COBOL declaration :
                                           20 PO-STREET PIC X(15)-->
                                       <xsd:element name="PO-STREET"
                                        type="xsd:string"
```

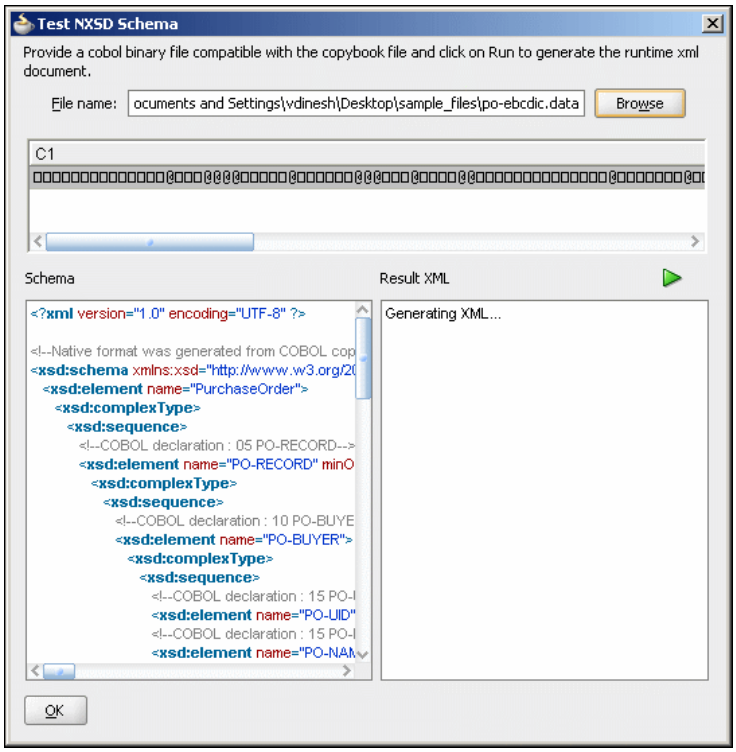
```
                                        nxsd:style="fixedLength"
                                        nxsd:padStyle="tail"
                                        nxsd:paddedBy=" "
                                              nxsd:length="15"/>
                                       <!--COBOL declaration :
                                         20 PO-CITY PIC X(10)-->
                                       <xsd:element name="PO-CITY"
                                        type="xsd:string"
                                        nxsd:style="fixedLength"
                                        nxsd:padStyle="tail"
                                        nxsd:paddedBy=" "
                                               nxsd:length="10"/>
                                  <!--COBOL declaration : 20 PO-ZIP PIC 9(5)-->
                                              <xsd:element name="PO-ZIP"
                                               type="xsd:long"
                                               nxsd:style="fixedLength"
                                               nxsd:padStyle="head"
                                               nxsd:paddedBy="0" nxsd:length="5"/>
                                              <!--COBOL declaration : 20 PO-STATE PIC
X(2)-->
                                              <xsd:element name="PO-STATE"
                                               type="xsd:string"
                                               nxsd:style="fixedLength"
                                               nxsd:padStyle="tail"
                                               nxsd:paddedBy=" " nxsd:length="2"/>
                                           </xsd:sequence>
                                        </xsd:complexType>
                                     </xsd:element>
                                   </xsd:sequence>
                                </xsd:complexType>
                             </xsd:element>
                             <!--COBOL declaration : 10 PO-ITEM-->
                             <xsd:element name="PO-ITEM">
                                <xsd:complexType>
                                   <xsd:sequence>
                                      <!--COBOL declaration :
                                                  15 POITEM OCCURS 3 TIMES-->
                                      <xsd:element name=
                                            "POITEM" minOccurs="3" maxOccurs="3">
                                         <xsd:complexType>
                                            <xsd:sequence>
                                               <!--COBOL declaration
                                                     : 20 PO-LINE-ITEM-->
                                               <xsd:element name="PO-LINE-ITEM">
                                                  <xsd:complexType>
                                                     <xsd:sequence>
                                          <!--COBOL declaration :
                                                     25 PO-ITEM-ID PIC 9(3)-->
                                          <xsd:element name="PO-ITEM-ID"
                                          type="xsd:long"
                                          nxsd:style="fixedLength"
                                                          nxsd:padStyle="head"
                                          nxsd:paddedBy="0" nxsd:length="3"/>
                                         <!--COBOL declaration :
                                                     25 PO-ITEM-NAME PIC X(40)-->
                                          <xsd:element name="PO-ITEM-NAME"
                                          type="xsd:string"
                                          nxsd:style="fixedLength"
                                          nxsd:padStyle="tail"
                                          nxsd:paddedBy="
                                         " nxsd:length="40"/>
```

```
                                    <!--COBOL declaration :
                                           25 PO-ITEM-QUANTITY PIC 9(2)-->
                                           <xsd:element name=
                                                "PO-ITEM-QUANTITY"
                                             type="xsd:long"
                                             nxsd:style="fixedLength"
                                             nxsd:padStyle="head"
                                             nxsd:paddedBy="0"
                                             nxsd:length="2"/>
                                    <!--COBOL declaration :
                                              25 PO-ITEM-PRICE PIC 9(5)V9(2)-->
                                              <xsd:element name=
                                                 "PO-ITEM-PRICE"
                                              type="xsd:decimal"
                                              nxsd:style="virtualDecimal"
                                      extn:assumeDecimal="5"
                                              extn:picSize="7"/>
                                                 </xsd:sequence>
                                              </xsd:complexType>
                                            </xsd:element>
                                         </xsd:sequence>
                                       </xsd:complexType>
                                     </xsd:element>
                                  </xsd:sequence>
                               </xsd:complexType>
                            </xsd:element>
                            <!--COBOL declaration :
                                10 PO-TOTAL PIC 9(7)V9(2)-->
                            <xsd:element name="PO-TOTAL"
                              type="xsd:decimal"
                              nxsd:style="virtualDecimal"
                              extn:assumeDecimal="7" extn:picSize="9"/>
                         </xsd:sequence>
                      </xsd:complexType>
                   </xsd:element>
                </xsd:sequence>
             </xsd:complexType>
          </xsd:element>
       </xsd:schema>
```

8. Click **Test**. The **Test NXSD Schema** dialog is displayed.

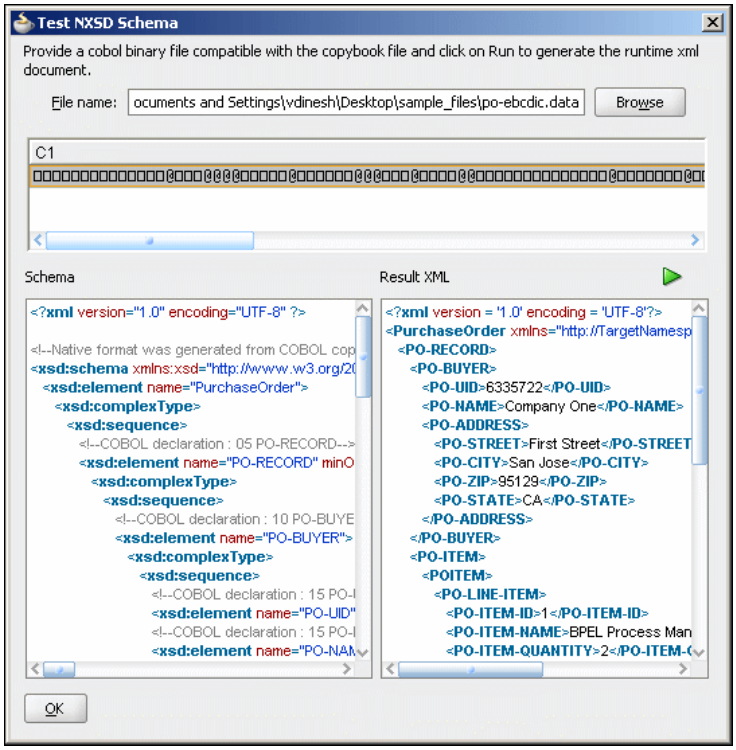
9. Click **Browse** and select the `po-ebcdic.data` file in the **File Name** field. The **Test NXSD Schema** dialog is displayed, as shown in Figure 6-69.

Figure 6-69 Test NXSD Schema Dialog



10. Click the **Generate XML** icon. The Result XML is displayed on the right pane of the **Test NXSD Schema** dialog, as shown in Figure 6-70.

Figure 6-70 Test NXSD Schema Dialog



The native data using the corresponding native schema format is translated to the following XML, as shown in the example below.

Example - Native Data Using the Corresponding Native Schema

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<PurchaseOrder
            xmlns="http://TargetNamespace.com/Read">
   <PO-RECORD>
      <PO-BUYER>
         <PO-UID>6335722</PO-UID>
         <PO-NAME>Company One</PO-NAME>
         <PO-ADDRESS>
            <PO-STREET>First Street</PO-STREET>
            <PO-CITY>San Jose</PO-CITY>
            <PO-ZIP>95129</PO-ZIP>
            <PO-STATE>CA</PO-STATE>
         </PO-ADDRESS>
      </PO-BUYER>
      <PO-ITEM>
         <POITEM>
            <PO-LINE-ITEM>
               <PO-ITEM-ID>1</PO-ITEM-ID>
               <PO-ITEM-NAME>
                     BPEL Process Manager Enterprise
 Edition</PO-ITEM-NAME>
               <PO-ITEM-QUANTITY>2
                           </PO-ITEM-QUANTITY>
               <PO-ITEM-PRICE>40000.0
                           </PO-ITEM-PRICE>
            </PO-LINE-ITEM>
         </POITEM>
         <POITEM>
            <PO-LINE-ITEM>
               <PO-ITEM-ID>2</PO-ITEM-ID>
               <PO-ITEM-NAME>
                        BPEL Process Manager Standard
 Edition</PO-ITEM-NAME>
               <PO-ITEM-QUANTITY>5
                        </PO-ITEM-QUANTITY>
               <PO-ITEM-PRICE>50000.0
                        </PO-ITEM-PRICE>
            </PO-LINE-ITEM>
         </POITEM>
         <POITEM>
            <PO-LINE-ITEM>
               <PO-ITEM-ID>3</PO-ITEM-ID>
               <PO-ITEM-NAME>
                       BPEL Process Manager Developer
 Edition</PO-ITEM-NAME>
               <PO-ITEM-QUANTITY>20
                           </PO-ITEM-QUANTITY>
               <PO-ITEM-PRICE>20000.0
                         </PO-ITEM-PRICE>
            </PO-LINE-ITEM>
         </POITEM>
      </PO-ITEM>
      <PO-TOTAL>730000.0</PO-TOTAL>
   </PO-RECORD>
</PurchaseOrder>
```

In this (non-01 level) case, an unbounded sequence of the root level items is generated.

11. Click **OK**. The **Generated Native Format File** page is displayed.
12. Click **Next**. The **Native Format Builder Finish** page is displayed.
13. Click **Finish**. The **Adapter Configuration Wizard Messages** page is displayed, containing the generated NXSD.

Single Root Level, Virtual Decimal, Fixed-Length Array

The following COBOL Copybook has a single root level item PO-RECORD. In a single root level case, the level number does not matter because the converter works in the same way. This COBOL Copybook also shows an example of a field declared as a virtual decimal (PO-ITEM-PRICE).

```
05 PO-RECORD.
      10 PO-BUYER.
        15 PO-UID    PIC 9(7).
        15 PO-NAME   PIC X(15).
        15 PO-ADDRESS.
            20 PO-STREET PIC X(15).
            20 PO-CITY   PIC X(10).
            20 PO-ZIP    PIC 9(5).
            20 PO-STATE  PIC X(2).
      10 PO-ITEM.
        15 POITEM OCCURS 3 TIMES.
            20 PO-LINE-ITEM.
              25 PO-ITEM-ID       PIC 9(3).
              25 PO-ITEM-NAME     PIC X(40).
              25 PO-ITEM-QUANTITY PIC 9(2).
              25 PO-ITEM-PRICE    PIC 9(5)V9(2).
      10 PO-TOTAL PIC  9(7)V9(2).
```

The generated schema looks as follows:

Example - Generated Schema

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--Native format was generated from COBOL copybook:D:\work\
jDevProjects\CCB\Copybooks\po-ccb.cpy-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
       xmlns:extn="http://xmlns.oracle.com/
                             pcbpel/nxsd/extensions"
       targetNamespace=
                  "http://TargetNamespace.com/ccb/singleRoot"
       xmlns:tns="http://TargetNamespace.com/ccb/singleRoot"
       elementFormDefault="qualified"
                    attributeFormDefault="unqualified"
       nxsd:version="NXSD" nxsd:encoding=
                   "cp037" nxsd:byteOrder="bigEndian"
       nxsd:stream="chars">
 <xsd:element name="Root-Element">
  <xsd:complexType>
   <xsd:sequence>
    <!--COBOL declaration : 05 PO-RECORD -->
    <xsd:element name="PO-RECORD"
                        minOccurs="1"
                     maxOccurs="unbounded">
     <xsd:complexType>
```

```
<xsd:sequence>
 <!--COBOL declaration : 10 PO-BUYER-->
 <xsd:element name="PO-BUYER">
  <xsd:complexType>
   <xsd:sequence>
    <!--COBOL declaration : 15 PO-UID PIC 9(7)-->
    <xsd:element name="PO-UID"
                           type="xsd:long"
            nxsd:style="fixedLength"
                           nxsd:padStyle="head"
            nxsd:paddedBy="0" nxsd:length="7"/>
    <!--COBOL declaration :
                     15 PO-NAME PIC X(15)-->
    <xsd:element name="PO-NAME"
                          type="xsd:string"
            nxsd:style="fixedLength"
                           nxsd:padStyle="tail"
            nxsd:paddedBy=" "
                   nxsd:length="15"/>
    <!--COBOL declaration : 15 PO-ADDRESS-->
    <xsd:element name="PO-ADDRESS">
            <xsd:complexType>
            <xsd:sequence>
    <!--COBOL declaration :
                   20 PO-STREET PIC X(15)-->
    <xsd:element name="PO-STREET"
                      type="xsd:string"
            nxsd:style="fixedLength"
            nxsd:padStyle="tail" nxsd:paddedBy=" "
            nxsd:length="15"/>
    <!--COBOL declaration :
                       20 PO-CITY PIC X(10)-->
    <xsd:element name="PO-CITY"
                             type="xsd:string"
            nxsd:style="fixedLength"
            nxsd:padStyle="tail"
                              nxsd:paddedBy=" "
            nxsd:length="10"/>
    <!--COBOL declaration :
                         20 PO-ZIP PIC 9(5)-->
    <xsd:element name="PO-ZIP"
                           type="xsd:long"
            nxsd:style="fixedLength"
            nxsd:padStyle="head"
                         nxsd:paddedBy="0"
            nxsd:length="5"/>
    <!--COBOL declaration :
                     20 PO-STATE PIC X(2)-->
    <xsd:element name="PO-STATE"
                           type="xsd:string"
            nxsd:style="fixedLength"
            nxsd:padStyle="tail"
                        nxsd:paddedBy=" "
            nxsd:length="2"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<!--COBOL declaration : 10 PO-ITEM-->
```

```
<xsd:element name="PO-ITEM">
 <xsd:complexType>
  <xsd:sequence>
   <!--COBOL declaration :
              15 POITEM OCCURS 3 TIMES-->
   <xsd:element name="POITEM" minOccurs="3"
                    maxOccurs="3">
    <xsd:complexType>
     <xsd:sequence>
      <!--COBOL declaration : 20 PO-LINE-ITEM-->
      <xsd:element name="PO-LINE-ITEM">
       <xsd:complexType>
        <xsd:sequence>
         <!--COBOL declaration :
                  25 PO-ITEM-ID PIC 9(3)-->
         <xsd:element name="PO-ITEM-ID"
                              type="xsd:long"
                 nxsd:style="fixedLength"
                 nxsd:padStyle="head"
                 nxsd:paddedBy="0"
                         nxsd:length="3"/>
         <!--COBOL declaration :
               25 PO-ITEM-NAME PIC X(40)-->
         <xsd:element name="PO-ITEM-NAME"
                 type="xsd:string"
                 nxsd:style="fixedLength"
                 nxsd:padStyle="tail"
                 nxsd:paddedBy=" "
                           nxsd:length="40"/>
         <!--COBOL declaration :
                       25 PO-ITEM-QUANTITY
                            PIC 9(2)-->
         <xsd:element name="PO-ITEM-QUANTITY"
                 type="xsd:long"
                 nxsd:style="fixedLength"
                 nxsd:padStyle="head"
                 nxsd:paddedBy="0"
                         nxsd:length="2"/>
         <!--COBOL declaration :
                    25 PO-ITEM-PRICE PIC
                                9(5)V9(2)-->
         <xsd:element name="PO-ITEM-PRICE"
                 type="xsd:decimal"
                 nxsd:style="virtualDecimal"
                 extn:assumeDecimal="5"
                 extn:picSize="7"/>
        </xsd:sequence>
       </xsd:complexType>
      </xsd:element>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
   </xsd:element>
   <!--COBOL declaration : 10 PO-TOTAL PIC 9(7)V9(2)-->
   <xsd:element name="PO-TOTAL" type="xsd:decimal"
           nxsd:style="virtualDecimal"
                 extn:assumeDecimal="7"
           extn:picSize=" "/>
  </xsd:sequence>
```

```
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Variable Length Array

```
05 EMP-RECORD .
   10 EMP-NAME         PIC X(30).
   10 EMP-DIV-NUM      PIC 9(5).
   10 DIV-ENTRY OCCURS 1 TO 50 TIMES
      DEPENDING ON EMP-DIV-NUM.
      20 DIV-CODE             PIC X(30).
```

The generated schema is shown in example below.

Example - The Generated Schema for Variable Length Array

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--Native format was generated from COBOL copybook:D:\work\
jDevProjects\CCB\Copybooks\odo.cpy-->
<xsd:schema xmlns:xsd=
               "http://www.w3.org/2001/XMLSchema"
       xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
       xmlns:extn="http://xmlns.oracle.com/
                       pcbpel/nxsd/extensions"
       targetNamespace=
                      "http://TargetNamespace.com/
                       ccb/varLengthArray"
       xmlns:tns="http://TargetNamespace.com/
                       ccb/varLengthArray"
       elementFormDefault="qualified"
                      attributeFormDefault=
                            "unqualified"
       nxsd:version="NXSD" nxsd:encoding="cp037"
                      nxsd:byteOrder="bigEndian"
       nxsd:stream="chars">
 <xsd:element name="Root-Element">
  <xsd:complexType>
   <xsd:sequence>
    <!--COBOL declaration :05 EMP-RECORD -->
    <xsd:element name="EMP-RECORD"
                minOccurs="1"
                    maxOccurs="unbounded">
     <xsd:annotation>
      <xsd:appinfo>
       <nxsd:variables>
        <nxsd:variable name="DIV-ENTRY_var0"/>
       </nxsd:variables>
      </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
       <xsd:sequence>
        <!--COBOL declaration:10 EMP-NAME PIC X(30)-->
        <xsd:element name="EMP-NAME"
                           type="xsd:string"
                nxsd:style="fixedLength"
                      nxsd:padStyle="tail"
                nxsd:paddedBy=" "
                           nxsd:length="30"/>
```

```
        <!--COBOL declaration:10 EMP-DIV-NUM PIC 9(5)-->
        <xsd:element name="EMP-DIV-NUM"
                            type="xsd:long"
                nxsd:style="fixedLength"
                            nxsd:padStyle="head"
                nxsd:paddedBy="0"
                          nxsd:length="5">
         <xsd:annotation>
          <xsd:appinfo>
           <nxsd:variables>
            <nxsd:assign name="DIV-ENTRY_var0"
                               value="${0}"/>
           </nxsd:variables>
          </xsd:appinfo>
         </xsd:annotation>
        </xsd:element>
       <!--COBOL declaration :10 DIV-ENTRY OCCURS 1 TO 50 TIMES DEPENDING ON
          EMP-DIV-NUM-->
        <xsd:element name="DIV-ENTRY"
                              nxsd:style="array"
                nxsd:arrayLength=
                         "${DIV-ENTRY_var0}"
                               minOccurs="1"
                maxOccurs="50">
         <xsd:complexType>
          <xsd:sequence>
           <!--COBOL declaration : 20 DIV-CODE PIC X(30)-->
           <xsd:element name=
                       "DIV-CODE" type="xsd:string"
                   nxsd:style="fixedLength"
                              nxsd:padStyle="tail"
                   nxsd:paddedBy=" "
                             nxsd:length="30"/>
          </xsd:sequence>
         </xsd:complexType>
        </xsd:element>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Numeric Types

```
01   NUMERIC-FORMATS.
     05    Salary         PIC 9(5) COMP-3.
     05    Rating         PICTURE S9(5).
     05    Age            PIC 9(3) USAGE COMP.
     05    Revenue        PIC 9(3)V9(2).
     05    Growth         PIC S9(3) SIGN IS LEADING.
     05    Computation    COMP-1.
```

The generated schema is shown in the example below.

Example - Generated Schema for Numeric Types

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--Native format was generated from COBOL copybook :
D:\work\jDevProjects\CCB\Copybooks\numeric.cpy-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
      xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
      xmlns:extn="http://xmlns.oracle.com/pcbpel/nxsd/extensions"
      targetNamespace="http://TargetNamespace.com/ccb/numeric"
      xmlns:tns="http://TargetNamespace.com/ccb/numeric"
      elementFormDefault="qualified"
                      attributeFormDefault="unqualified"
      nxsd:version="NXSD" nxsd:encoding="cp037"
                      nxsd:byteOrder="bigEndian"
      nxsd:stream="bytes">
  <xsd:element name="Numerics">
   <xsd:complexType>
    <xsd:sequence>
      <!--COBOL declaration :01 NUMERIC-FORMATS-->
      <xsd:element name="NUMERIC-FORMATS"
                   minOccurs="1" maxOccurs="unbounded">
       <xsd:complexType>
        <xsd:sequence>
         <!--COBOL declaration : 05 Salary PIC 9(5) COMP-3-->
         <xsd:element name="Salary"
                      type="xsd:long" nxsd:style="comp3"
                extn:sign="unticked" extn:picSize="5"/>
         <!--COBOL declaration:05 Rating PICTURE S9(5)-->
         <xsd:element name="Rating"
                             type="xsd:string"
                nxsd:style="signZoned" extn:sign="ticked"
                extn:picSize="5"
                          extn:signPosn="tailUpperNibble"/>
         <!--COBOL declaration:05 Age PIC 9(3) USAGE COMP-->
         <xsd:element name="Age" type="xsd:long"
                                  nxsd:style="comp"
                extn:picSize="3" extn:sign="unticked"/>
         <!--COBOL declaration:05 Revenue PIC 9(3)V9(2)-->
         <xsd:element name="Revenue" type="xsd:decimal"
                nxsd:style="virtualDecimal"
                                  extn:assumeDecimal="3"
                extn:picSize="5"/>
         <!--COBOL declaration : 05 Growth PIC S9(3) SIGN IS LEADING-->
         <xsd:element name="Growth" type="xsd:string"
                nxsd:style="signZoned"
                                 extn:sign="ticked"
                extn:picSize="3"
                           extn:signPosn="headUpperNibble"/>
         <!--COBOL declaration : 05 Computation COMP-1-->
         <xsd:element name="Computation"
                                  type="xsd:float"
                nxsd:style="comp1" extn:sign="ticked"/>
        </xsd:sequence>
       </xsd:complexType>
      </xsd:element>
     </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

In this case, all numeric types follow formats specified according to IBM COBOL formats. If the data file originates from a different system by using different layouts, the generated schema requires modification.

Command Line Tool for Testing NXSD Translator

You might want to test your nXSD schema to ensure that nXSD annotations are correct and that generated XML/native data conforms to your business semantics. If you want to do that currently, you must write a BPEL process with an inbound or outbound File Adapter partner link, or both, configured with the appropriate nXSD schema and test them on the SOA server. This is both time-consuming and error prone.

A simple standalone test client is available that can enable you to verify your nXSD schemas. You can download the test tool jar from http://download.oracle.com/otndocs/test-translator.jar

Prerequisites

Before you use the test client to verify your nXSD schemas, add the following jars in the classpath. These jars (except for `test-translator.jar`) are available as a part of your SOA installation. You must use Java 6 to run the test client.

- `bpm-infra.jar`. This is the nXSD runtime jar available under `$SOA_HOME/soa/modules/oracle.soa.fabric_11.1.1`
- `xmlparserv2.jar`. This is the Oracle XDK library for parsing available under `$FMW_HOME/oracle_common/modules/oracle.xdk_11.1.0`
- `xml.jar` This is the Oracle XDK library for schema validation available under `$FMW_HOME/oracle_common/modules/oracle.xdk_11.1.0`
- `mail.jar` .This is the Java mail API.
- `test-translator.jar`. You must rename the extension from jarr to jar.

Running the Test Tool

Now you can run `java xlator.util.Translate -help` and the usage should be displayed as shown in Figure 6-71. When supply the `-help` option, the tool supplies a list of options and defaults.

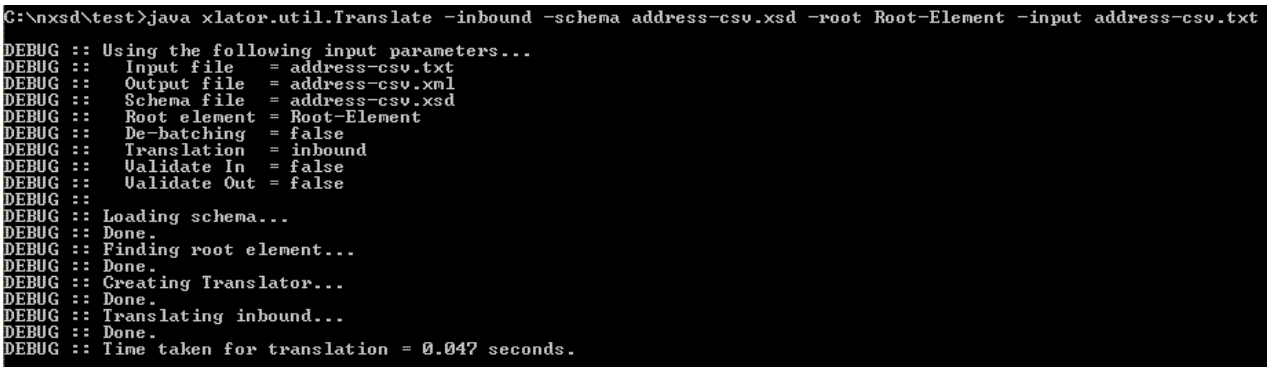
Figure 6-71 Running java xlator.util.Translate -help

```
C:\nxsd\test>java xlator.util.Translate -help

usage: java xlator.util.Translate (options) <Input File>

options:
  -schema <Native Schema>       The schema to use for translation.
  -root <name>                  Localname of the root element declaration to use.
  -inbound  ! -outbound         Do inbound/outbound translation.
  -output <Output File>         The translated file.
  -debatch <Publish Size>       Turn debatching on with the specified publish size.
  -validate (in ! out ! both)   Turn on xml validation of the input/output/both file.
  -debug (on ! off)             Turn on/off debug statements.
  -help                         This help screen.

defaults:
  Validation:  off
  De-batching: off
  Debug: on
```

The following example sample execution of the test client converts the `address-csv.txt` file to address-csv.xml.

The command is:`java xlator.util.Translate -inbound -schema address-csv.xsd -root Root-Element -input address-csv.txt -output address-csv.xml`

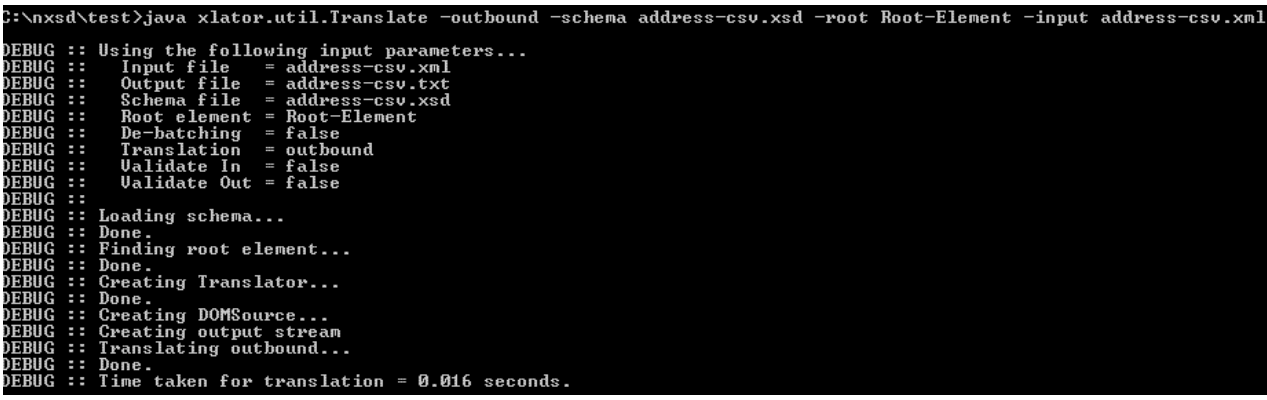
Figure 6-72 Using the Test Tool to Convert txt to xml



Sample execution of the test client to convert `address-csv.xml` to `address-csv.txt`:

```
java xlator.util.Translate -outbound -schema address-csv.xsd -root Root-Element -
input address-csv.xml -output address-csv.txt
```

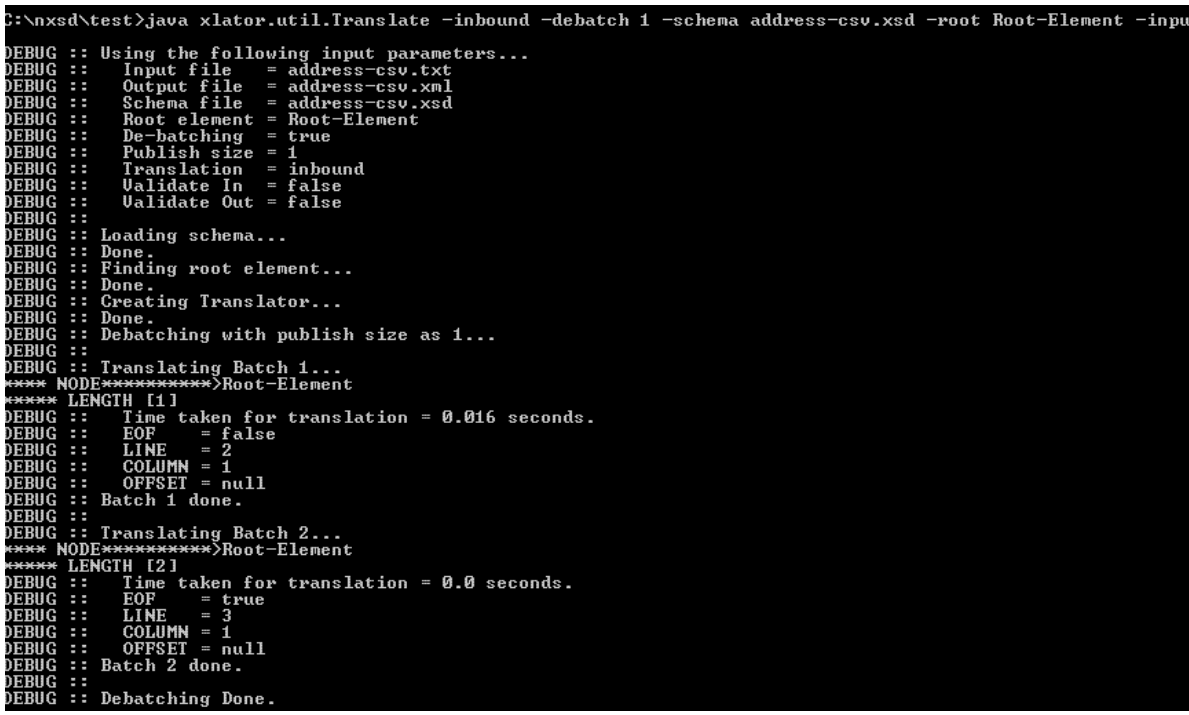
Figure 6-73 Using the Test Tool to Convert xml to txt



The following sample execution of the test client converts `address-csv.txt` to a series of `address-csv.xml_batch_%SEQ%.xml files` using de-batching.

```
java xlator.util.Translate -inbound -debatch 1 -schema address-csv.xsd -root Root-
Element -input address-csv.txt -output address-csv.xml
```

Figure 6-74 Using the Test Tool to Convert address-csv.txt to a Series of batch xml Files



Using the Native Format Builder to Perform MFL Conversion

Oracle WebLogic integration uses a specific XML file format, called MFL format Language, which is a proprietary language that describes native data. MFL is an XML but not a schema file, and MFL cannot be used for SOA transformation/mapping without a an nXSD file.

The MFL conversion feature provides pluggability of MFL in the existing framework, and generates schema corresponding to the MFL format. You can then use the generated schema at runtime.

See the Open Service Bus documentation on MFL, "Defining Data Structures with Message Format Language" in Developing Services with Oracle Service Bus.

Converting an MFL Format File to Schema Format

To accomplish the first step, you tell JDeveloper to convert MFL to NXSD Schema files.

Generating the Schema File and Adding it to the SOA Composite Process

You then use JDeveloper to generate the schema file, which is consumed by the Native Format Builder translator. Because MFL files do not have a target namespace associated with them, you must provide the target namespace for the generated schema file.

If you saved the generated schema file in SOA Project XSD directory when converting the MFL to Schema files, you can skip this step.

If you have generated the schema file in a directory other than that of SOA Project, you must add the schema file generated in Step 1 to the SOA Composite Project's XSD folder.

Once you have added the schema file to the SOA Project, you can use this schema file as you would use any schema file in the project for payload transformation or payload mapping to a variable. The translator runtime uses the attributes defined in the schema file to invoke the MFL Translator at runtime.

Sample MFL File

A sample MFL file is:

```
<?xml version='1.0' encoding='windows-1252'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
<MessageFormat name='Root-Element' version='2.01'>
    <StructFormat name='Address' repeat='*'>
      <FieldFormat name='Name' type='String' delim=',' codepage='windows-1252'/>
      <FieldFormat name='Street1' type='String' delim=','
codepage='windows-1252'/>
    </StructFormat>
</MessageFormat>
```

Sample Schema File Created from the Sample MFL File

The schema file created from the MFL file has two new namespaces:

- `xmlns:mfl="http://www.bea.com/mfl"`
- `xmlns:nxsd=http://xmlns.oracle.com/pcbpel/nxsd`

Additionally, the version of the NXSD translator in the schema is set to MFL: `nxsd:version="MFL"`

A sample schema file produced from the MFL file is shown in the example below.

Example - Sample Schema File Produced from MFL File

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                      xmlns="http://MyTNS1"
targetNamespace="http://MyTNS1"
                     xmlns:mfl="http://www.bea.com/mfl" xmlns:nxsd="http://
xmlns.oracle.com/pcbpel/nxsd"
                   elementFormDefault="qualified"
                     nxsd:version="MFL">
    <xsd:import namespace="http://www.bea.com/mfl"
                     schemaLocation="mfl.xsd"/>
      <xsd:element name="Root-Element">
          <xsd:annotation>
            <xsd:appinfo>
<mfl:MessageFormat encoding="windows-1252"/>
</xsd:appinfo>
</xsd:annotation>
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Address" minOccurs="1"
                       maxOccurs="unbounded">
<xsd:annotation>
<xsd:appinfo>
<mfl:StructFormat repeat="*"/>
</xsd:appinfo>
</xsd:annotation>
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Name" type="mfl:String">
                                <xsd:annotation>
```

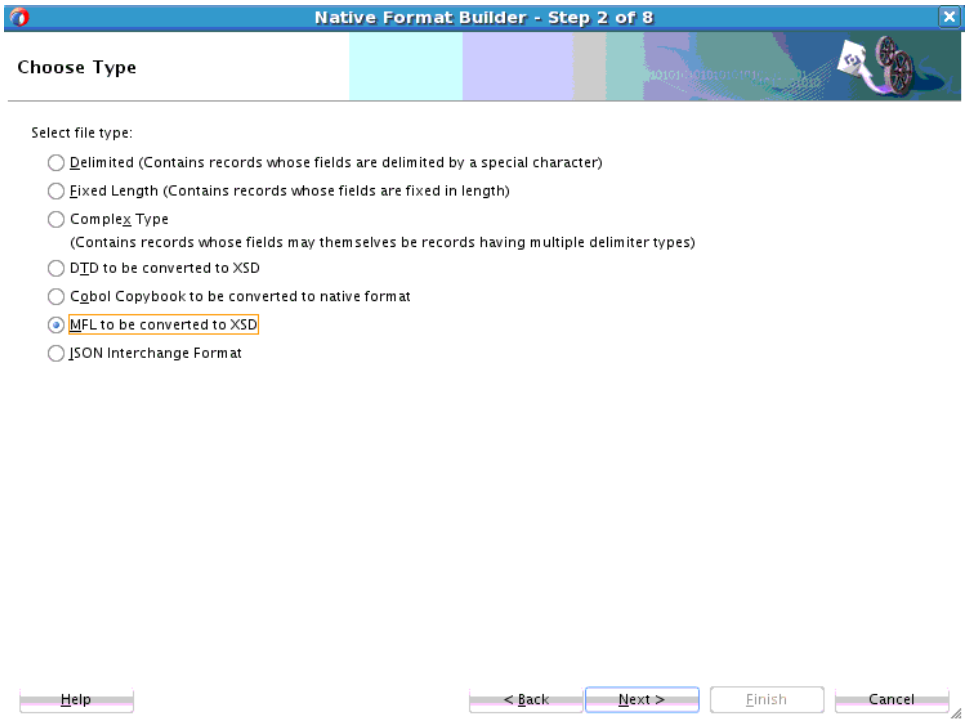
```
                                    <xsd:appinfo>
                                        <mfl:FieldFormat delim=","
                                          codepage="windows-1252"/>
                                    </xsd:appinfo>
                                </xsd:annotation>
                            </xsd:element>
                            <xsd:element name=
                               "Street1" type="mfl:String">
                                <xsd:annotation>
                                    <xsd:appinfo>
                                        <mfl:FieldFormat delim=","
                                         codepage="windows-1252"/>
                                    </xsd:appinfo>
                                </xsd:annotation>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Native Format Builder Wizard Flow for MFL File Conversion

Follow these steps to use the Native Format Builder Wizard to convert MFL files and generate a schema from them.

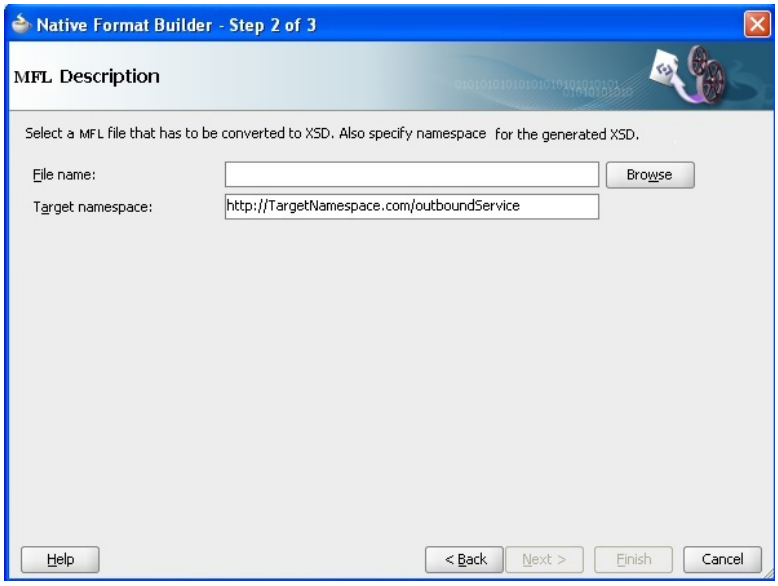
1. On the Choose Type page, select Create New-->> MFL to be created to NXSD.

Figure 6-75 Native Format Builder Choose Type Screen with MFL to be Converted to XSD Selected



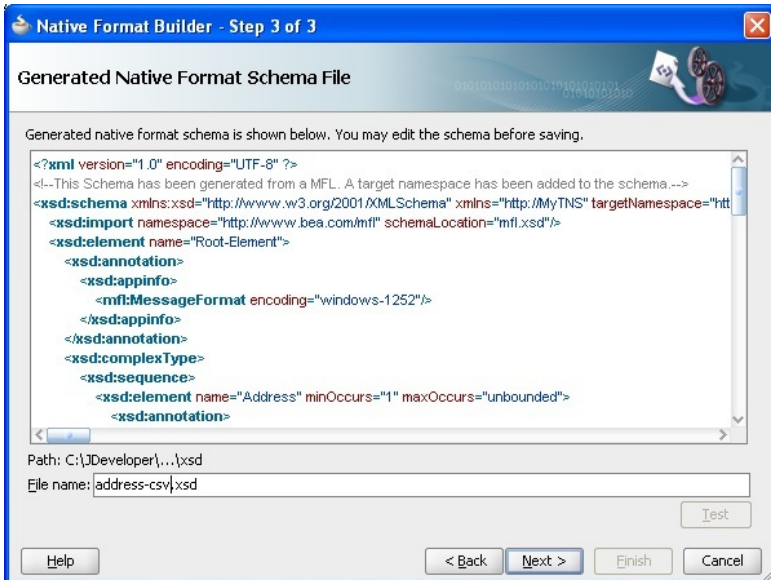
2. The Native Format Builder Wizard displays the **Description Screen**. Provide the location of the MFL file you want converted to XSD and a target namespace you want specified in the schema file. Click **Next**.

Figure 6-76 Native Format Builder Wizard Showing the MFL Description Screen



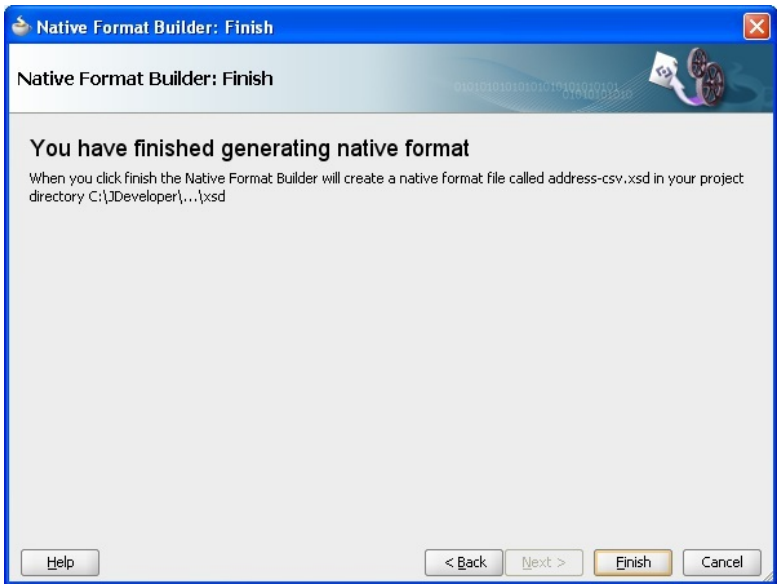
3. Once you have provided the file location and target namespace, the Wizard displays the generated schema file. You can edit the generated schema file, if required, and save it to the location you want, typically the location of the xsd folder in the SOA Composite Project.

Figure 6-77 Native Format Builder Screen Showing the Generated Native Format Schema File



4. Click **Next** to display the **Native Format Builder Finish Screen**, which indicates where the schema file resides.

Figure 6-78 The Native Format Builder Finish Screen



By clicking **Finish** you complete the generation of schema file from the MFL file. The generated schema file is saved at the desired location.

Multi-Character Streaming Support

Often fixed length data length is available in terms of Bytes rather than that of Characters. The field data is in multi-byte character encoding.

In this case, the number of Bytes is not equivalent to the number of Characters in the field. Reading this type of data is not supported by the NXSD Translator.

Currently, the Translator assumes length in terms of Characters only. The Translator uses the Character Reader to read the specified number of Characters.

With support for Multi-Byte Streaming, you can specify the length of a field in terms of Bytes. Translator uses the Byte Reader to read the specified number of Bytes and convert read bytes to characters. These characters are then placed in the generated XML.

Similarly, the outbound Translator converts the XML Data to the specified byte length and write those bytes to the output stream

Shared Delimiters

You often use delimiters when using the Native Format Builder. The behavior of delimiters within the Native Format Builder often depends on the context of their use.

For instance, if a schema document has optional nodes, the existence of the delimiter of their parent could suggest that the native stream does not have data corresponding to these optional nodes. Alternatively, it could be that the delimiter of the parent itself is part of the data for an optional node. It all depends on context.

Similarly, for an outbound scenario, while processing the last field of a group, the NXSD Translator might require hints related to whether placing the last field delimiter followed by group delimiter or simply by a group delimiter would suffice.

With the Shared Delimiter approach, the NXSD Translator addresses such nuances related to the implied meaning of delimiters based on context and provides custom annotations for you to control the meaning of delimiters in context.

Basic Concepts for NXSD Translator with Inbound Shared Delimiter Processing

A shared delimiter indicates that the delimiter marks both the end of the group of data, and the end of the last field of the group. The delimiter is shared between the last field of the group and the end of the group.

Terminating the Cell and the Array

Within the NXSD framework, arrays are used to define repeating structures. `nxsd:cellSeparatedBy` construct defines the delimiter for each cell in the array. This delimiter is used to terminate both the cell and last field of the structure if found before the delimiter of the last field of this structure.

The `nxsd:arrayTerminatedBy` construct marks the end of array itself; it is used to terminate the cell and the array if it is found before the cellSeparatedBy delimiter.

Important Terminology

The following terms are important in understanding the concept of shared delimiters:

- **Last non-optional node**: The node is in a sequence structure which is wrapped by an array. This node is *required*, and all trailing nodes after this are *optional*.
- **Trailing optional node**: The node is in a sequence structure which is wrapped by an array. This node is *optional* and all trailing nodes after this node are *optional*.
- **Pending delimiters list**: List containing all delimiters starting from the immediate *parent array delimiter* (which itself could be the last *required* node of an array which has a shared delimiter).
- **Array**: A collection of cells.
- **Array terminator** marks the end of array
- **Cell separator**: marks the end of each cell of the array.
- Each cell can contain a **structure** - typically the structure repeats and that is why it is surrounded by an array.
- Structure can contain **required** and **optional** elements.
- **Delimiters** can be at different levels:
  - **Field Level**
  - **Group Levels**

Sharing Cell Separator and Array Terminators

Either the cell separator or the array terminator or both can be shared with the elements of group. The cell separator defines the sharing boundary of a cell; the array terminator defines the array and specifies if the array contains any trailing optional fields.

You can specify the terminators as being shared with the elements of group with the following two annotations:

- nxsd:cellSeparator="shared" Defines that the **array cell separator** is shared with the last non-optional field delimiter or with any of the trailing optional field delimiters. The default value for this annotation is "nonshared". This means that under shared mode, when the translator encounters a cell separator while processing any last non optional or trailing optional field in an array, the Translator assumes that this field is over and that the array does not contain any trailing optional field. If the field is over, and there is no trailing optional field, the Translator then closes the current field along with the cell.
- `nxsd:arrayTerminator="shared"` The array terminator is shared with the cell separator and with the last non optional field or with any trailing optional field.

  The default value for this attribute is `nonshared`. This means that when the Translator encounters an **array terminator** while processing the last non-optional field or any trailing optional field, the Translator assumes that the current field is over and array do not contains any trailing optional fields.

  If the array is over, and does not contain any trailing optional fields, the Translator closes the current field, current cell and also the array.

Behavior

Shared delimiter behavior also depends on the nature of the cell. There are three main types of behavior, in addition to a miscellaneous type:

- **Terminated**
- **Fixed Length**
- **Surrounded**

Terminated behavior depends on if the trailing optional children have a delimiter, or if one of the parent delimiters is found.

- If trailing optional children are terminated, the Translator reads from the native data looking for both the child terminator and the pending delimiters.
- If the trailing optional child's terminator is found, generate the optional child.
- However, if one of the parent delimiters is found, map the currently read data to the current optional child, generate it in xml, ignore all optional trailing and close the parent structure till the level the parent delimiter was found.

Fixed length behavior is related to a trailing optional child being fixed length, or if a pending delimiter exists.

- If the trailing optional child is fixed length, before processing the child, look for pending delimiters.
- If one of the parent delimiters is found, ignore trailing optional and close the structures till the parent delimiters structure.
- If pending delimiter does not exist, the Translator reads fixed length number of characters corresponding to this optional node, bypasses pending delimiters if they come as part of the fixed length, generates optional node in XML.

Note:

The NXSD Translator does not handle the scenarios where the shared delimiter is actually present at the start of the fixed length data (that is, fixed length data cannot be escaped as quoted)

Surrounded behavior depends on if a child is surrounded, the behavior of the surroundedBy delimiter, and the existence of a parent delimiter.

- If trailing optional child is surrounded, before processing the child, look for pending delimiters.
- If a pending delimiter does not exist, read the surrounded native data corresponding to this optional node, bypass the pending delimiters if they come before the `surroundedBy` delimiter, and generate an optional node in XML. Repeat this logic for the next trailing optional node.
- If one of the parent delimiters is found, ignore the trailing optional and close the structures until the parent delimiters structure is determined.

  Note that when the comma is used as both shared cell and array separator in NXSD schema, it results in first and only element in the output.

  For example, Input native data is `Field1,Field2,Field3,Field4,Field5` and

  `cellSeparatedBy=','` and `arrayTerminatedBy=','` and both are shared, so the result is only first Field. For example,

```
<sharedDelim xmlns="http://xmlns.mydomain.com/pcbpel/nxsd/test">
    <Group1>
       <Group2>
          <Field1>Field1</Field1>
       </Group2>
    </Group1>
 </sharedDelim>
```

Terminated Use Cases

The use cases in this section relate to terminated behavior: whether trailing optional children are terminated, if trailing optional child's terminator is found, and if one of the parent delimiters is found.

Refer to Figure 6-79 for an illustration of the schema for the use cases, and to Table 6-11 for a description of the use cases. The description includes a list of the fields

Figure 6-79 Schema for Terminated Use Cases

Table 6-11 Terminated Use Cases

Num	Name	Use Case Description	Output

1	`Terminated_example_1.xsd`	Field1#Field2$@Field11#Field22$^Field3%!
		--- 	
		2 cells of group2 followed by field3 comprise one and only cell of group1.
			`<sharedDelim>`
			` <Group1>`
			`  <Group2>`
			
			`<Field1>Field1</Field1>`
			
			`<Field2>Field2</Field2>`
			`  </Group2>`
			`  <Group2>`
			
			`<Field1>Field11</Field1>`
			
			`<Field2>Field22</Field2>`
			`  </Group2>`
			`  <Field3>Field3</Field3>`
			`  </Group1>`
			`</sharedDelim>`

Table 6-11 (Cont.) Terminated Use Cases

Num	Name	Use Case Description	Output

2	`Terminated_example_2.xsd`	Field1@Field11#Field22$^Field3%!
		--- 	
		2 cells of group2. In the first, optional Field2 is missing, in the second, optional Field2 is present. Along with Field3 comprise one cell of array Group1
			`<sharedDelim>`
			`<Group1>`
			`<Group2>`
			`<Field1>Field1</Field1>`
			`</Group2>`
			`<Group2>`
			`<Field1>Field11</Field1>`
			`<Field2>Field22</Field2>`
			`</Group2>`

3	`Terminated_example_3.xsd`	Field1@Field11#Field22!
		--- 	
		2 cells of group2. In the first, optional Field2 is missing, in the second, optional Field2 is present. Optional Field3 is missing and this comprises a cell of Group1.
			`<sharedDelim>`
			`<Group1>`
			`<Group2>`
			`<Field1>Field1</Field1>`
			`</Group2>`
			`<Group2>`
			`<Field1>Field11</Field1>`
			`<Field2>Field22</Field2>`
			`</Group2>`
			`</Group1>`
			`</sharedDelim>`

4	Terminated_example_4.xsd	Field1!
		--- 	
		1 cell of Group2 with missing optional Field2 comprises a cell of Group1 with missing optional Field3.
			`<sharedDelim>`
			`<Group1>`
			`<Group2>`
			`<Field1>Field1</Field1> </Group2> </Group1> </sharedDelim>`

Table 6-11 (Cont.) Terminated Use Cases

Num	Name	Use Case Description	Output
5	Terminated_example_5.xsd	Field0!Field1#Field2$@Field11#Field22$^Field3%!Field111! --- 3 cells of Group1. First cell has 1 cell of Group2 with missing optional Field2 and missing optional Field3. Second cell has 2 cells of Group2 with both the fields and also contains optional Field3. Third cell again is same as first cell.	<sharedDelim> <Group1> <Group2> <Field1>Field0</Field1> </Group2> </Group1> <Group1> <Group2> <Field1>Field1</Field1> <Field2>Field2</Field2> </Group2> <Group2> <Field1>Field11</Field1> <Field2>Field22</Field2> </Group2> <Field3>Field3</Field3> </Group1> <Group1> <Group2> <Field1>Field111</Field1> </Group2> </Group1> </sharedDelim>

Fixed Length Use Cases

The use cases in this section are all fixed length use cases.

See Figure 6-80 for an illustration of the schema for the use cases and Table 6-12 for a description of the use cases. The description includes a list of the fields

Figure 6-80 Fixed Length Use Cases Schema

Table 6-12 provides the names, use case descriptions and resulting output for fixed length use cases.

Table 6-12 Fixed Length Use Cases

Num	Name	Use Case Description	Output
1	`Fixed_length_example1.xsd`	Field01#Field02@Field11#Field12^! --- Only cell of Group1 has two cells of Group2. Each cell of Group2 has both the required terminated Field1 element and optional fixed length Field2. Optional fixed Field3 is missing group Group1.	`<fo_1> <Group1> <Group2> <Field1>Field01</Field1> <Field2>Field02</Field2> </Group2> <Group2> <Field1>Field11</Field1> <Field2>Field12</Field2> </Group2> </Group1> </fo_1>`
2	`Fixed_length_example2.xsd`	Field01! --- First cell of Group2 has required fixedLength Field01 and missing optional fixedLength Field2. Group1 has missing optional fixedLength Field3.	`<fo_1> <Group1> <Group2> <Field1>Field01</Field1> </Group2> </Group1> </fo_1>`

Table 6-12 (Cont.) Fixed Length Use Cases

Num	Name	Use Case Description	Output
3	`Fixed_length_example3.xsd`	Field01!Field11Field02@Field21Field12^Field03!Field31! --- Group1 has 3 cells. First cell only contains the required fixedLength Field1.Second cell contains two cells of Group2. Each of the two cells contain both the fixedLength fields Field1 and Field2. second cell of Group1 also contains fixedLengthOptional Field3. Third cell of Group1 is same as first cell.	`<fo_1> <Group1> <Group2> <Field1>Field01</Field1> </Group2> </Group1> <Group1> <Group2> <Field1>Field11</Field1> <Field2>Field02</Field2> </Group2> <Group2> <Field1>Field21</Field1> <Field2>Field12</Field2> </Group2> <Field3>Field03</Field3> </Group1> <Group1> <Group2> <Field1>Field31</Field1> </Group2> </Group1> </fo_1>`

Surrounded Use Cases

Surrounded uses cases include those where the Translator looks for ".

If the trailing optional child is surrounded, it looks for pending delimiters, or if the pending delimiter does not exist, it reads the surrounded data corresponding to this optional node, bypass pending delimiters if they come before the `surroundedBy` delimiter, and finally generate the optional node in XML.

See Figure 6-81 for an illustration of the schema for the use cases and Table 6-13.

Figure 6-81 Shared Delimiter Surround Schema

Table 6-13 Surrounded Use Cases

Num	Name	Use Case Description	Output

1	`Surrounded_example_1.xsd`	Field01#Field02@Field11#Field12^[Field03]! --- Group1 has 1 cell comprising of repeating structure Group2 and optional surrounded Field3. Group2 has 2 cells with both the required terminated Field1 and optional surrounded Field2.	`<sharedDelim>`
			`<Group1>`
			`<Group2>`
			
			`<Field1>Field1</Field1>`
			
			`<Field2>Field2</Field2>`
			`</Group2>`
			`<Group2>`
			
			`<Field1>Field11</Field1>`
			
			`<Field2>Field22</Field2>`
			`</Group2>`
			`<Field3>Field3</Field3>`
			`</Group1>`
			`</sharedDelim>`

Table 6-13 (Cont.) Surrounded Use Cases

Num	Name	Use Case Description	Output
2	Surrounded_example_2.xsd	Field01#@Field11#Field12 ^! |--- Group1 as 1 cell comprising of repeating structure Group2 and missing optional surrounded Field3. Group2 has 2 cells. First cell has the required terminated Field1 but misses the optional surrounded Field2. Second cell contains both the fields.	<fo_1> <Group1> <Group2> <Field1>Field01</Field1> </Group2> <Group2> <Field1>Field11</Field1> <Field2>Field12</Field2> </Group2> </Group1> </fo_1>
3	Surrounded_example_3.xsd	Field01! --- Group1 has 1 cell comprising of 1 cell of Group2 comprising of required fixedLength Field1 and missing optional surrounded Field2. Group1 also misses optional surrounded Field2.	<fo_1> <Group1> <Group2> <Field1>Field01</Field1> </Group2> </Group1> </fo_1>

Table 6-13 (Cont.) Surrounded Use Cases

Num	Name	Use Case Description	Output
4	Surrounded_example_4.xsd	Field01!Field11-Field02-@Field21-Field12-^[Field03]!Field31! --- Group1 has 3 cells. First cell has a cell of Group2 which contains only the required FixedLength Field1 and misses the optional surrounded Field2. Second cell of Group1 contains 2 cells of Group2 (both complete) and Field3. Third cell of Group1 is same as First.	<fo_1> <Group1> <Group2> <Field1>Field01</Field1> </Group2> </Group1> <Group1> <Group2> <Field1>Field11</Field1> <Field2>Field02</Field2> </Group2> <Group2> <Field1>Field21</Field1> <Field2>Field12</Field2> </Group2> <Field3>Field03</Field3> </Group1> <Group1> <Group2> <Field1>Field31</Field1> </Group2> </Group1> </fo_1>

Miscellaneous Use Cases

The following use cases are a miscellany of different types of uses of shared delimiters.

Shared_array_Terminator.xsd

The sample data for this shared array Terminator use case is

```
"Smith, John","0161-499-1717"
Fred,"0161-499-1718"#^"Smith, John","0161-499-1717"
#*
```

Following is the XSD for this use case:

Example - Shared Array Terminator Sample XSD

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="arrayTerminator">
  <complexType>
    <sequence>
      <element name="arrayOfMembers" maxOccurs="unbounded"
          nxsd:style="array" nxsd:cellSeparator="nonshared"
nxsd:cellSeparatedBy="^"  nxsd:arrayTerminatedBy="*">
    <complexType>
      <sequence>
       <element name="Member" maxOccurs="unbounded"
            nxsd:style="array" nxsd:arrayTerminatedBy="#"
nxsd:arrayTerminator="shared">
    <complexType>
     <sequence>
      <element name="Name" type="string" nxsd:style="terminated"
nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="Telephone" type="string" nxsd:style="terminated"
nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
     </sequence>
      </complexType>
           </element>
            </sequence>
           </complexType>
      </element>
    </sequence>
  </complexType>
</element>

</schema>
```

The generated XML is

Example - Generated SML for Shared Array Terminator Sample

```
<arrayTerminator xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <arrayOfMembers>
      <Member>
         <Name>Smith, John</Name>
         <Telephone>0161-499-1717</Telephone>
      </Member>
      <Member>
         <Name>Fred</Name>
         <Telephone>0161-499-1718</Telephone>
      </Member>
   </arrayOfMembers>
   <arrayOfMembers>
      <Member>
         <Name>Smith, John</Name>
         <Telephone>0161-499-1717</Telephone>
      </Member>
   </arrayOfMembers>
</arrayTerminator>
```

Shared_trailing_array.xsd

The sample data for `shared.trailingArray.xsd` is

```
"Smith, John","0161-499-1717"
Fred,"0161-499-1718"^"Smith, John","0161-499-1717"
#*
```

The sample NXSD schema for this use case is shown below.

Example - Shared Trailing Array Sample NXSD Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="trailingArray">
  <complexType>
    <sequence>
       <element name="arrayOfMembers" maxOccurs="unbounded"
                                              nxsd:style="array"
nxsd:cellSeparator="shared" nxsd:cellSeparatedBy="^"  nxsd:arrayTerminatedBy="*">
      <complexType>
       <sequence>
        <element name="Member" maxOccurs="unbounded"
                    nxsd:style="array"
nxsd:arrayTerminatedBy="#"                      nxsd:arrayTerminator="shared">
       <complexType>
        <sequence>
        <element name="Name type="string" nxsd:style="terminated"
nxsd:terminatedBy="," nxsd:quotedBy='"'/>
        <element name="Telephone" type="string"
nxsd:style="terminated"              nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'
minOccurs="0"/>
    </sequence>
    </complexType>
     </element>
       </sequence>
           </complexType>
       </element>
    </sequence>
  </complexType>
</element>
</schema>
```

The generated XML for this use case is shown in the example below.

Example - Generated XML for Shared Trailing Array Use Case

```
<trailingArray xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <arrayOfMembers>
      <Member>
         <Name>Smith, John</Name>
         <Telephone>0161-499-1717</Telephone>
      </Member>
      <Member>
```

```
         <Name>Fred</Name>
         <Telephone>0161-499-1718</Telephone>
      </Member>
   </arrayOfMembers>
   <arrayOfMembers>
      <Member>
         <Name>Smith, John</Name>
         <Telephone>0161-499-1717</Telephone>
      </Member>
   </arrayOfMembers>
</trailingArray>
</element>
```

Shared_trailing_OptionalArray.xsd Use Case

This shared trailingOptionalArray use case sample data is given in the following examples:

Example - Shared Trailing Optional Array Use Case Sample Data

```
"Smith, John","0161-499-1717"
Fred^"Smith, John","0161-499-1717"#*
```

Example - Shared Trailing Optional Array Use Case NXSD

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace=
         "http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="trailingOptionalArray">
  <complexType>
    <sequence>
        <element name="arrayOfMembers"
                       maxOccurs="unbounded"
      nxsd:style="array" nxsd:cellSeparator=
                "shared" nxsd:cellSeparatedBy="^"
nxsd:arrayTerminatedBy="*">
          <complexType>
            <sequence>
                <element name="Member"
                             maxOccurs="unbounded"
                  nxsd:style="array"
                  nxsd:arrayTerminatedBy="#"
                  nxsd:arrayTerminator="shared">
                    <complexType>
                       <sequence>
                       <element name=
                         "Name" type="string"
                    nxsd:style="terminated"
                    nxsd:terminatedBy=","
                    nxsd:quotedBy='"'/>
                       <element name="Telephone"
                       type="string"
                       nxsd:style="terminated"
                       nxsd:terminatedBy="${eol}"
nxsd:quotedBy='"' minOccurs="0"/>
```

```
                      </sequence>
                    </complexType>
                     </element>
                     </sequence>
                      </complexType>
                </element>
    </sequence>
  </complexType>
</element>
</schema>
```

Example - Generated XML for the Shared Trailing Optional Array Use Case

```
 <trailingOptionalArray xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <arrayOfMembers>
      <Member>
         <Name>Smith, John</Name>
         <Telephone>0161-499-1717</Telephone>
      </Member>
      <Member>
         <Name>Fred</Name>
      </Member>
   </arrayOfMembers>
   <arrayOfMembers>
      <Member>
         <Name>Smith, John</Name>
         <Telephone>0161-499-1717</Telephone>
      </Member>
   </arrayOfMembers>
</trailingOptionalArray>
```

Part II

Message Adapters

Part II contains the following chapters:

7

Oracle JCA Adapter for AQ

This chapter describes how to use the Oracle JCA Adapter for AQ (Oracle AQ Adapter), which enables an Oracle BPEL Process Manager (Oracle BPEL PM) or an Oracle Mediator to interact with a single consumer or a multi consumer queue.

This chapter includes the following sections:

Introduction to the Oracle AQ Adapter

Oracle Streams Advanced Queuing (AQ) provides a flexible mechanism for bidirectional, asynchronous communication between participating applications. Advanced queues are an Oracle database feature, and are therefore scalable and reliable. Other features of Oracle database, such as backup and recovery (including any-point-in-time recovery), logging, transactional services, and system management, are also inherited by advanced queues. Multiple queues can also service a single application, partitioning messages in a variety of ways and providing another level of scalability through load balancing.

This section includes the following sections:

For more information on Oracle AQ, see “Introduction to Oracle Streams AQ" in the Oracle Streams Advanced Queuing User's Guide.

Oracle AQ Adapter Integration with Oracle BPEL Process Manager and Oracle Mediator

JCA Binding Component is used for the bidirectional integration of the JCA 1.5 resource adapters with Oracle BPEL Process Manager and Oracle Mediator. JCA Binding Component is based on standards and employs the Web service Invocation Framework (WSIF) technology for exposing the underlying JCA interactions as Web services.

For more information about Oracle AQ Adapter architecture, adapter integration with Oracle BPEL Process Manager and Oracle Mediator, and adapter deployments, see Adapter Integration with Components .

Oracle AQ Adapter Integration with Oracle Mediator

The Mediator Server supports Oracle AQ Adapter and enables you to define inbound and outbound adapter services for each. An inbound adapter service receives data from an Oracle AQ Adapter and transforms it into an XML message. An outbound adapter service sends data to a target application by transforming an XML message into the native format of the given adapter.

Using the Mediator Server, you can send or receive messages from Oracle Advanced Queuing single or multi consumer queues.

Note:

Oracle BPEL PM pre-dates Mediator and most of this guide and the samples implicitly assume use with Oracle BPEL PM. However, the Oracle AQ Adapter works equally well with either Oracle BPEL PM or Mediator. For any mention of Oracle BPEL PM here, you may substitute Mediator, instead.

Oracle AQ Adapter Features

The Oracle AQ Adapter is both a producer and a consumer of AQ messages. The enqueue operation is exposed as a JCA outbound interaction. The dequeue operation is exposed as a JCA inbound interaction.

The Oracle AQ Adapter supports ADT (Oracle object type), `XMLType`, and `RAW` queues as payloads. It also supports extracting a payload from one ADT member column.

The Oracle AQ Adapter supports normalized properties for enqueue and dequeue operations.

For more information about the properties supported by Oracle AQ Adapter, see Oracle AQ Adapter Properties.

You can obtain the Oracle AQ Adapter samples by accessing the Oracle SOA Sample Code site.

This section includes the following topics:

- Enqueue-Specific Features (Message Production)
- Dequeue and Enqueue Features
- Synchronous Request-Response
- Synchronous Dequeue
- Supported ADT Payload Types
- Native Format Builder Wizard
- Normalized Message Support
- Is DOM 2 Compliant
- Is Message-Size Aware
- Multiple Receiver Threads

- DequeueTimeout Property
- Control Dequeue Timeout and Multiple Inbound Polling Threads
- Stream Payload Support
- Inbound Retries
- Error Handling Support
- Performance Tuning

Enqueue-Specific Features (Message Production)

The Oracle AQ Adapter supports the following features of Oracle Streams AQ:

- Correlation Identifier

  In the Adapter Configuration Wizard, you can specify a correlation identifier when defining an enqueue operation, which you use to retrieve specific messages.

- Multi consumer Queue

  In Oracle Streams AQ, multiple consumers can process and consume a single message. To use this feature, you must create multi consumer queues and enqueue the messages into these queues. In this configuration, a single message can be consumed by multiple AQ consumer (dequeue operation), either through the default subscription list or with an override recipient list. Under this scenario, a message remains in the queue until it is consumed by all of its intended consumer agents. The Oracle AQ Adapter enqueue header property (`jca.aq.RecipientList`) enables you to specify the override recipient list (string values separated by commas) that can retrieve messages from a queue. All consumers that are added as subscribers to a multi consumer queue must have unique values for the `Recipient` parameter. Two subscribers cannot have the same values for the `NAME`, `ADDRESS`, and `PROTOCOL` attributes.

- Message Priority

  If you specify the priority of enqueued messages, then the messages are dequeued in priority order. If two messages have the same priority, then the order in which they are dequeued is determined by the enqueue time. You can also create a first-in, first-out (FIFO) priority queue by specifying the enqueue time priority as the sort order of the messages. This priority is a property of the Oracle AQ Adapter enqueue header. The enqueue time is set automatically by the underlying AQ application.

  Here is an example of how to create the FIFO queue:

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE( \
queue_table => 'OE_orders_pr_mqtab', \
sort_list =>'priority,enq_time', \
comment => 'Order Entry Priority \
MultiConsumer Orders queue table',\
multiple_consumers => TRUE, \
queue_payload_type => 'BOLADM.order_typ', \
compatible => '8.1', \
primary_instance => 2, \
secondary_instance => 1);
EXECUTE DBMS_AQADM.CREATE_QUEUE ( \
queue_name => 'OE_bookedorders_que', \
queue_table => 'OE_orders_pr_mqtab');
```

- Time Specification and Scheduling

  In Oracle Streams AQ, you can specify a delay interval and an expiration interval. The delay interval determines when an enqueued message is marked as available to the dequeuers after the message is enqueued. When a message is enqueued with a delay time set, the message is marked in a `WAIT` state. Messages in a `WAIT` state are masked from the default dequeue calls. The expiration time property is used to specify an expiration time, and the message is automatically moved to an exception queue if the message is not consumed before its expiration.

Dequeue and Enqueue Features

Oracle Streams AQ provides the following dequeuing options:

- Poll option
- Notification option

The poll option involves processing the messages as they arrive and polling repeatedly for messages. The Oracle AQ Adapter supports a polling mechanism for consuming AQ messages.

The Oracle AQ Adapter supports the following features of Oracle Streams AQ:

- Multi consumer Queue

  The Oracle AQ Adapter can retrieve messages from a multi consumer queue.

- Navigation of Messages for Dequeuing

  Messages do not have to be dequeued in the same order in which they were enqueued. You can use a correlation identifier to specify dequeue order. The Adapter Configuration Wizard defines the correlation ID for the dequeue operation.

- Retries with Delays

  The number of retries is a property of the Oracle AQ Adapter dequeue header. If the number of retries exceeds the limit, then the message is moved to an exception queue that you specify. The exception queue is a property of the Oracle AQ Adapter enqueue header.

- Rule-Based Subscription

  Oracle Streams AQ provides content-based message filtering and subject-based message filtering. A rule defines one or more consumers' interest in subscribing to messages that conform to that rule. For a subject-based rule, you specify a Boolean expression using syntax similar to the `WHERE` clause of a SQL query. This Boolean expression can include conditions on message properties (current priority and correlation ID), user data properties (object payloads only), and functions (as specified in the `WHERE` clause of a SQL query).

- Oracle AQ Adapter Header Properties

  For more information about Oracle AQ Adapter header properties, see Oracle AQ Adapter Properties.

- Dequeue Condition

The Dequeue condition is an advanced queuing product feature that Oracle AQ Adapter uses. If a dequeue condition is specified and no messages meet the specified condition, then no dequeue happens.

A dequeue condition element is a Boolean expression using syntax similar to the `WHERE` clause of a SQL query. This Boolean expression can include conditions on message properties, user object payload data properties, and PL/SQL or SQL functions. Message properties include `priority`, `corrid`, and other columns in the queue table.

When a dequeue is performed from a multi subscriber queue, it is sometimes necessary to screen the messages and accept only those that meet certain conditions. These conditions may concern header information, such as in selecting messages of only priority 1, or some aspect of the message payload, such as in selecting only loan applications above $100,000.

The Message Selector Rule field is displayed in Step 15 if you select a multi subscriber queue. Enter a subscription rule in the form of a Boolean expression using syntax similar to a SQL `WHERE` clause, such as `priority = 1`, or `TAB.USER_DATA.amount > 1000`. The adapter dequeues only those messages for which this Boolean expression is true.

You must select the **Access to non-payload fields also needed** check box to access header information.

When this check box is selected, the generated WSDL file has additional code in the `type` section:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<?binding.jca Inbound_aq.jca?>
<definitions name="Inbound"   targetNamespace="http://xmlns.oracle.
   com/pcbpel/adapter/aq/Inbound/" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/" xmlns:plt="http://
schemas.xmlsoap.org/ws/2003/05/partner-link/" xmlns:obj1="http://
xmlns.oracle.com/xdb/SCOTT" xmlns:imp1="http://www.oracle.com/ipdemo">
    <types>
        <schema attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://xmlns.oracle.com/
pcbpel/adapter/aq/Inbound/" xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/aq/
inbound/" xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT">
            <import namespace="http://xmlns.oracle.com/xdb/SCOTT"
schemaLocation="xsd/SCOTT_MAGAZINE_TYPE.xsd"/>
            <import namespace="http://xmlns.oracle.com/pcbpel/adapter/aq/
inbound/" schemaLocation="xsd/aqAdapterInboundHeader.xsd"/>
            <complexType name="HeaderCType">
                <sequence>
                    <element name="QueueHeader"
                             type="hdr:HeaderType"/>
                    <element name="PayloadHeader"
                        type="obj1:MAGAZINE_TYPE"/>
                </sequence>
            </complexType>
            <element name="Header" type="tns:HeaderCType"/>
        </schema>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://www.oracle.com/ipdemo" schemaLocation="xsd/
simpleMagazine.xsd"/>
        </schema>
    </types>
    <message name="simpleMagazine_msg">
```

```
        <part name="simpleMagazine"
                    element="imp1:simpleMagazine"/>
    </message>
    <message name="Header_msg">
        <part name="Header" element="tns:Header"/>
    </message>
    <portType name="Dequeue_ptt">
        <operation name="Dequeue">
            <input message="tns:simpleMagazine_msg"/>
        </operation>
    </portType>
    <plt:partnerLinkType name="Dequeue_plt">
        <plt:role name="Dequeue_role">
            <plt:portType name="tns:Dequeue_ptt"/>
        </plt:role>
    </plt:partnerLinkType>
</definitions>
```

Note that `PayloadHeader` is the type for the whole ADT of the queue. The payload contains only the chosen payload field. If you selected **Access to non-payload fields also needed**, then the `PayloadHeader` (`.jca.aq.HeaderDocument`) contains the whole ADT (including the payload field, which is also present in the header, but ignored by the adapter.)

For more information about Oracle AQ Adapter architecture, adapter integration with Oracle BPEL Process Manager and Oracle Mediator, and adapter deployments, see Introduction to Oracle JCA Adapters.

Synchronous Request-Response

You can employ the AQ Adapter Configuration Wizard to model a process that enables the Oracle AQ Adapter to be used in a synchronous request-response interaction pattern.

In this scenario, the Oracle AQ Adapter sends a request to the request queue and waits for a response from the reply queue before further execution continues. Internally, the Oracle AQ Adapter uses a new interaction pattern, `AQRequestReplyInteractionSpec`. This interaction spec enables you to configure a request and reply destination name.

A variation enables you to use a temporary destination as part of the reply queue. In turn, the Adapter sets the `AQReplyTo` header to the reply destination.

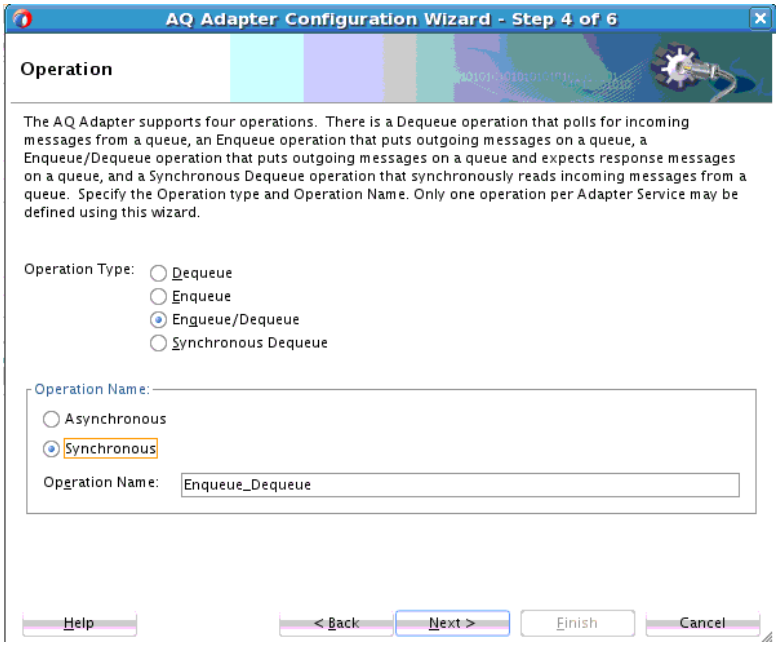
Configuration Wizard Flow for AQ Synchronous Request-Response Interaction Pattern

Follow these steps to use the AQ Adapter Configuration Wizard to configure an AQ Adapter for synchronous request-response.

1. Open the **AQ Adapter Configuration Wizard** and proceed through the initial steps. On the **AQ Adapter Operation Page**, select **Enqueue/Dequeue**. The **Operation Name** section on the page is enabled.

   Select **Synchronous**, and either use the supplied Operation Name `Enqueue_Dequeue`, or change the name to one of your own.

Figure 7-1 AQ Adapter Configuration Wizard Operation Screen with Enqueue/ Dequeue and Synchronous Operation Selected



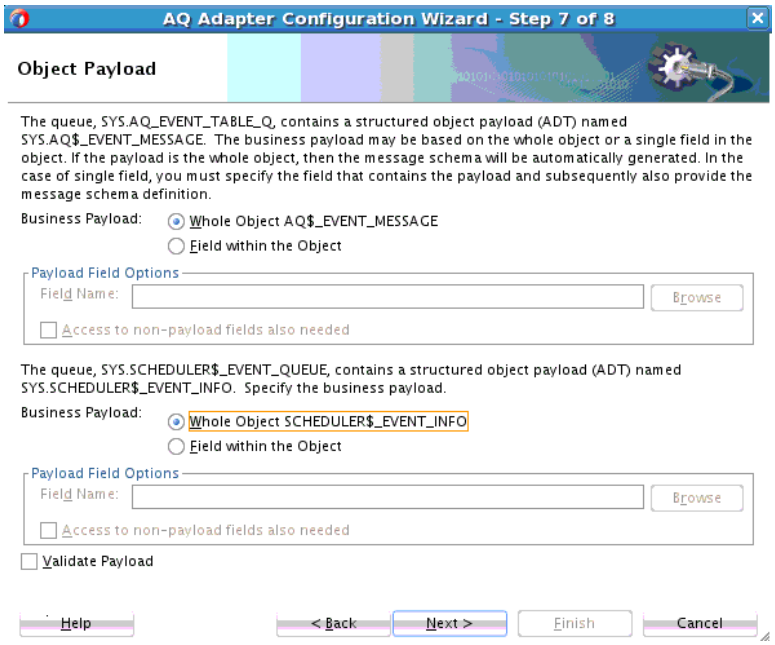
2. On the **Queue Name** screen, provide **Outbound Queue Information** and **Inbound Queue Information**. You can browse for a Database Schema and select the appropriate Queue Name, or use the Default Schema and select a **Queue Name** from the list available with the Default Schema selected.

3. The **Queue Parameters** screen appears. Provide the following, and then select **Next**. Note that

   - In the **Consumer** field, provide the consumer name for a multi-consumer queue.
   - In the **Message Selector Rule** field, provide a Message Selector Rule. Refer to Dequeue and Enqueue Features for more information on using Message Selector Rules.
   - In the Dequeue Condition, provide conditions on messages. Refer to Dequeue and Enqueue Features for more information on using Dequeue Condition.

4. The **AQ Adapter Configuration Wizard Object Payload Screen** appears. This screen enables you to enter information about the object payloads for both inbound and outbound Queues.

Figure 7-2 AQ Adapter Configuration Wizard Object Payload Screen

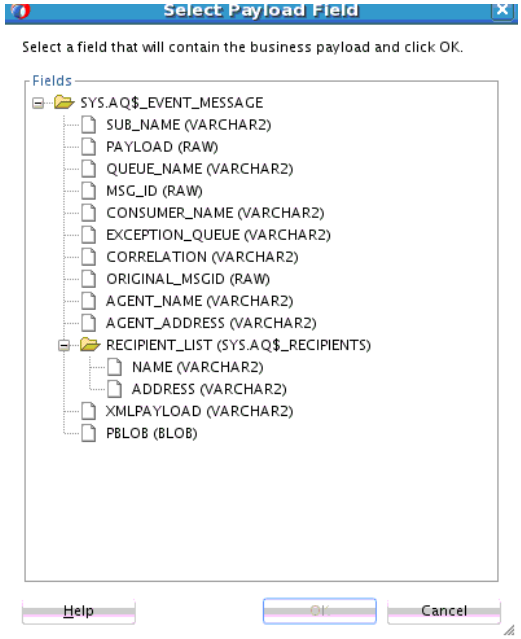


- Text at the top specifies the name of a structured object payload. The rest of the text provides additional information about the payload. For the **Business Payload** either the **Whole object** or the **Field within the Object**.
- If you specified **Field within the Object**, the **Payload Field Options** section is enabled. There you can enter directly or **Browse** to enter the field name.

  If you select Browse, the **Select Payload Field Browser** opens. Select the field or fields that contains the business payload and click **OK** to populate the field name on the **Object Payload** screen.

  You can also select **Validate Payload** to validate the payload for the dequeue operation. When this option is selected, the resulting XML payload is validated against the schema before further processing. In case of failure, the message is rejected.

Figure 7-3 The AQ Adapter Configuration Wizard Select Payload Field Browser



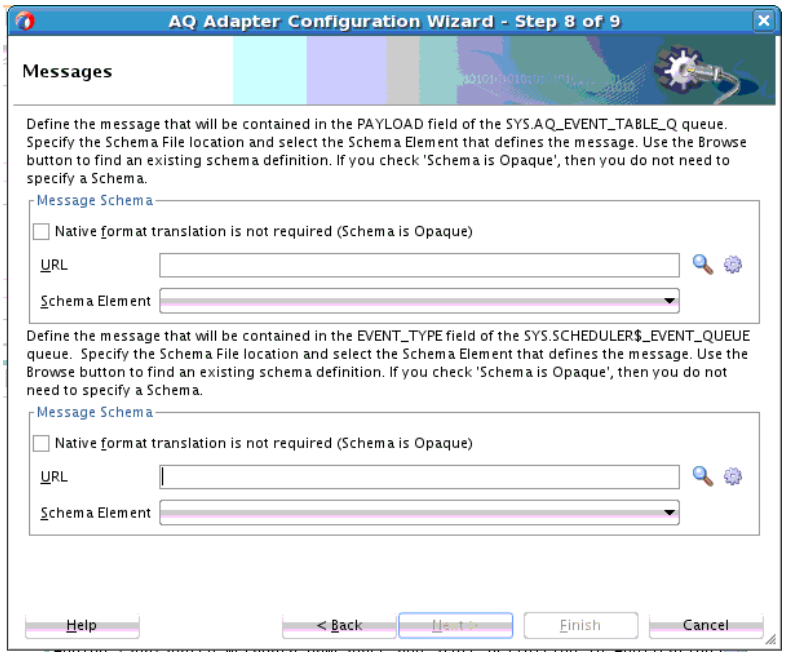
Select the checkbox **Access to non-payload fields needed** if you also need the schema generated for these fields. Select this checkbox if you would want ADT object attributes to be available as header properties. For example, your payload may be a JPG image. You might want to specify a person's name in the non payload field. This selection generates an additional header schema file (`object_name.xsd`, where object_name is the structured payload object used by the queue. The xsd name is additionally prefixed by the schema; if you are connected to schema scott, the name would be `SCOTT_object_name.xsd`. If you select ADT fields on the Object page or if Whole ADT payload is selected, the Messages page is displayed for you to specify schemas.If you specify a ADT Field within the structured object, the outbound ADT field name must be same as the inbound ADT Field name.

- Fill in the information in a similar manner for the Outbound Queue in the second half of the AQ Adapter Configuration Wizard Object Payload Screen.
- Click **Next**.

5. The **AQ Adapter Configuration Wizard Messages Screen** appears.

   - On this screen, you are prompted to define the message that is contained in the fields of both queues you have specified. As with other message screens for this and other adapters, you can indicate if the message is opaque and native format translation is required, or you can specify a schema by providing the schema and the URL and the schema element.
   - Click **Next** to finish configuring the AQ Adapter in a Synchronous Request-Response interaction pattern.

Figure 7-4 The AQ Adapter Configuration Wizard Messages Screen



Editing an AQ Adapter using the Synchronous Request-Reply Interaction Pattern

Note that you can also select a previously-configured AQ Adapter using the SOA Composite Editor and invoking the AQ Adapter wizard in edit mode. However, using this method, you cannot modify the operation type, which you have already defined, but you can modify the selected queue and queue parameters, and modify the payload schema.

Synchronous Dequeue

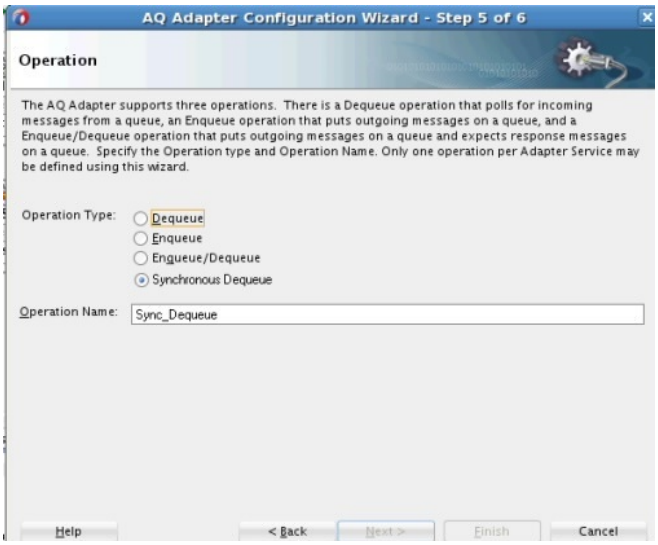
You can configure an outbound Synchronous Dequeue for an AQ Adapter by selecting the **Synchronous Dequeue** operation type on the AQ Operation Type page.

Configuration Wizard Flow for AQ Synchronous Dequeue

Follow these steps to use the AQ Adapter Configuration Wizard to configure AQ for synchronous dequeue.

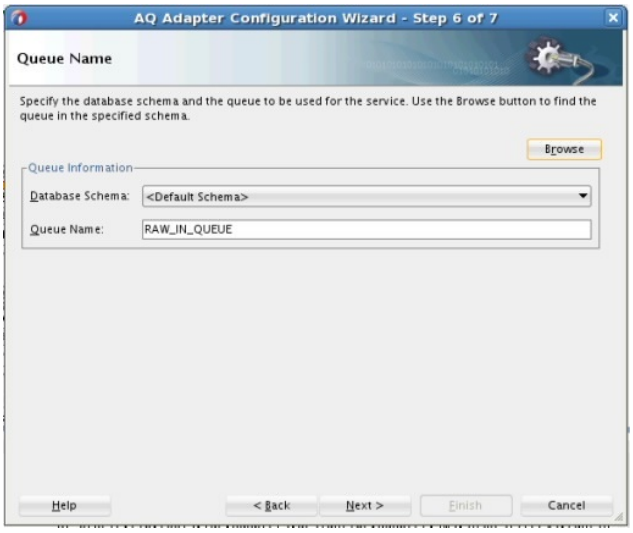
1. Select **Synchronous Dequeue** as the operation type. The operation named for the Synchronous Dequeue is defaulted to **Sync_Dequeue**. However, you can overwrite this name if you wish.

Figure 7-5 AQ Adapter Configuration Wizard Operation Screen with Synchronous Dequeue Operation Selected and Operation Name Sync_Dequeue Shown



2. Click **Next**. The **AQ Adapter Configuration Wizard Queue Name** screen appears. On this screen, you can specify the database schema and the inbound and outbound queue to be used for this service. Click **Next**.

Figure 7-6 AQ Adapter Configuration Wizard Queue Name Screen with Database Schema Defaulted and a Queue Name Selected

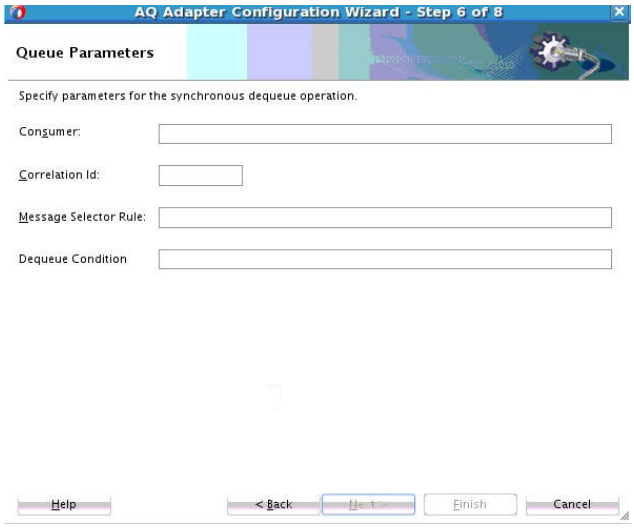


3. On the **Queue Parameters Screen**, indicate the parameters for the **Dequeue** operator: the **Consumer**, the **Message Selector Rule** and the **Dequeue Condition**, and click **Next**. (Note that only if you select a Multi-Consumer Queue, do you get this screen which shows four fields. If we select just a Single-Consumer Queue, you get a screen with just two fields which are Correlation Id and Dequeue Condition. In that case, you do not get Consumer and Message Selector Rule in the Configuration Wizard.

   - **Consumer**: Specify the consumer name for a multi-consumer queue.

- **Correlation ID**: Enter an optional correlation ID from 1 to 30 characters in length.
- **Message Selector Rule**: Provide a Message Selector Rule. Refer to Dequeue and Enqueue Features for more information on using Message Selector Rules.
- **Dequeue Condition**: Displayed only when you select dequeue in the Operation page, this enables you to provide conditions on messages. Refer to Dequeue and Enqueue Features for more information on using Dequeue Condition.

Figure 7-7 The AQ Adapter Configuration Wizard Queue Parameters Screen



JCA File for Synchronous Request-Reply

The next sections provide file artifacts for both synchronous request-reply and synchronous dequeue. The following example shows the jca file for synchronous request-reply scenario:

```
<adapter-config name="AQService" adapter="AQ" wsdlLocation="../WSDLs/
AQRequestReply.wsdl" xmlns="http://platform.integration.oracle/blocks/adapter/fw/
metadata">
  <connection-factory location="eis/AQ/slc01gid" UIConnectionName="slc01gid"/>
  <endpoint-interaction portType="Enqueue_Dequeue_ptt" operation="Enqueue_Dequeue"
UITransmissionPrimitive="Request-response">
    <interaction-spec
className="oracle.tip.adapter.aq.v2.jca.AQRequestReplyInteractionSpec">
     <property name="EnqueueQueue" value="EDN_OAOO_QUEUE"/>
      <property name="RecipientList" value="recp1"/>
      <property name="DequueQueue" value="EDN_EVENT_QUEUE"/>
      <property name="ObjectFieldName" value="PAYLOAD"/>
     <property name="Consumer" value="cons1"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

JCA File for Synchronous Dequeue

The following example shows the jca file for the Synchronous Dequeue scenario:

```
<adapter-config name="AQSyncDequeue" adapter="AQ" wsdlLocation="../WSDLs/
AQSyncDequeue.wsdl" xmlns="http://platform.integration.oracle/blocks/adapter/fw/
metadata">
  <connection-factory location="eis/AQ/slc01gid" UIConnectionName="slc01gid"/>
```

```
  <endpoint-interaction portType="Sync_Dequeue_ptt" operation="Sync_Dequeue">
    <interaction-spec
className="oracle.tip.adapter.aq.v2.jca.AQReceiveNoWaitInteractionSpec">
      <property name="Consumer" value="cons1"/>
      <property name="SchemaValidation" value="false"/>
      <property name="QueueName" value="EDN_OAOO_QUEUE"/>
      <property name="ObjectFieldName" value="PAYLOAD"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

Supported ADT Payload Types

The Oracle AQ Adapter supports the following RAW types:

- BLOB
- CHAR
- CLOB
- DATE
- DECIMAL
- DOUBLE PRECISION
- FLOAT
- INTEGER
- NUMBER
- REAL
- SMALLINT
- TIMESTAMP
- VARCHAR2

In addition to the RAW types mentioned in the preceding list, the Oracle AQ Adapter supports primitive types and arrays of objects.

Note:

The Oracle AQ Adapter does not currently support the following data types for ADT columns: TIMESTAMP WITH LOCAL TIMEZONE and TIMESTAMP WITH TIMEZONE.

If you choose a payload field instead of the whole ADT, then choose one of the following data types as the payload field:

- CLOB, either XSD or opaque schema
- VARCHAR2, either XSD or opaque schema
- BLOB, opaque schema only

- XMLTYPE, either XSD or opaque schema

Native Format Builder Wizard

JDeveloper BPEL Designer provides the Native Format Builder Wizard to define XSD files of various formats, including for the AQ RAW payload.

For more information about the Native Format Builder wizard, see Native Format Builder Wizard .

To obtain sample code that demonstrates usage of the Native Format Builder access the latest SOA Sample code under **Middleware & Tools** from Sample Code for Developers and Adminspage.

Payload Schema

The payload schemas depend on the payload type. In the whole ADT case, the schema is completely generated by the Adapter Configuration Wizard. In an ADT case where the payload case selected is BLOB, an opaque schema as defined in the following example must be used:

```
<element name="opaqueElement" type="base64Binary" />
```

In all other cases, you can either provide a schema or use an opaque schema, as shown in Table 7-1.

Table 7-1 Payload Schema

Payload Type	Supported Schema
RAW	User-provided schema or opaque schema.
Whole ADT	Must use a schema generated by the Adapter Configuration Wizard, which is based on the queue structure.
ADT with VARCHAR2 picked as payload	User-provided schema or opaque schema.
ADT with CLOB picked as payload user-provided schema or opaque schema	User-provided schema or opaque schema.
ADT with BLOB picked as payload opaque schema	Opaque schema.
XMLTYPE	User-provided schema or opaque schema.

If you do not have an XSD file but the payload data is formatted (for example, in a comma-delimited value (CSV) format), you can use the Native Format Builder wizard to generate an appropriate XSD. The Adapter Configuration Wizard is integrated with the Native Format Builder wizard. In the Adapter Configuration Wizard Messages window, click **Define Schema for Native Format** to access the Native Format Builder wizard.

Normalized Message Support

Header manipulation and propagation is a key business integration messaging requirement. Oracle BPEL PM, Mediator, Oracle JCA, and B2B rely extensively on header support to solve customers' integration needs. For example, you can preserve a

file name from the source directory to the target directory by propagating it through message headers. In Oracle BPEL PM and Mediator, you can access, manipulate, and set headers with varying degrees of UI support.

Note:

AQ Adapter inbound and outbound headers supported in the 10.1.3 release are supported in 11*g* through normalized message properties.

For more information, see Correlation Support Within Adapters

Propagating Headers in a Normalized Message:

A normalized message is simplified to have only two parts, properties and payload.Typically, properties are name-value pairs of scalar types. To fit the existing complex headers into properties, properties are flattened into scalar types.

Manipulating Headers in Design Time:

The user experience is simplified while manipulating headers in design time, because the complex properties are predetermined. In the Mediator or BPEL designer, you can manipulate the headers with some reserved key words. For example, currently in Mediator, you can access an inbound File adapter, *fileName* header using the following expression:

```
$nmproperty.InboundFileHeaderType.fileName
```

However, this method does not address the properties that are dynamically generated based on your input. For example, in the AQ Adapter Wizard, you can propagate some fields from an AQ object as headers. Based on your choice, the header definitions are defined. These definitions are not predetermined and hence cannot be accounted for in the list of predetermined property definitions. You cannot design header manipulation of the dynamic properties before they are defined. To address this limitation, you must generate all the necessary services (composite entry points) and references. This restriction applies to services that are expected to generate dynamic properties. Once dynamic properties are generated, they must be stored for each composite. Only then you can manipulate the dynamic properties in Mediator or BPEL designer.

Identifying Properties That Must Be Propagated over the Life Cycle of the Normalized Message:

Some properties must be propagated across the life cycle of the message, whereas some must not be propagated. The properties that must be propagated are referred to as *propagatable* properties, whereas properties that must not be propagated are referred to as *non-propagatable* properties.

Is DOM 2 Compliant

Oracle AQ Adapter is Document Object Model Level 2 (DOM 2) compliant, that is, the AQ adapter can generate document objects that are compliant with DOM2 specification.

Is Message-Size Aware

Oracle AQ Adapter is message-size aware, that is, Oracle AQ Adapter calculates the message size and reports the size back to JCA Binding Component. The API, related to size, exposed by JCA Binding Component can be used for reporting purposes.

Multiple Receiver Threads

Oracle AQ Adapter supports an activation endpoint property, `adapter.aq.dequeue.threads`. Setting this property is a preferred way to spawn multiple threads for the inbound message flow between the adapter and the Enterprise Information System (EIS). Earlier versions of the Oracle AQ Adapter relied on the `activationInstances` endpoint property, which was used by JCA Binding Component to initiate multiple endpoints.

DequeueTimeout Property

The `DequeueTimeOut` property supports multiple inbound dequeue threads. The value for this property determines how many seconds the `dequeue()` API waits for messages before it returns and the next polling cycle begins.

Add this property to the `composite.xml` file, as shown in the following example:

```
<service name="Inbound" ui:wsdlLocation="Inbound.wsdl">
<interface.wsdl interface="http://xmlns.oracle.com/pcbpel/adapter/aq/
AQ_InboundRetry_Mediator/AQ2JMSInboundRetry/Inbound%2F#wsdl.interface(Dequeue_ptt)"/>
<binding.jca config="Inbound_aq.jca">
<property name="DequeueTimeOut" type="xs:integer" many="false"override="may">10
</property>
</binding.jca>
</service>
```

Control Dequeue Timeout and Multiple Inbound Polling Threads

Oracle AQ Adapter provides system properties to control dequeue timeout and multiple inbound polling threads for each Java Virtual Machine (JVM), systemwide, instead of for each process.

The system property provided by Oracle AQ Adapter to control dequeue timeout is `oracle.adapter.aq.wait`, and the property that controls inbound polling threads is `adapter.aq.dequeue.threads`.

Stream Payload Support

Oracle AQ Adapter provides support to stream payload. When you enable this feature, the payload is streamed to a database instead of getting manipulated in SOA runtime as in a memory DOM. You use this feature while handling large payloads. To enable support to stream payload, you must select the Enable Streaming check box while defining the dequeue operation parameters in Oracle JDeveloper (JDeveloper). When you select the Enable Streaming check box, a corresponding Boolean property `EnableStreaming` is appended to the `ActivationSpec` properties defined in the respective `.jca` file, as shown in the following example. If the EnableStreaming property does not exist, then the default value `false` is assumed. The property is applicable when processing `Raw` messages, `XMLType` messages, and `ADT` type messages for which a payload is specified though an `ADT` attribute.

```
<activation-spec className="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec">
      <property name="QueueName" value="RAW_IN_QUEUE"/>
      <property name="DatabaseSchema" value="SCOTT"/>
      <property name="EnableStreaming" value="true"/>
</activation-spec>
```

Oracle AQ Adapter Inbound Retries

If you configure the Oracle AQ Adapter inbound retries to retry for more than 5 times by using the `jca.retry.count` service binding property for a retriable exception, then ensure that the queue is created with max_retries value that is greater then the value used for `jca.retry.count`. If nothing is specified, then the queue is created with a `max_retries` value of 5 which would mean that the message ends up in the exception queue after 5 retries and is not be delivered to adapter for further processing. If `jca.retry.count` is specified with a value of 5 or less, then you do not have to change the queue `max_retries` property.

Use the following code to change the `max_retries` property when creating a queue:

```
begin
DBMS_AQADM.CREATE_QUEUE_TABLE ( queue_table =>
'RAW_IN_QUEUE_TABLE',queue_payload_type => 'RAW');
DBMS_AQADM.CREATE_QUEUE ( queue_name => 'RAW_IN_QUEUE',queue_table=>
'RAW_IN_QUEUE_TABLE', max_retries=>1500);
DBMS_AQADM.START_QUEUE ( queue_name => 'RAW_IN_QUEUE');
DBMS_AQADM.CREATE_QUEUE_TABLE ( queue_table => 'RAW_OUT_QUEUE_TABLE',
queue_payload_type => 'RAW');
DBMS_AQADM.CREATE_QUEUE ( queue_name  => 'RAW_OUT_QUEUE', queue_table =>
'RAW_OUT_QUEUE_TABLE');
DBMS_AQADM.START_QUEUE ( queue_name => 'RAW_OUT_QUEUE');
end;
```

Error Handling Support

For information about error handling, see Error Handling.

Performance Tuning

Oracle AQ Adapter supports performance tuning features.

For more information, see Oracle JCA Adapter Tuning Guide and Oracle JCA Adapter Properties.

Oracle AQ Adapter Deployment

The Oracle AQ Adapter comes deployed to the application server as part of the install. It contains a single adapter instance entry `eis/AQ/aqSample`, which points to the data source `jdbc/aqSample`. The data source is not created as part of install and must be created manually. The connection information to the database is inside the data source definition.

When deploying a SOA project that uses the Oracle AQ Adapter instance `eis/AQ/aqSample` that exists at the time of installation, you must first create a data source at `jdbc/aqSample`. On the other hand, if a new adapter instance is preferred, then you must add a new adapter instance and restart the application server. This is because you want to point to a data source other than the one referred in the existing adapter instance `jdbc/aqSample`, or because you chose a name for the adapter instance that does not yet exist. For instance, if you create a connection in JDeveloper named `DBConnection1`, then by default the AQ Adapter service points to `eis/AQ/DBConnection1`, as shown in Figure 7-13.

You can also check which adapter instance the service is pointing to by looking at the `.jca` file, as shown in the following code snippet:

```
<connection-factory location="eis/AQ/aqSample" … />
```

In the preceding example, the location is the JNDI name of the adapter instance at runtime.

You can create a new AQ Adapter instance through the Oracle WebLogic Administration Console, as mentioned in Adding an Adapter Connection Factory or by directly editing the `weblogic-ra.xml` file. The following are the steps to edit `weblogic-ra.xml`:

1. Search `fmwhome/` for `AqAdapter.rar`.
2. Unzip the file.
3. Edit `META-INF/weblogic-ra.xml` (and possibly `ra.xml`.)
4. Jar the file again.
5. Restart the application server.

The following is a sample AQ adapter instance in `weblogic-ra.xml`:

Example - Sample AQ Adapter Instance in weblogic-ra.xml

```
<connection-instance>
  <jndi-name>eis/AQ/aqSample</jndi-name>
  <connection-properties>
    <properties>
      <property>
         <name>XADataSourceName</name>
         <value>jdbc/aqSample</value>
      </property>
      <property>
    <name>DataSourceName</name>
    <value></value>
      </property>
      <property>
    <name>ConnectionString</name>
    <value></value>
      </property>
      <property>
    <name>UserName</name>
    <value></value>
      </property>
      <property>
    <name>Password</name>
    <value></value>
      </property>
      <property>
    <name>DefaultNChar</name>
    <value>false</value>
      </property>
      <property>
    <name>UseDefaultConnectionManager</name>
    <value>false</value>
      </property>
    </properties>
  </connection-properties>
</connection-instance>
```

The mandatory properties are: `jndi-name`, `XADataSourceName` or `DataSourceName`. The `jndi-name` property must match the location attribute in

the `.jca` file, and is the name of the adapter instance. The `XADataSourceName` or `DataSourceName` property is the name of the underlying data source (which has the connection information). Specify one of the properties `XADataSourceName` or `DataSourceName`. The usage depends on if the scenario involves and would require adapter to participate in global transaction or if local transaction semantics are sufficient. In the former case `XADataSourceName` must be specified while in the latter case `DataSourceName` must be specified. When specifying `XADataSourceName` property ensure that the physical data source it refers to is XA enabled while when specifying `DataSourceName` property the physical data source it refers to might or might not be XA enabled.

Most Common Mistakes

The following are the two most common mistakes with deployment:

- Not creating an adapter instance entry that matches the location attribute in your `.jca` file (or not creating a instance at all.)
- Setting the location attribute in the `.jca` file to the name of the data source directly.

For the latter, there is a level of indirection in that you give the name of the adapter instance (`eis/AQ/...`), which itself points to the data source pool (`jdbc/...`). It is a common mistake to miss this indirection and give the name `jdbc/...` directly in the location attribute.

Additional Adapter Instance Properties

There are additional properties in the AQ Adapter instance beyond `xADataSourceName`, `dataSourceName`.

For information about the Oracle AQ Adapter instance properties, see Oracle AQ Adapter Properties.

Oracle AQ Adapter Use Cases

This section includes the following topics:

- Generic Use Case
- ADT Queue
- RAW Queue

Generic Use Case

The following use cases include a general walkthrough of the Adapter Configuration Wizard, followed by examples of how you can modify the general procedure in different situations. Each example shows relevant parts of the generated WSDL and JCA files.

This section includes the following topics:

- The Adapter Configuration Wizard Walkthrough
- Dequeuing and Enqueuing Object and ADT Payloads
- Dequeuing One Column of the Object Payload
- Configuring the Enqueue/Dequeue Operation Type

- Using Correlation ID for Filtering Messages During Dequeue
- Enqueuing and Dequeuing from Multisubscriber Queues

The Adapter Configuration Wizard Walkthrough

In this example, you create an Oracle AQ Adapter service that dequeues messages to the `service_in_queue` queue, with a payload that is one field within the `service_type` object, and with a user-defined schema.

This section describes the tasks required to configure Oracle AQ Adapter by using the Adapter Configuration Wizard in JDeveloper.

This section includes the following topics:

- Meeting Prerequisites
- Creating an Application and an SOA Project
- Defining an Service
- Generated WSDL and JCA Files

Meeting Prerequisites

This example assumes that you are familiar with basic BPEL constructs, such as activities and partner links, and JDeveloper environment for creating and deploying BPEL composite.

You must have access to a database with the SCOTT schema.

To perform this use case, you require the following files from the `artifacts.zip` file contained in the `adapters-aq-103-adtclobpayload` sample:

- `artifacts/sql/setup_user.sql`
- `artifacts/sql/create_type_service.sql`
- `artifacts/sql/create_queues.sql`
- `artifacts/sql/dequeue_service.sql`
- `artifacts/sql/enqueue_service.sql`

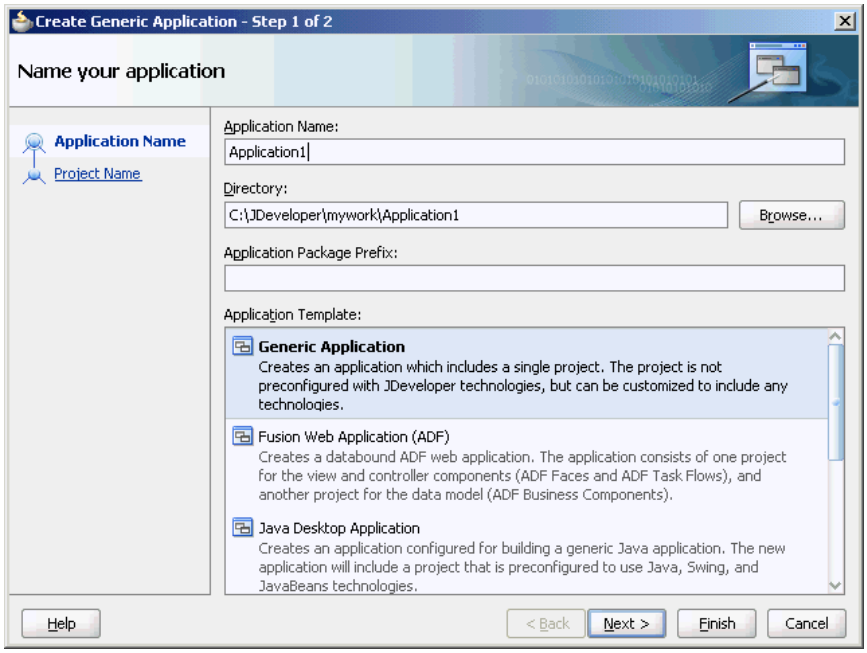
To obtain the `adapters-aq-103-adtclobpayload` sample, access the Oracle Sample SOA Code site.

Creating an Application and an SOA Project

You must create a JDeveloper application to contain the SOA composite. Perform the following steps to create an application, a SOA project:

1. Open JDeveloper.
2. In the **Application Navigator**, click **New Application**. The Create Generic Application Name your application page is displayed, as shown in Figure 7-8.
3. Enter a name for the application in the **Application Name** field.
4. In the **Application Template** list, choose **Generic Application**.

Figure 7-8 The Create Generic Application Name your application Page



5. Click **Next**.

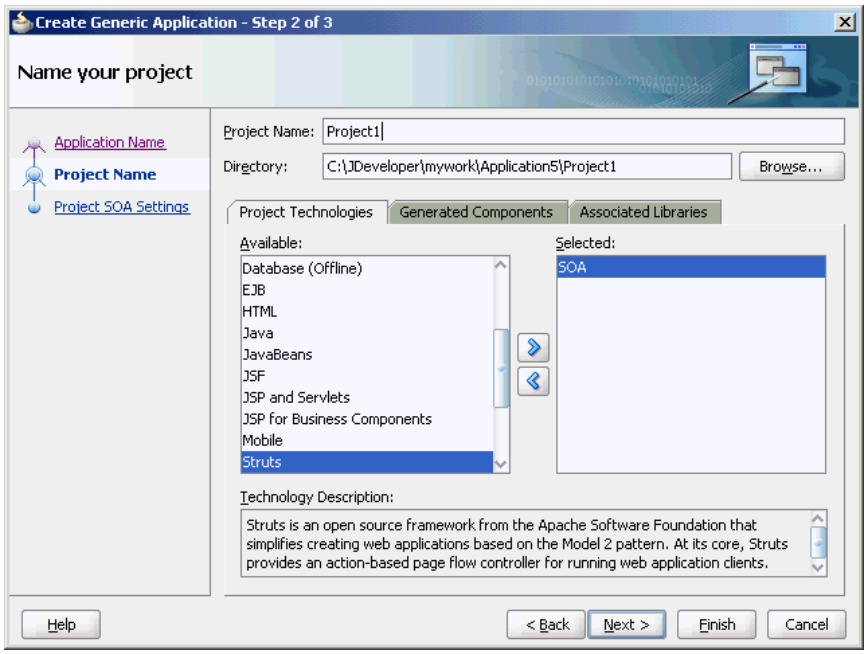
   The Create Generic Application Name your project page is displayed, as shown in Figure 7-9.

6. In the **Project Name** field, enter a descriptive name.

   For example, `SOAComposite`.

7. In the **Available list** in the **Project Technologies** tab, double-click **SOA** to move it to the Selected list.

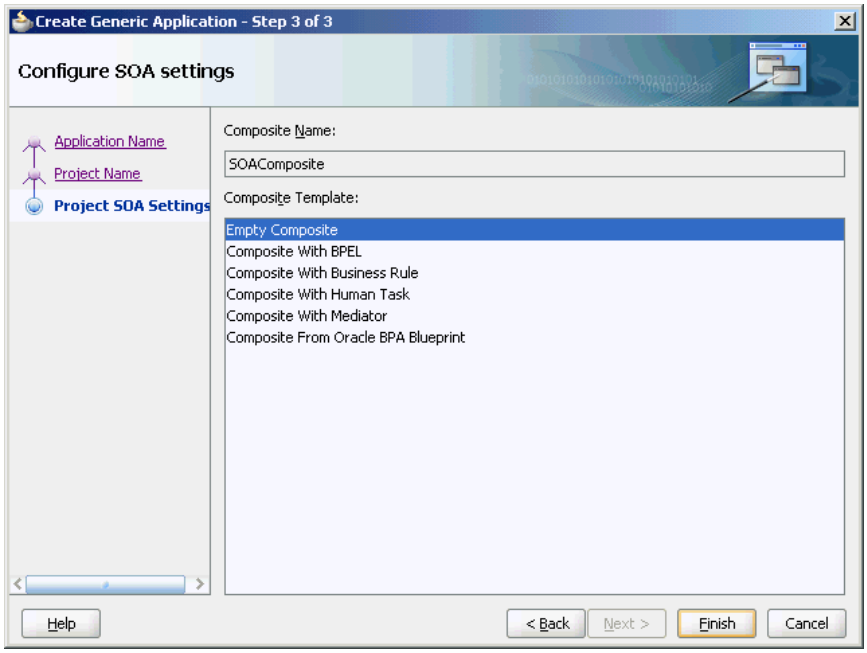
Figure 7-9 The Create Generic Application Name your Generic project Page



8. Click **Next**.

   The Create Generic Application Configure SOA settings page is displayed, as shown in Figure 7-10.

Figure 7-10 The Create Generic Application Configure SOA settings Page

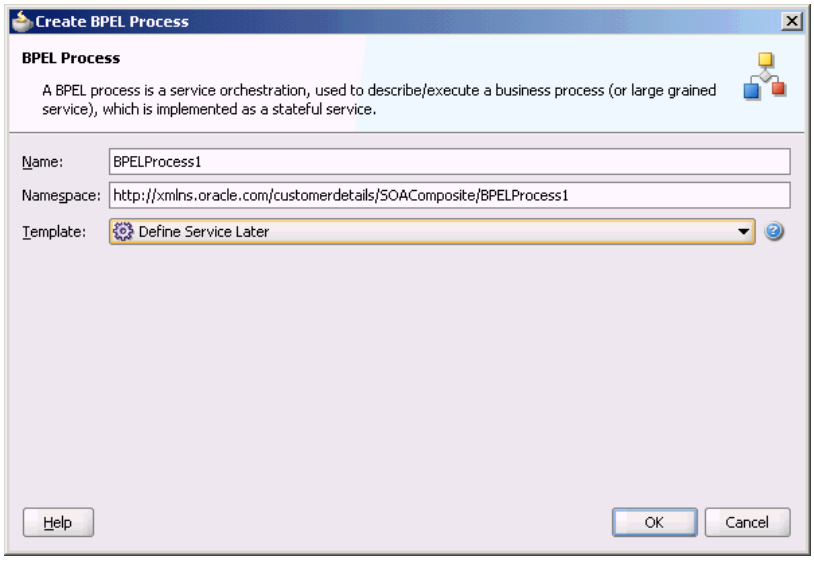


9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

   You have created a new application and an SOA project. This automatically creates an SOA composite.

   The Create BPEL Process page is displayed, as shown in Figure 7-11.

Figure 7-11 The Create BPEL Process Page



10. Enter a name for the BPEL process in the **Name** field. For example, `CustomerDetails`.

11. Select **Define Service Later** in the Template list, and then click **OK**.

    You have created the `CustomerDetails` BPEL process.

Defining an Oracle AQ Adapter Service

The next step is to define an Oracle AQ Adapter service. Perform the following steps to create an Oracle AQ Adapter service:

1. In the Components window, select **SOA**.

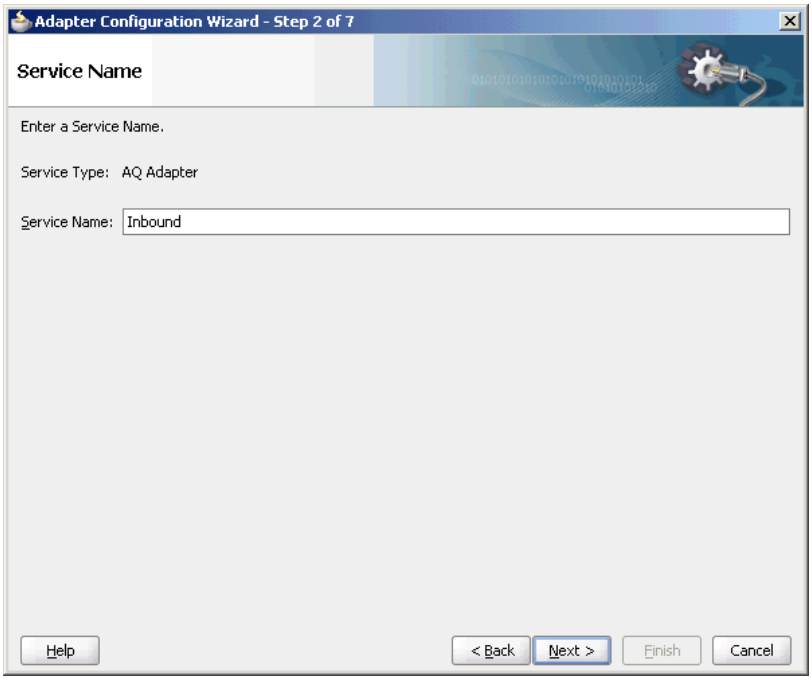
2. Drag and drop **AQ Adapter** from the Service Adapters list to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard Welcome page is displayed.

3. Click **Next**.

   The Adapter Configuration Wizard Service Name page is displayed, as shown in Figure 7-12.

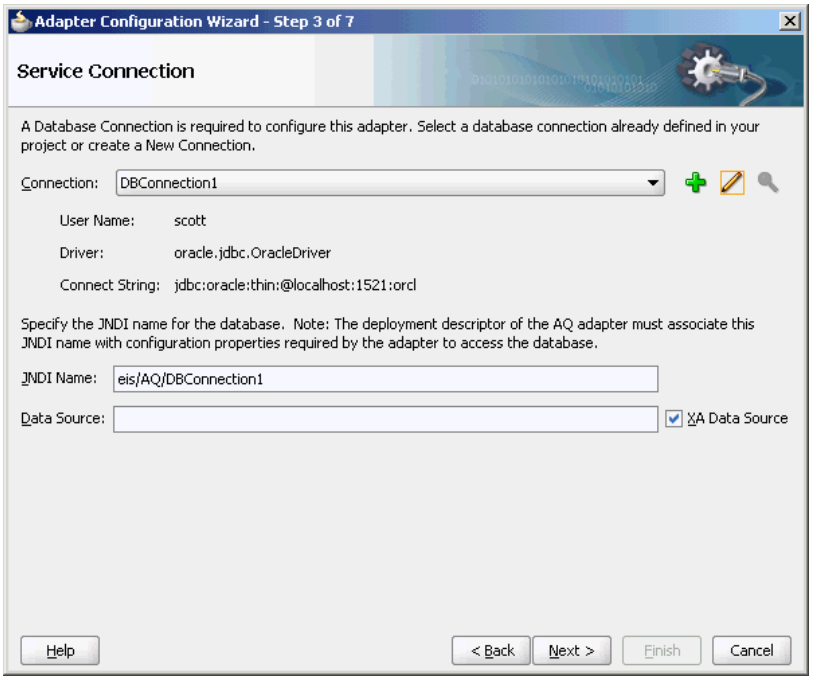
   Figure 7-12 The Adapter Configuration Wizard Service Name Page



4. Specify a service name, and then click **Next**.

   The Adapter Configuration Wizard Service Connection page is displayed, as shown in Figure 7-13.

Figure 7-13 Adapter Configuration Wizard Service Connection Page



5. Click the plus icon to create a database connection.

   The Create Database Connection page is displayed.

   Note:

   You must connect to the database where Oracle Applications is running.

6. Enter the following information:

   a. For **Create Connection In**, choose **Application Resources**.

   b. In the **Connection Name** field, specify a unique name for the database connection.

      In this example, type **DBConnection1**.

   c. From the **Connection Type** box, select **Oracle (JDBC)**.

   d. In the **UserName** field, specify the user name to be authorized for access to the database.

      In this example, type `scott`.

   e. In the **Role** field, enter a role, if applicable.

      This must be a specific database role, such as SYSDBA, as defined in the database. This field is optional. In this example, leave the **Role** field blank.

   f. In the **Password** field, specify the password to be associated with the specified user name.

      In this example, type `tiger`.

   g. Select **Save Password** and **Deploy Password**.

h. From the Driver list, select **Thin**.

i. In the **Host Name** field, enter a value to identify the computer running the Oracle server.

   Use an IP address or a host name that can be resolved by TCP/IP, for example, `myserver`. The default value is `localhost`.

j. In the **JDBC Port** field, enter a value to identify the TCP/IP port. The default is `1521`.

k. In the **SID** field, enter a value for the unique system identifier (SID) of an Oracle database instance.

   The default is `XE`.

l. Click **Test Connection** to determine whether the specified information establishes a connection with the database.

   A Success message is displayed.

m. Click **OK**.

   The Connection you created is displayed in the Connection field in the Service Name page.

   Notice that the Java Naming and Directory Interface (JNDI) name in the JNDI Name field is populated after you have created the database connection. The JNDI name acts as a placeholder for the connection used when your service is deployed to the BPEL server. Using JNDI as a placeholder enables you to use different databases for development and later production.

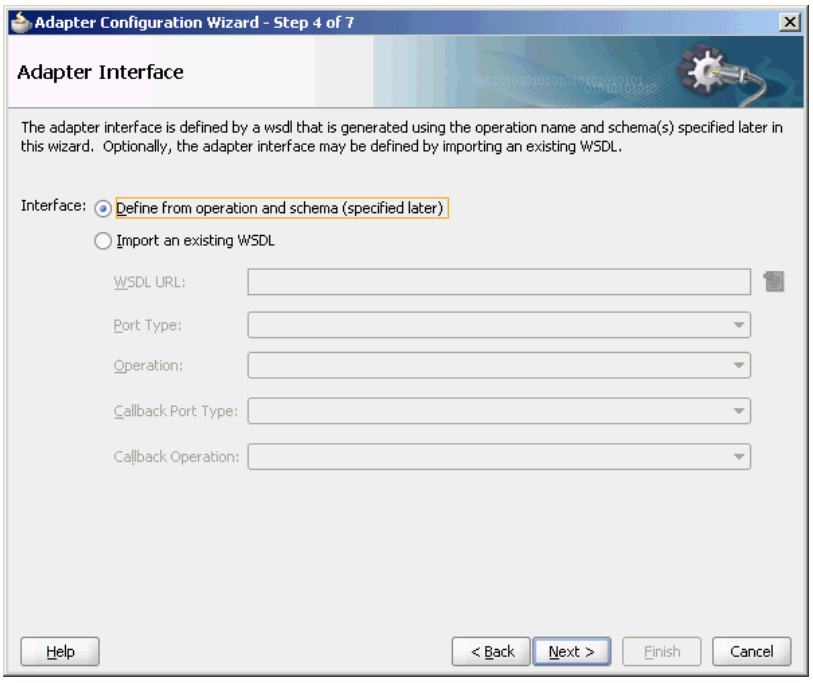
   The value specified in the JNDI name must exist in the Oracle AQ Adapter `weblogic-ra.xml` file to ensure that the adapter runs in managed mode. A default connection instance `eis/AQ/aqSample` is shipped and can be used as the default value for this field. To use this connection instance, it would still require that a data source is created with the JNDI name `jdbc/aqSample`.

7. Click **Next**.

   The Adapter Configuration Wizard Adapter Interface page is displayed, as shown in Figure 7-14.

8. In the Adapter Interface page, choose **Define from operation and schema (specified later)**.

Figure 7-14 The Adapter Configuration Wizard Adapter Interface Page



9. Click **Next**.

   The Operation page is displayed.

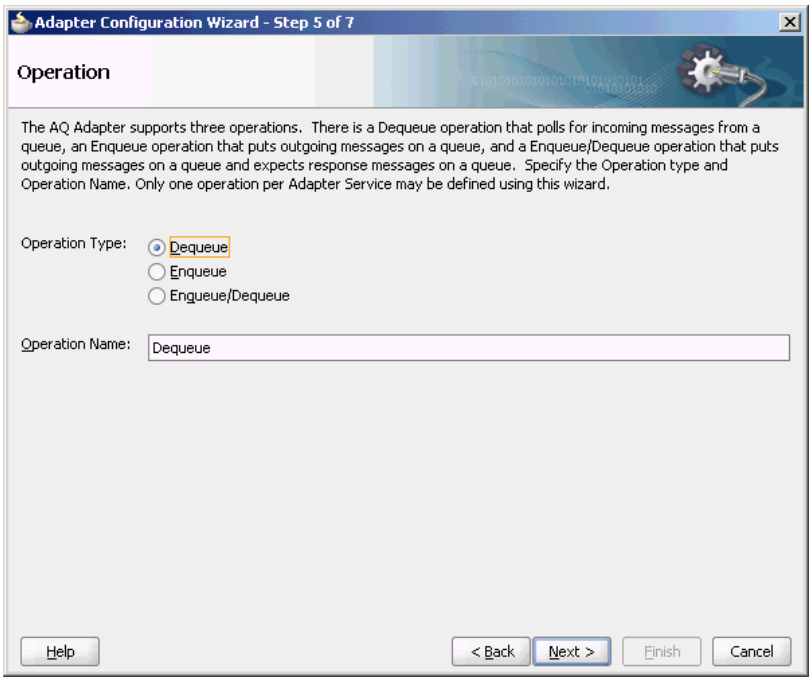
10. Oracle AQ Adapter supports three operations:

    - **Dequeue**: Polls for incoming messages from a queue.
    - **Enqueue**: Puts outgoing messages in a queue.
    - **Enqueue/Dequeue**: Puts outgoing messages in a queue and expects response messages in a queue.

    In this example, select **Dequeue**, as shown in Figure 7-15.

    The operation is automatically named after the operation that you selected. However, you can edit the **Operation Name** field.

Figure 7-15 The Adapter Configuration Wizard Operation Page



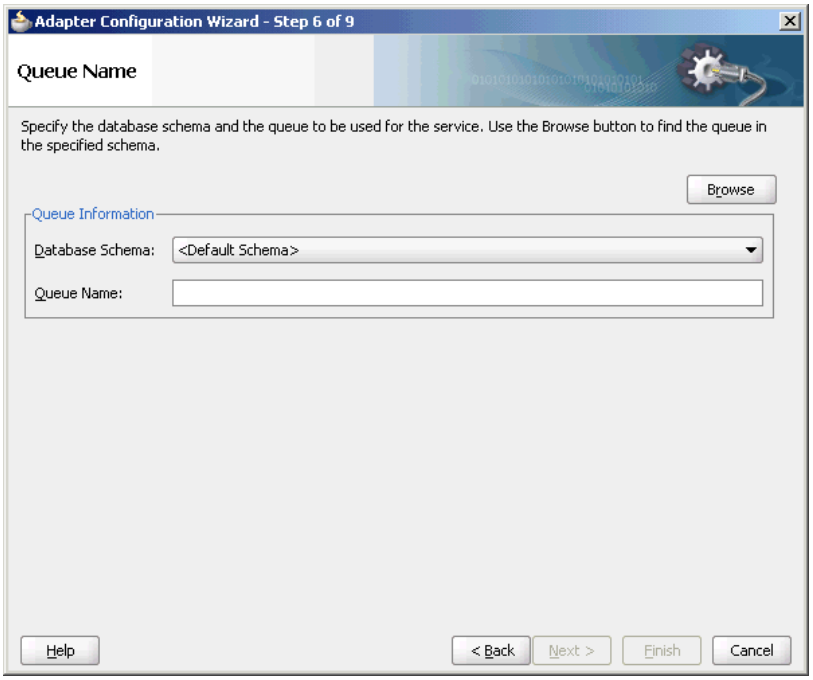
Note:

When creating a SOA composite that uses an inbound Oracle AQ Adapter dequeuing from a queue based on an ADT (Oracle Object) data type - if the SchemaValidation property in the .jca config file is set to `true`, any empty (NULL) members of the ADT payload in the dequeued message results in the `AQ_INVALID_PAYLOAD` error being raised and further resulting in the message being rejected. To avoid the message being rejected, you must set the SchemaValidation property to `false`.

11. Click **Next**.

    The Adapter Configuration Wizard Queue Name page is displayed, as shown in Figure 7-16.

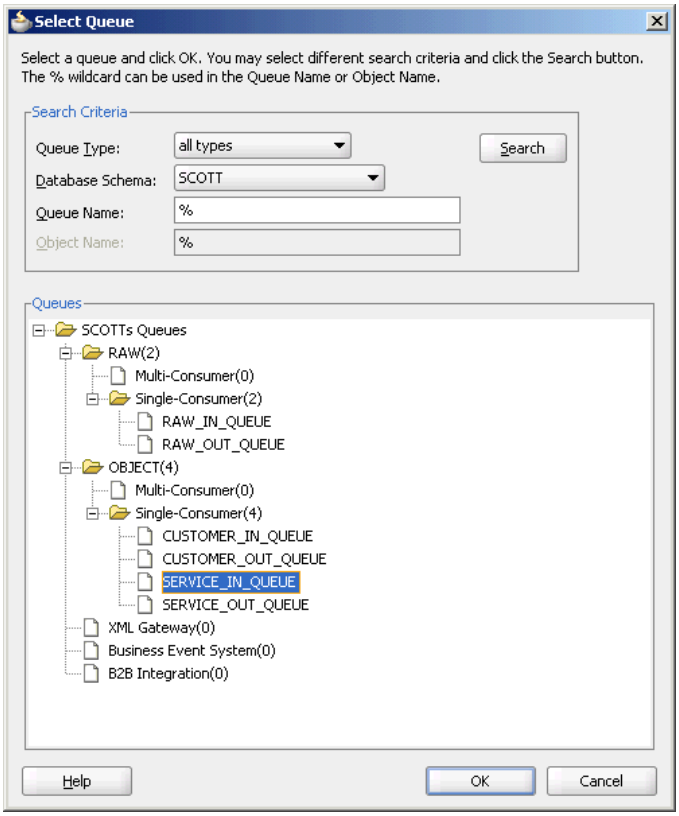
Figure 7-16 The Adapter Configuration Wizard Queue Name Page



12. Select a database schema from the Database Schema list, or click **Browse** to browse for the schema. In this example, click **Browse**.

    The Select Queue dialog is displayed, as shown in Figure 7-17.

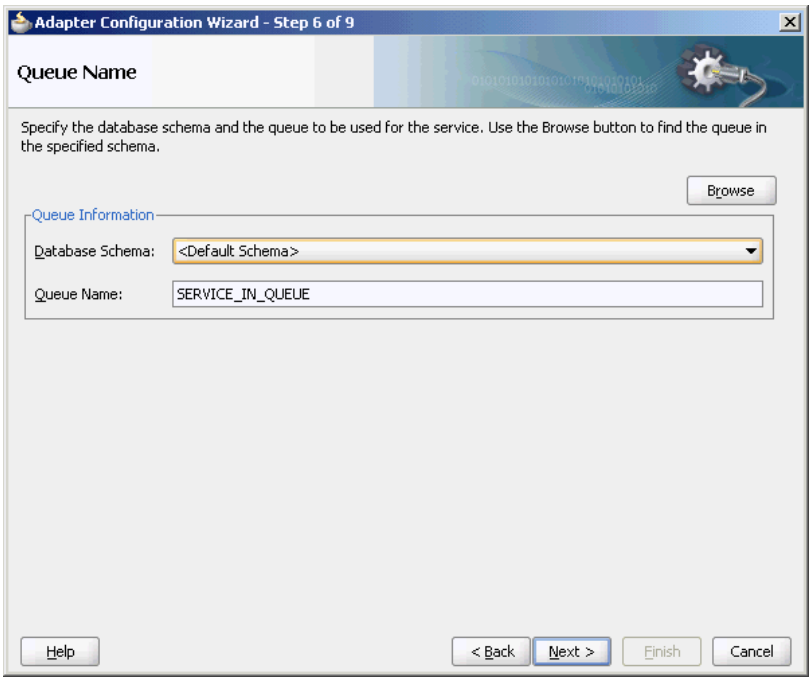
Figure 7-17 The Select Queue Dialog



13. Select the required queue, and then click **OK**.

    In this example, select SERVICE_IN_QUEUE. The Queue Name page is displayed again with the Queue Name field populated with SERVICE_IN_QUEUE, as shown in Figure 7-18.

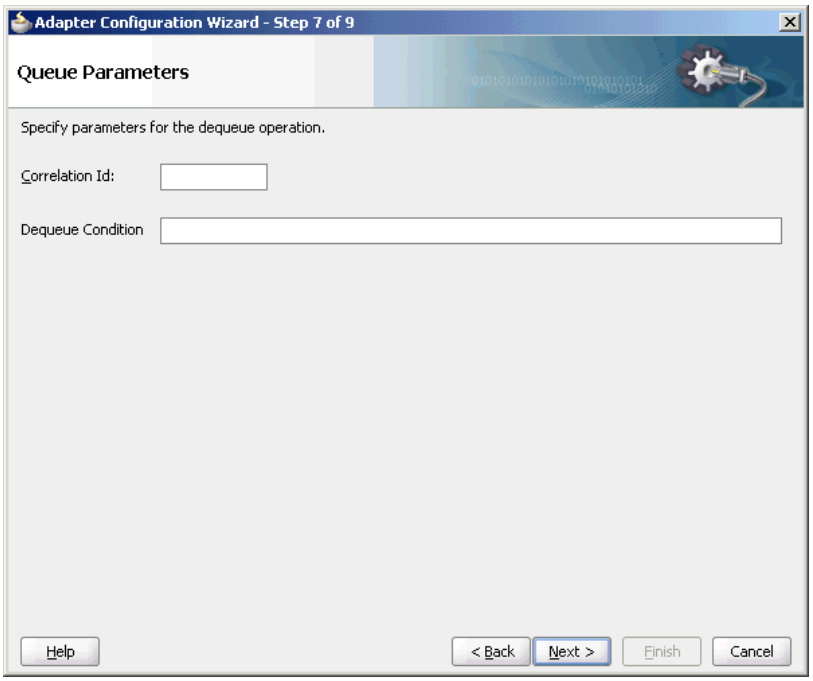
Figure 7-18 The Adapter Configuration Wizard Queue Name Page



14. Click **Next**.

    The Adapter Configuration Wizard Queue Parameters page is displayed, as shown in Figure 7-19.

Figure 7-19 The Adapter Configuration Wizard Queue Parameters Page



15. Enter values for the parameters, and then click **Next**.

    - **Correlation ID**: Enter an optional correlation ID from 1 to 30 characters in length. This is used to identify messages that can be retrieved at a later time by a dequeue activity using the same correlation ID.

      The value to enter is agreed upon between the enqueuing sender and the dequeuing receiver for asynchronous conversations. The correlation ID maps to an AQ header property. Correlation IDs in the inbound direction enable you to be selective about the message to dequeue. This field is optional. If you do not enter a value, then all the messages in the queue are processed.

      If you enter a value for the Correlation ID in the outbound direction, then all outbound messages have the correct ID set to the value entered. You can override this value on a per message basis in the correlation field of the outbound header.

    - **Dequeue Condition**: Displayed only when you select dequeue in the Operation page.

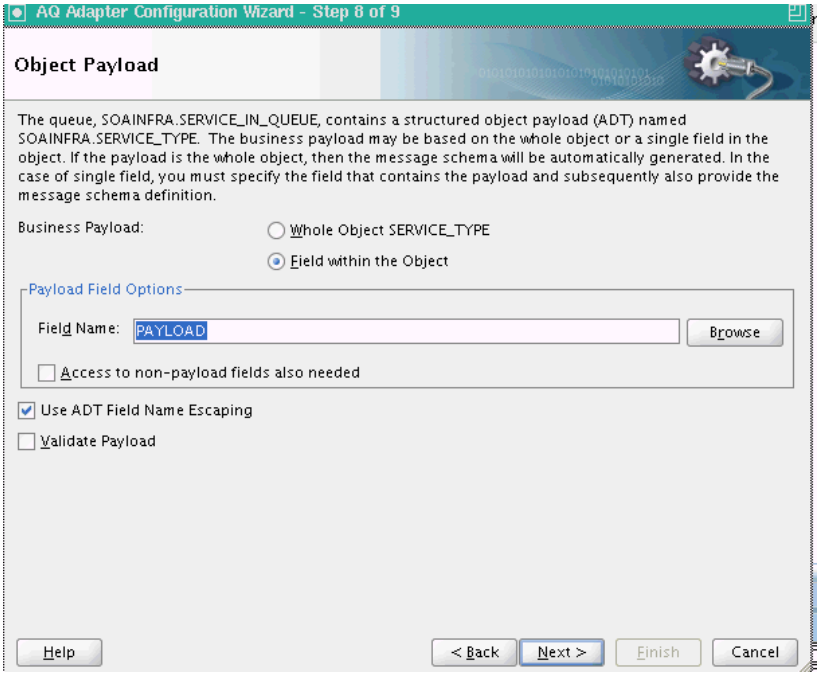
      Enter a Boolean expression similar to the `WHERE` clause of a `SQL` query. This expression can include conditions on message properties, user data properties (`object payloads` only), and PL/SQL or SQL functions. If more than one message satisfies the dequeue condition, then the order of dequeuing is indeterminate, and the sort order of the queue is not honored.

      This field is displayed for inbound single consumer and multi consumer queues.

16. Click **Next**.

    The Adapter Configuration Wizard Object Payload page is displayed, as shown in Figure 7-20.

Figure 7-20 The Adapter Configuration Wizard Object Payload Page

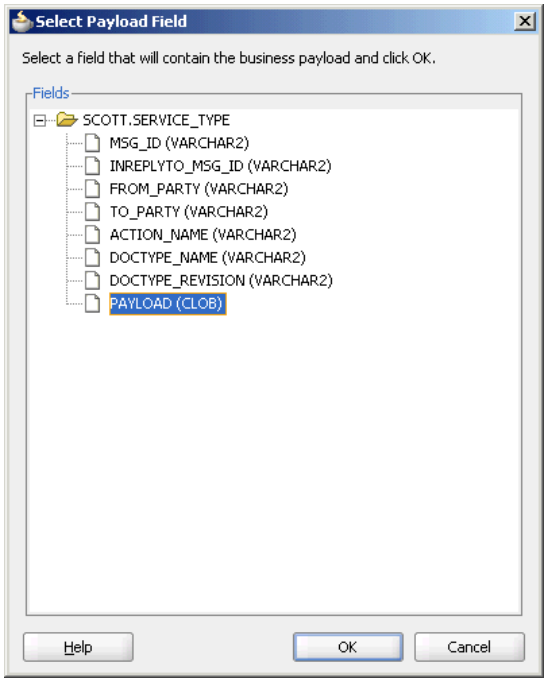


a. In Business Payload, select **Field within the Object**.

b. Click **Browse** in the Payload Fields Options section to select a field that contains the business payload.

The Select Payload Field dialog is displayed, as shown in Figure 7-21.

Figure 7-21 The Select Payload Field Dialog

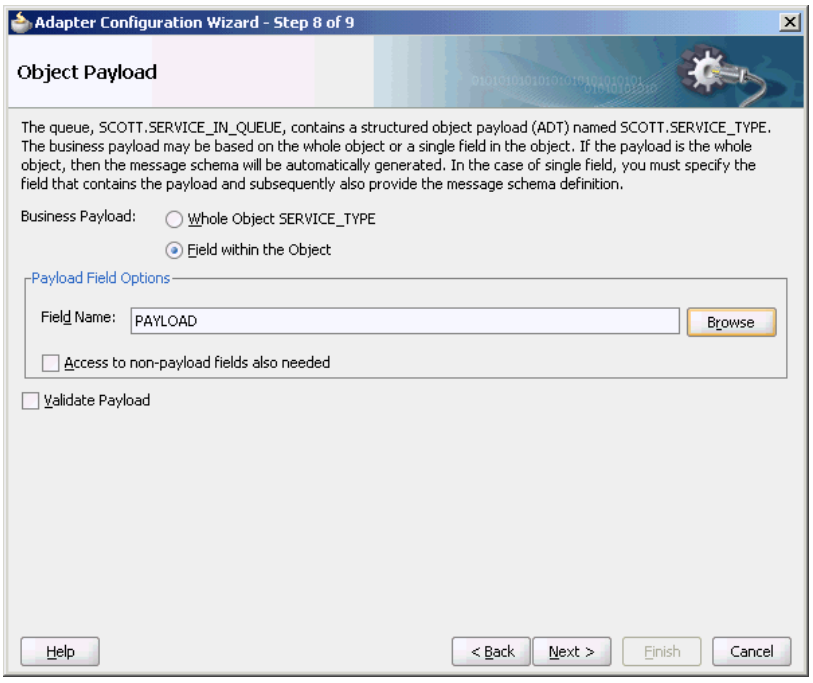


17. Select a field, and then click **OK**.

In this example, select **PAYLOAD (CLOB).**

The Object Payload field is displayed with all the payload details filled up, as shown in Figure 7-22.

Figure 7-22 The Adapter Configuration Wizard Object payload Page



18. Select **Access to non-payload fields also needed**, and then click **Next**.

    The Messages page is displayed.

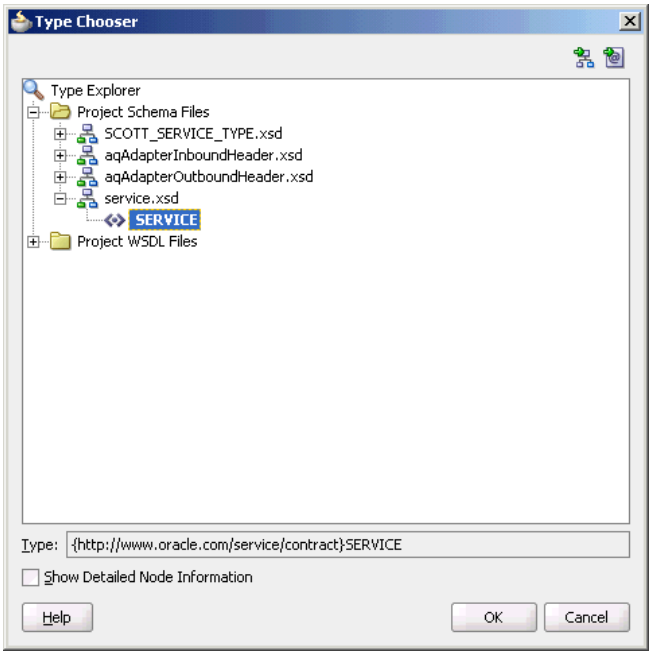
    The Message page has the following options:

    - **Native format translation is not required (Schema is Opaque)**: Select this option if you do not want to specify a schema. Selecting this option disables all the other fields under Message Schema.
    - **Define Schema for Native Format**: Click this to start the Native Format Builder wizard, which guides you through the process of defining the native format.
    - **URL**: You can enter the path for the schema file URL or click **Browse** to browse for the path.
    - **Schema Element**: The name of the schema element.

19. In this example, click the **Browse for schema file** icon to browse for the schema file URL.

    The Type Chooser dialog is displayed, as shown in Figure 7-23.

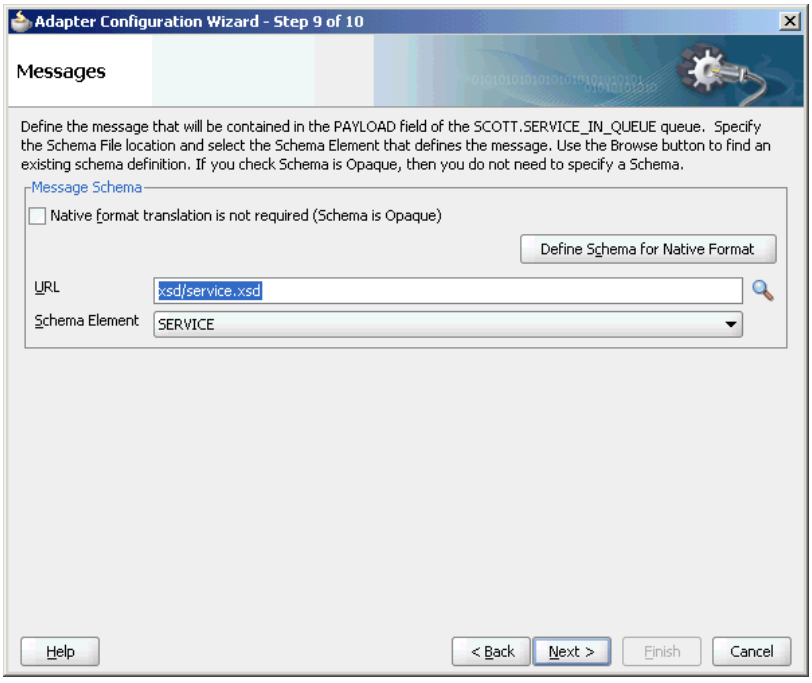
Figure 7-23 The Type Chooser Dialog



20. Select **SERVICE** from the list, as shown in Figure 7-23, and then click **OK**.

    The Messages page reappears, with the Schema Location and Schema Element fields populated, as shown in Figure 7-24.

Figure 7-24 The Adapter Configuration Wizard Messages Page



21. Click **Next**.

    The Finish screen is displayed. This page shows the path and name of the adapter file that the wizard creates.

22. Click **Finish**.

    You have created an AQ Adapter service with dequeue operation.

23. Click **OK**.

Generated WSDL and JCA Files

The adapter service generates a WSDL and a JCA file to serve as the defined adapter interface.

The following is the WSDL file generated for the dequeue operation:

```
<definitions name="Inbound" targetNamespace="http://xmlns.oracle.com/pcbpel/
adapter/aq/Inbound/" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://
xmlns.oracle.com/pcbpel/adapter/aq/Inbound/" xmlns:plt="http://
schemas.xmlsoap.org/ws/2003/05/partner-link/" xmlns:obj1="http://
xmlns.oracle.com/xdb/SCOTT" xmlns:imp1="http://www.oracle.com/service/contract">
    <types>
    <schema attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/" xmlns="http://
www.w3.org/2001/XMLSchema" xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/
Inbound/" xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"
xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT">
       <import namespace="http://xmlns.oracle.com/xdb/SCOTT" schemaLocation="xsd/
SCOTT_SERVICE_TYPE.xsd"/>
        <import namespace="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"
schemaLocation="xsd/aqAdapterInboundHeader.xsd"/>
            <complexType name="HeaderCType">
                <sequence>
                    <element name="QueueHeader" type="hdr:HeaderType"/>
                    <element name="PayloadHeader" type="obj1:SERVICE_TYPE"/>
                </sequence>
            </complexType>
            <element name="Header" type="tns:HeaderCType"/>
        </schema>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://www.oracle.com/service/contract"
schemaLocation="xsd/service.xsd"/>
        </schema>
    </types>
    <message name="SERVICE_msg">
        <part name="SERVICE" element="imp1:SERVICE"/>
    </message>
    <message name="Header_msg">
        <part name="Header" element="tns:Header"/>
    </message>
    <portType name="Dequeue_ptt">
        <operation name="Dequeue">
            <input message="tns:SERVICE_msg"/>
        </operation>
    </portType>
    <plt:partnerLinkType name="Dequeue_plt">
        <plt:role name="Dequeue_role">
            <plt:portType name="tns:Dequeue_ptt"/>
        </plt:role>
    </plt:partnerLinkType>
</definitions>
```

Dequeuing and Enqueuing Object and ADT Payloads

Dequeuing and enqueuing is covered in ADT Queue.

To enqueue or dequeue the entire object as the payload, perform the following:

- Select **Enqueue** or **Dequeue** in Step 10.
- Select **Whole Object CUSTOMER_TYPE**, and skip to Step 16.

For a working example of an ADT payload use case, refer to any of the following samples:

- `adapters-aq-102-adt`
- `adapters-aq-110-supportedadttypes`

You can obtain these samples by accessing the Oracle SOA Sample Code site.

Note:

If you modify an ADT type using evolution commands such as ALTER OBJECT, the AQ Adapter throws an ORA-25215 SQL exception.

The workaround to this exception is to use only CREATE OBJECT (without issuing evolution commands such as ALTER OBJECT) to add attributes to the ADT TYPES.

Dequeuing One Column of the Object Payload

The walkthrough is an example of dequeuing one field or column within an object payload.

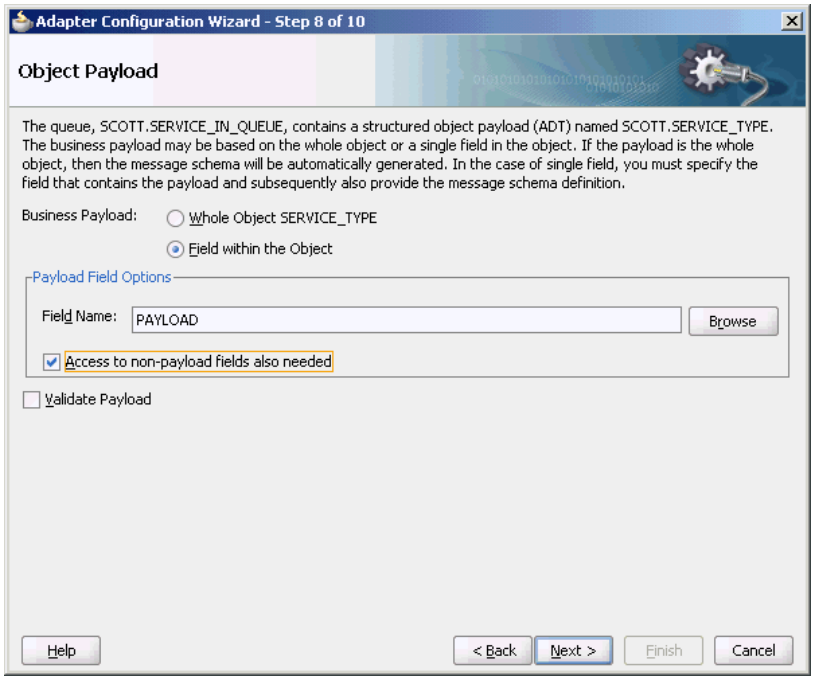
To create an Oracle AQ Adapter that dequeues one field in an object, you must perform the following steps in the Adapter Configuration Wizard Object Payload page:

1. Select **Field within the Object**.
2. Click **Browse** at the end of the **Field Name** field.

   The Select Payload Field dialog is displayed.
3. Select a field that contains the business payload, and then click **OK**.

   The Adapter Configuration Wizard Object Payload page with Field Name field populated with the field that you selected is displayed, as shown in Figure 7-25.

Figure 7-25 The Adapter Configuration Wizard Object Payload Page



4. Select **Access to non-payload fields also needed**, and then click **Next**.

The following segment of the generated JCA file specifies that one field, in this case the field named `PAYLOAD`, is dequeued in addition to payload header fields.

```
<adapter-config name="Inbound" adapter="AQ Adapter" xmlns="http://
platform.integration.oracle/blocks
    /adapter/fw/metadata">
    <connection-factory location="eis/AQ/aqSample"
       UIConnectionName="Connection1" adapterRef=""/>
  <endpoint-activation portType="Dequeue_ptt" operation="Dequeue">
    <activation-spec
className="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec">
      <property name="QueueName" value="SERVICE_IN_QUEUE"/>
      <property name="ObjectFieldName" value="PAYLOAD"/>
      <property name="PayloadHeaderRequired" value="true"/>
      <property name="SchemaValidation" value="false"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

For a working example of an ADT `CLOB` use case where one field or column within an object payload is dequeued, refer to the following samples:

- `adapters-aq-103-adtclobpayload`
- `adapters-aq-105-adtclobopaquepayload`

You can obtain these samples by accessing the Oracle SOA Sample Code site.

Configuring the Enqueue/Dequeue Operation Type

This use case walks you through the procedure for configuring the Enqueue/Dequeue operation type of the Oracle AQ Adapter, which lets the Oracle AQ Adapter put outgoing messages on a queue and expect response messages on a different queue.

This section includes the following topics:

Meeting Prerequisites

To perform this use case, you must have access to a database with the SCOTT schema. Also, you require the following files from the `artifacts.zip` file contained in the `adapters-aq-104-requestreply` sample:

- `create_queues.sql`
- `drop_queues.sql`
- `enqueue.sql`
- `SendReply.sql`
- `setup_user.sql`

To obtain the adapters-aq-104-requestreply sample code, access the Oracle SOA Sample Code site.

Creating an Application and an SOA Project

You must create a JDeveloper application to contain the SOA composite. Follow the steps documented in Creating an Application and an SOA Project to create an application, and an SOA project.

Defining an Oracle AQ Adapter Service

Perform the following steps to create an Oracle AQ Adapter service to put outgoing messages on a queue and expect response messages on a queue:

1. In the Components window, select **SOA**.
2. Drag and drop **AQ Adapter** from the Service Adapters list to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard Welcome page is displayed.
3. Click **Next**.

   The Adapter Configuration Wizard Service Name page is displayed, as shown in Figure 7-12.
4. Specify a service name, and then click **Next**.

   The Adapter Configuration Wizard Service Connection page is displayed.

5. Click the plus icon to create a database connection.

   The Create Database Connection page is displayed.

   Note:

   You must connect to the database where Oracle Applications is running.

6. Enter the following information:

   a. For **Create Connection In**, choose **Application Resources**.

   b. In the **Connection Name** field, specify a unique name for the database connection.

   c. From the **Connection Type** box, select **Oracle (JDBC)**.

   d. In the **UserName** field, specify the user name to be authorized for access to the database.

      In this example, type `scott`.

   e. In the **Role** field, enter a role, if applicable.

      This must be a specific database role, such as `SYSDBA`, as defined in the database. This field is optional. In this example, leave the **Role** field blank.

   f. In the **Password** field, specify the password to be associated with the specified user name.

      In this example, type `tiger`.

   g. Select **Save Password** and **Deploy Password**.

   h. From the Driver list, select **thin**.

   i. In the **Host Name** field, enter a value to identify the computer running the Oracle server.

      Use an IP address or a host name that can be resolved by TCP/IP, for example, `myserver`. The default value is `localhost`.

   j. In the **JDBC Port** field, enter a value to identify the TCP/IP port. The default is `1521`.

   k. In the **SID** field, enter a value for the unique system identifier (SID) of an Oracle database instance.

      The default is `XE`.

   l. Click **Test Connection** to determine whether the specified information establishes a connection with the database.

      A Success message is displayed.

   m. Click **OK**.

      The Connection you created is displayed in the Connection field in the Service Connection page.

      Also, the JNDI Name field is populated after you created the database connection.

The value specified in the JNDI name must exist in the Oracle AQ Adapter `weblogic-ra.xml` file to ensure that the adapter runs in managed mode. A default connection instance `eis/AQ/aqSample` is shipped and can be used as the default value for this field. To use this connection instance, it would still require that a data source is created with the JNDI name `jdbc/aqSample`.

7. Click **Next**.

   The Adapter Configuration Wizard Adapter Interface page is displayed.

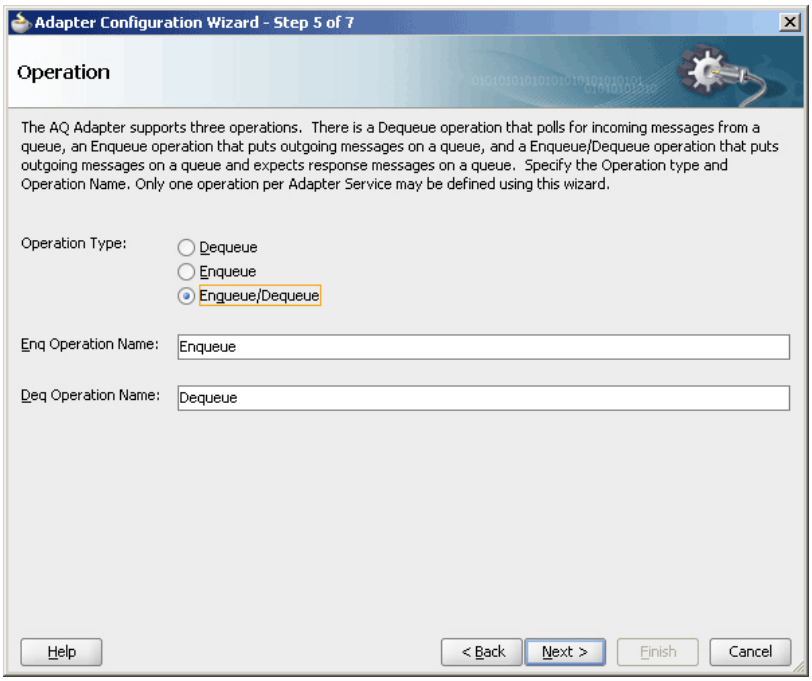
8. In the Adapter Interface page, choose **Define from operation and schema (specified later)**.

9. Click **Next**.

   The Operation page is displayed.

10. Select **Enqueue/Dequeue**, as shown in Figure 7-15.

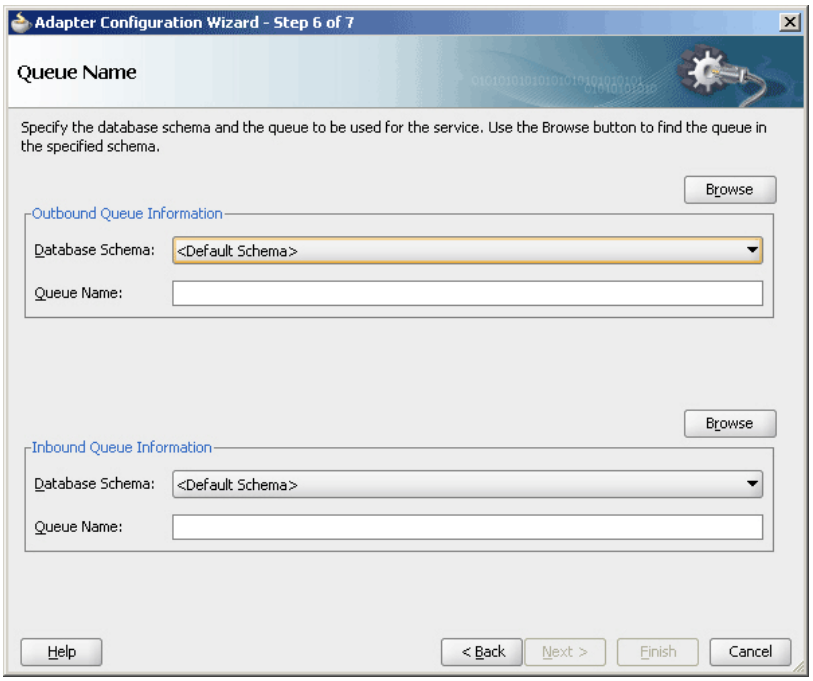
Figure 7-26 The Adapter Configuration Wizard Operation Page



11. Click **Next**.

    The Adapter Configuration Wizard Queue Name page is displayed, as shown in Figure 7-16.

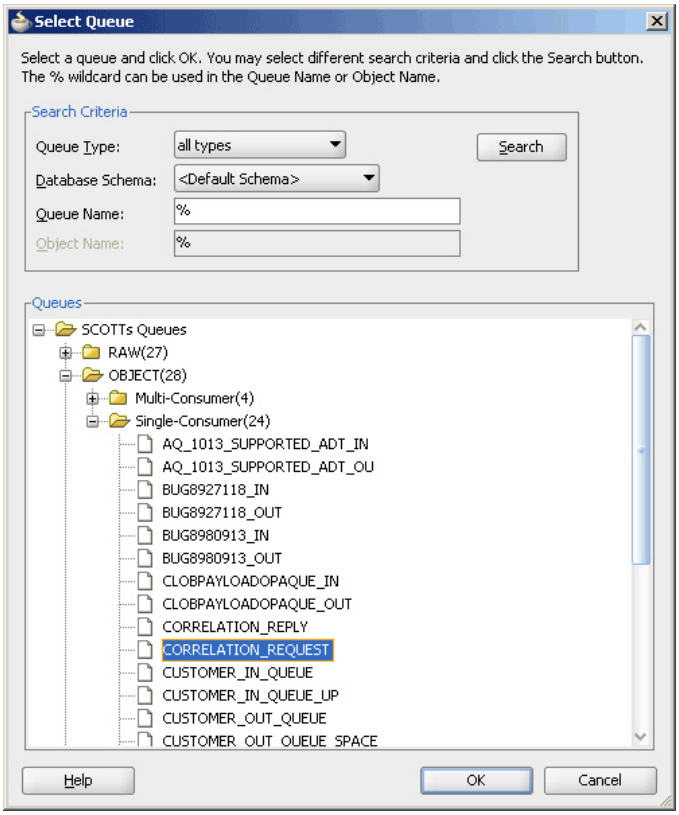
Figure 7-27 The Adapter Configuration Wizard Queue Name Page



12. Click **Browse** to browse for a request queue.

    The Select Queue dialog is displayed, as shown in Figure 7-28.

Figure 7-28 The Select Queue Dialog



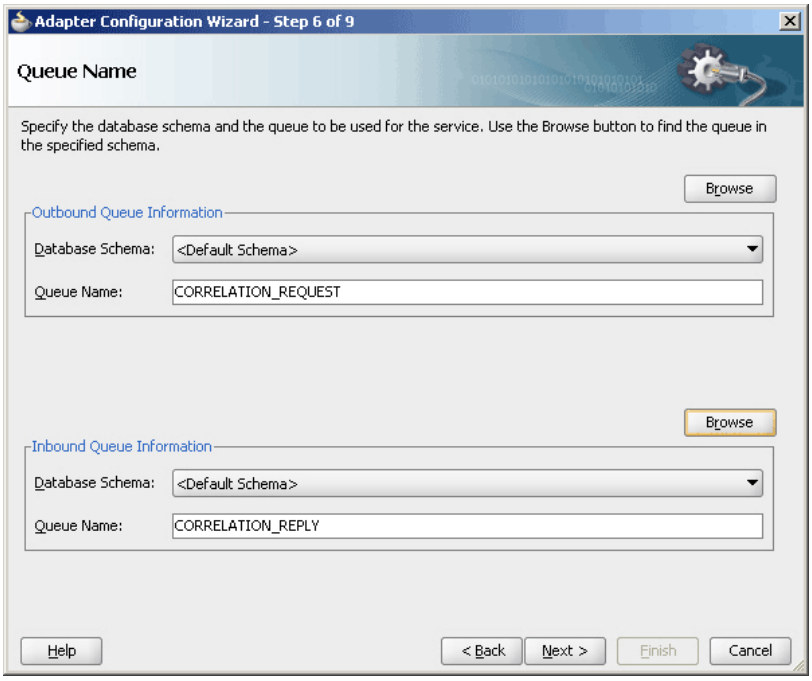
13. Select the required queue, and then click **OK**.

In this example, select **CORRELATION_REQUEST**. The Queue Name page is displayed with the Queue Name field populated with CORRELATION_REQUEST, as shown in Figure 7-29.

14. Repeat Step 12 and 13 for the enqueue queue information.

    The Queue Name page is displayed, as shown in Figure 7-29.

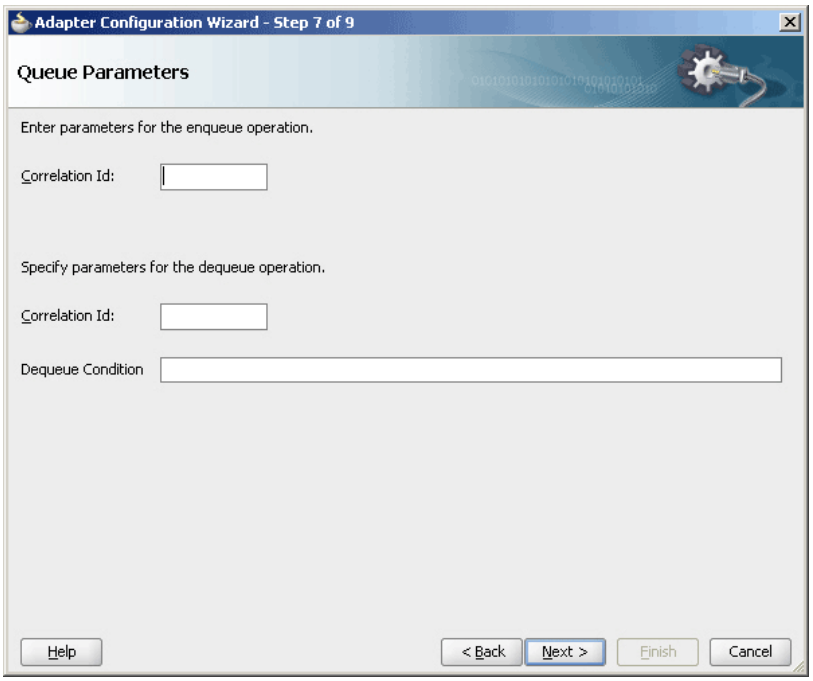
Figure 7-29 The Adapter Configuration Wizard Queue Name Page



15. Click **Next**.

    The Adapter Configuration Wizard Queue Parameters page is displayed, as shown in Figure 7-30.

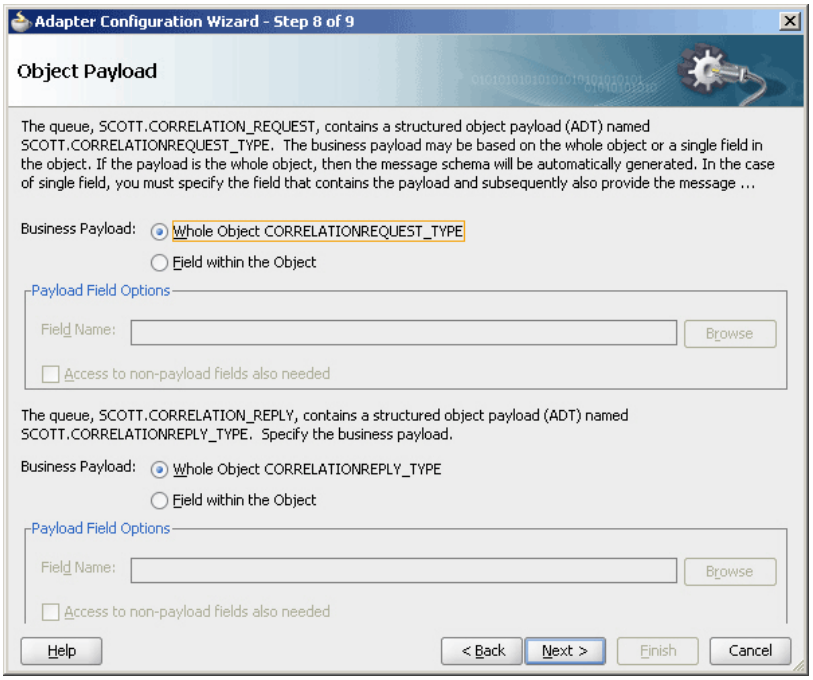
Figure 7-30 The Adapter Configuration Wizard Queue Parameters Page



16. Click **Next**.

    The Adapter Configuration Wizard Object Payload page is displayed, as shown in Figure 7-31.

Figure 7-31 The Adapter Configuration Wizard Object Payload Page



17. Select the Business Payload options, **Whole Object CORRELATIONREQUEST_TYPE** and **Whole Object CORRELATIONREPLY_TYPE**.

18. Click **Next**.

    The Finish screen is displayed. This page shows the path and name of the adapter file that the wizard creates.

19. Click **Finish**.

    You have created an AQ Adapter service for synchronous enqueue/dequeue operations.

20. Click **OK**.

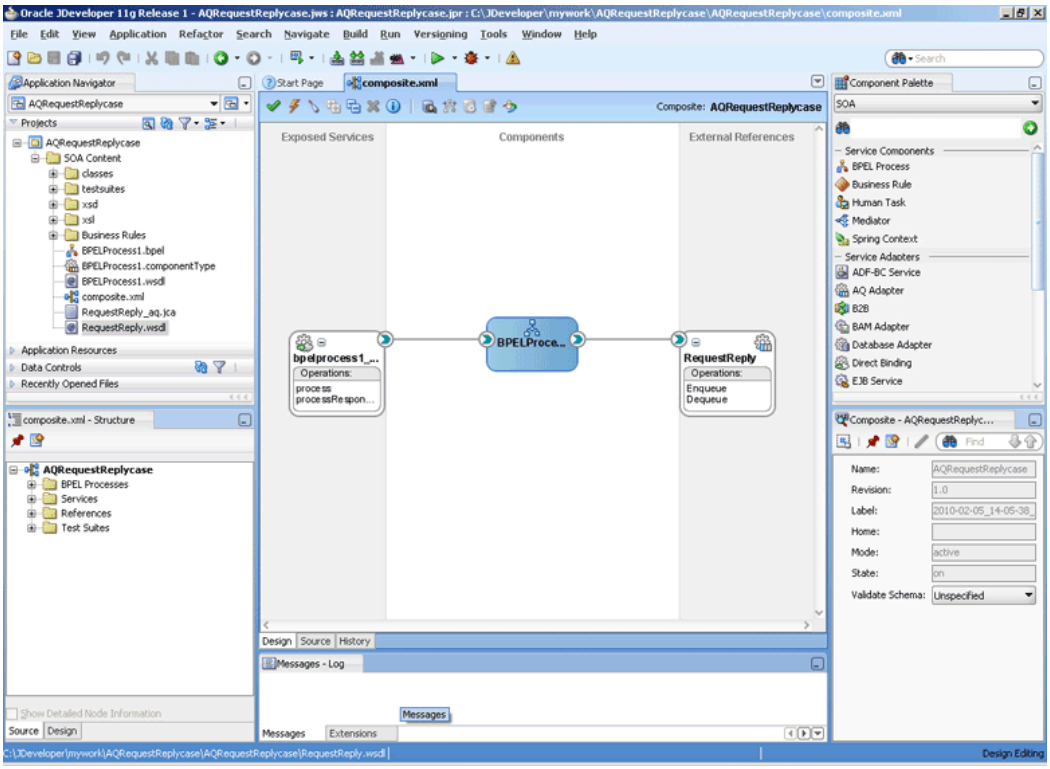
Wiring Services and Activities

You must assemble or wire the BPEL process and the Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in RequestReply in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 7-32.

   Figure 7-32 The JDeveloper - Composite.xml



2. Click **File**, **Save All**.

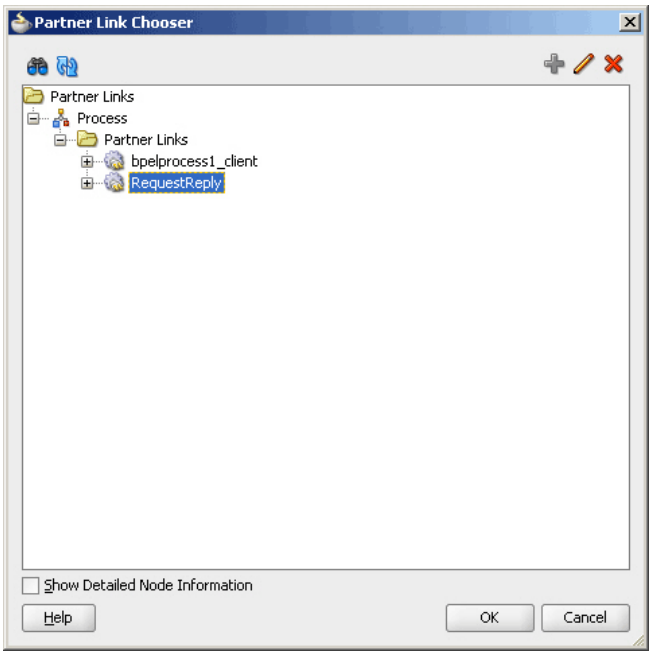
Add Invoke Activity

1. Double-click **BPELProcess1**. The **BPELProcess1.bpel** page is displayed.

2. Drag and drop an **Invoke** activity from the Components window to the design area.

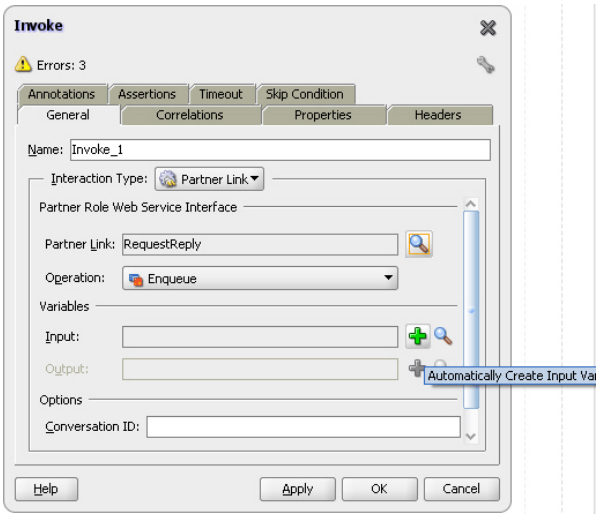
3. Double-click the **Invoke** activity. The Invoke dialog is displayed.
4. Enter a name for the invoke activity in the **Name** field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed, as shown in Figure 7-33.

Figure 7-33 The Partner Link Chooser Dialog



6. Select **RequestReply**, and click **OK**.
7. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog, as shown in Figure 7-34. The Create Variable dialog is displayed.

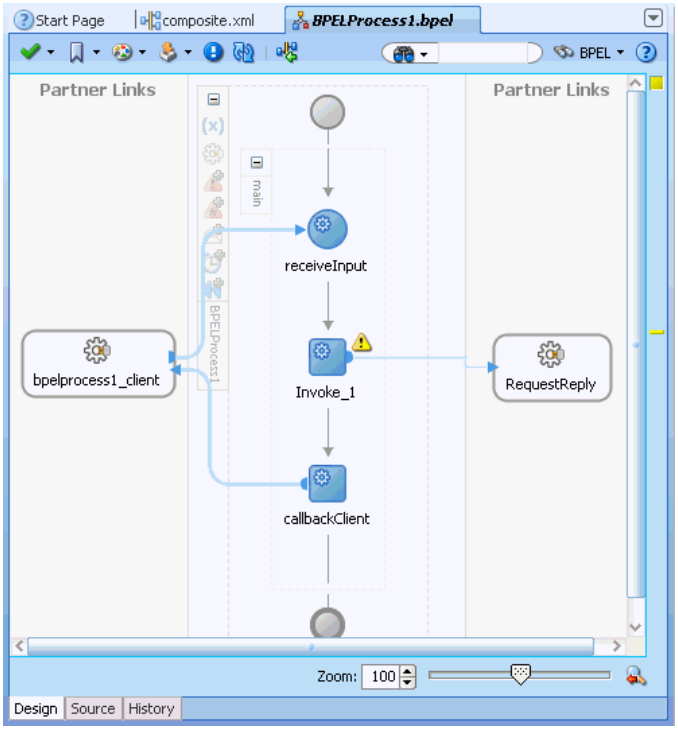
Figure 7-34 The Invoke Dialog



8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.

9. Click **OK**. The JDeveloper **BPELProcess1.bpel** page appears, as shown in Figure 7-35.

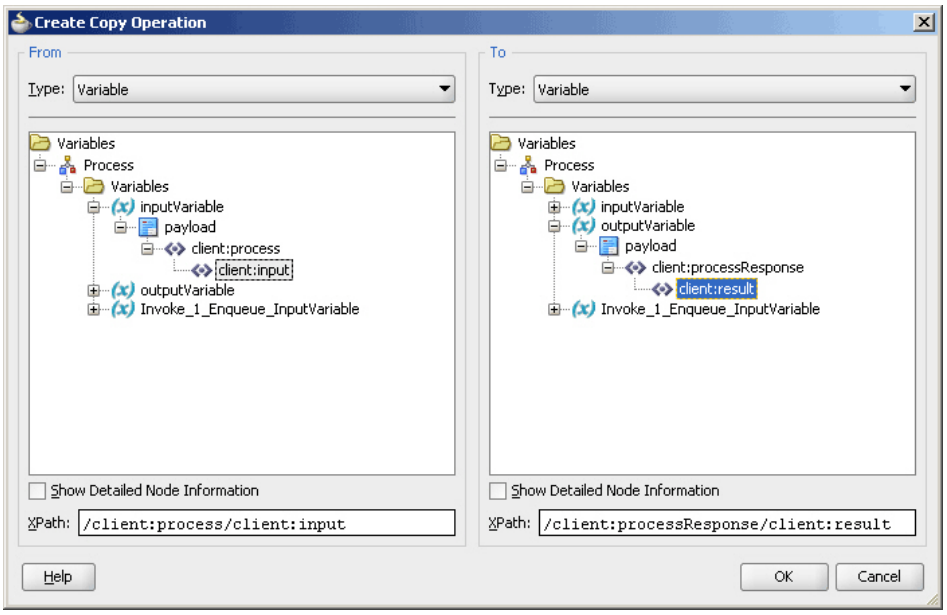
Figure 7-35 The JDeveloper - BPELProcess1.bpel Page



Add an Assign Activity

1. Drag and drop an **Assign** activity from the Components window to the design area.
2. Double-click the **Assign** activity. The Assign dialog is displayed.
3. Enter a name for the Assign activity in the **Name** field.
4. Click the **Copy Operation** tab.
5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
6. Create a copy operation from inputVariable to outputVariable, as shown in Figure 7-36.

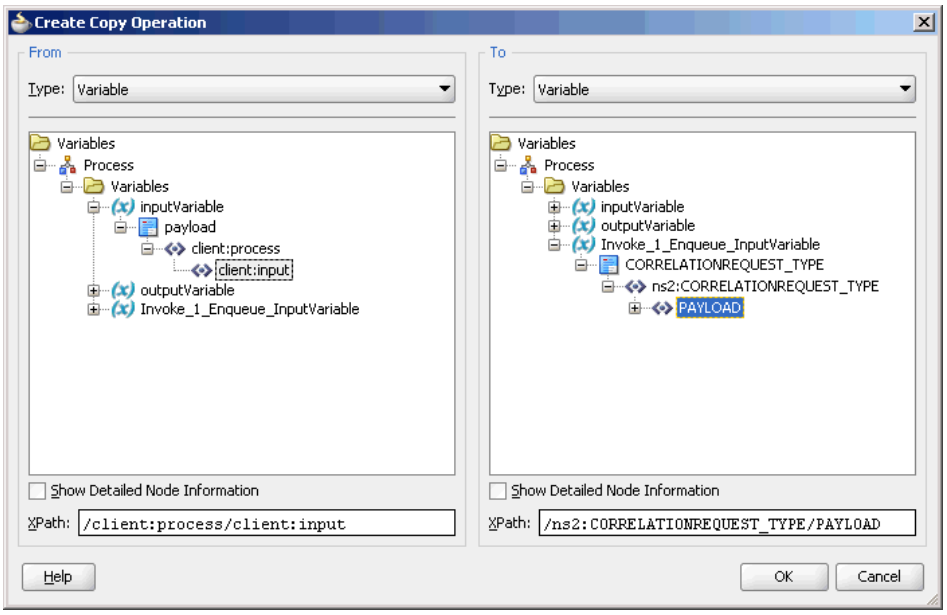
Figure 7-36 The Create Copy Operation Dialog



7. Click **OK** in the Create Copy Operation dialog.

8. Create another copy operation from inputVariable to Invoke_1_Enqueue_InputVariable, as shown in Figure 7-37.

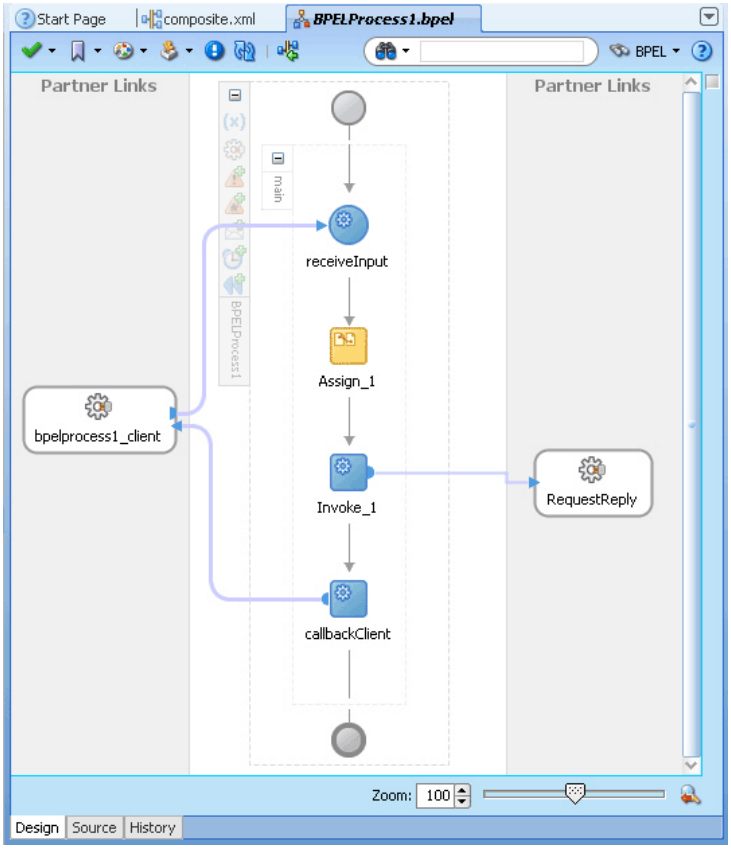
Figure 7-37 The Create Copy Operation Dialog



9. Click **OK** in the Create Copy Operation dialog.

10. Click **OK** to return to the JDeveloper BPELProcess1.bpel page, as shown in Figure 7-38.

Figure 7-38 The JDeveloper - BPELProcess1.bpel

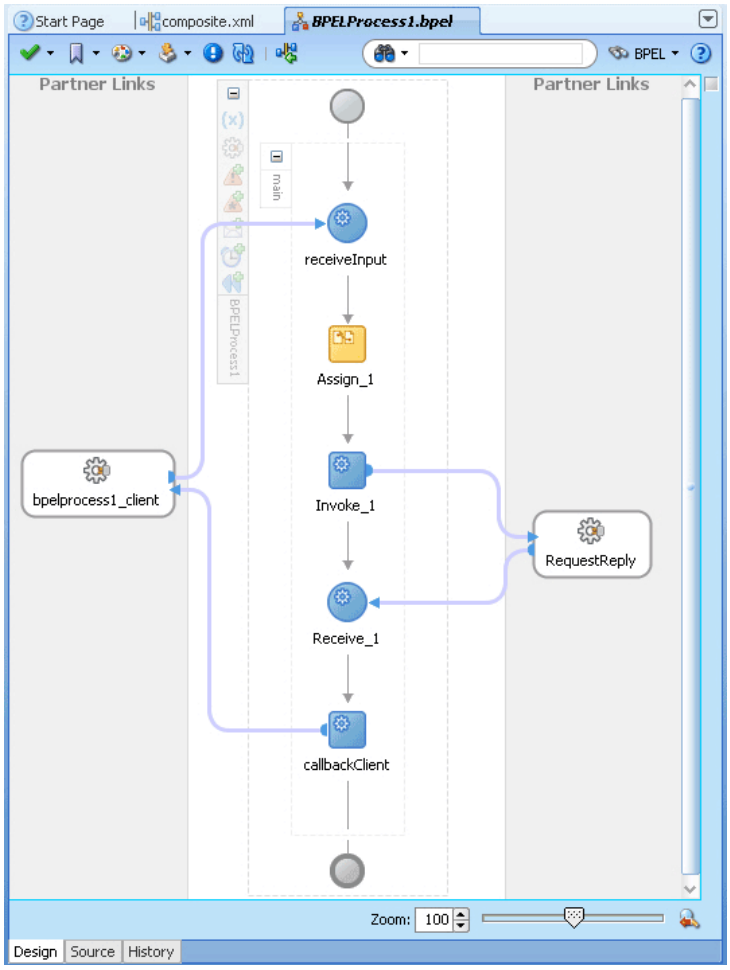


11. Click **File**, **Save All**.

Add a Receive Activity

1. Drag and drop a **Receive** activity from the Components window to the design area.
2. Double-click the **Receive** activity. The Receive dialog is displayed.
3. Enter a name for the Receive activity in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **RequestReply**, and click **OK**.
6. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
8. Check **Create Instance**, and click **OK**. The JDeveloper BPELProcess1.bpel page appears, as shown in Figure 7-39.

Figure 7-39 The JDeveloper - BPELProcess1.bpel



Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps.

The following are the steps to deploy the application profile by using JDeveloper:

1. Create an application server connection by using the procedure described in Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application by using the procedure described in Deploying Oracle JCA Adapter Applications from .

Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed composite by using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed is displayed in the Application Navigator.
2. Click an instance. The Flow Trace page is displayed.
3. Click the BPEL component instance. The Audit page is displayed.

4. Click the **Flow-Debug** tab to debug the instance.

Generated WSDL and JCA Files

The following WSDL file is generated for the Enqueue/Dequeue operation:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<?binding.jca Inbound_aq.jca?>
<definitions name="Inbound" targetNamespace="http://xmlns.oracle.com/pcbpel/
adapter/aq/Inbound/" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://
xmlns.oracle.com/pcbpel/adapter/aq/Inbound/" xmlns:plt="http://
schemas.xmlsoap.org/ws/2003/05/partner-link/" xmlns:obj1="http://
xmlns.oracle.com/xdb/SCOTT" xmlns:imp1="http://www.oracle.com/ipdemo">
    <types>
        <schema attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/" xmlns="http://
www.w3.org/2001/XMLSchema" xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/
Inbound/" xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"
xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT">
            <import namespace="http://xmlns.oracle.com/xdb/SCOTT"
schemaLocation="xsd/SCOTT_MAGAZINE_TYPE.xsd"/>
            <import namespace="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"
schemaLocation="xsd/aqAdapterInboundHeader.xsd"/>
            <complexType name="HeaderCType">
                <sequence>
                    <element name="QueueHeader" type="hdr:HeaderType"/>
                    <element name="PayloadHeader" type="obj1:MAGAZINE_TYPE"/>
                </sequence>
            </complexType>
            <element name="Header" type="tns:HeaderCType"/>
        </schema>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://www.oracle.com/ipdemo"
schemaLocation="xsd/simpleMagazine.xsd"/>
        </schema>
<?xml version = '1.0' encoding = 'UTF-8'?>
   <?binding.jca Inbound_aq.jca?>
    <definitions name="Inbound"
    targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/
     xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
      xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
      xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT"
     xmlns:imp1="http://www.oracle.com/ipdemo">
     <types>
 <schema attributeFormDefault="qualified" elementFormDefault="qualified"
    targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"

xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT">
<import namespace="http://xmlns.oracle.com/xdb/SCOTT"
schemaLocation="xsd/SCOTT_MAGAZINE_TYPE.xsd"/>
<import namespace="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"

     schemaLocation="xsd/aqAdapterInboundHeader.xsd"/>
<complexType name="HeaderCType">
 <sequence>
<element name="QueueHeader" type="hdr:HeaderType"/>
<element name="PayloadHeader" type="obj1:MAGAZINE_TYPE"/>
```

```
 </sequence>
</complexType>

   <element name="Header" type="tns:HeaderCType"/>
</schema>

        <schema xmlns="http://www.w3.org/2001/XMLSchema">
       <import namespace="http://www.oracle.com/ipdemo"
        schemaLocation="xsd/simpleMagazine.xsd"/>
</schema>
</types>
<message name="simpleMagazine_msg">

   <part name="simpleMagazine" element="imp1:simpleMagazine"/>
</message>
     <message name="Header_msg">
     <part name="Header" element="tns:Header"/>
</message>
<portType name="Dequeue_ptt">
    <operation name="Dequeue">
       <input message="tns:simpleMagazine_msg"/>
    </operation>
</portType>
    <plt:partnerLinkType name="Dequeue_plt">
        <plt:role name="Dequeue_role">
            <plt:portType name="tns:Dequeue_ptt"/>
        </plt:role>
   </plt:partnerLinkType>
</definitions>
```

The following JCA file is generated for the Enqueue/Dequeue operation:

```
<adapter-config name="RequestReply" adapter="AQ Adapter"
wsdlLocation="RequestReply.wsdl" xmlns="http://platform.integration.oracle/blocks/
adapter/fw/metadata">

  <connection-factory location="eis/AQ/aqSample" UIConnectionName="aqSample"
adapterRef=""/>
  <endpoint-activation portType="Dequeue_ptt" operation="Dequeue"
UITransmissionPrimitive="Request-response">
    <activation-spec
className="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec">
      <property name="QueueName" value="CORRELATION_REPLY"/>
    </activation-spec>
  </endpoint-activation>

  <endpoint-interaction portType="Enqueue_ptt" operation="Enqueue"
UITransmissionPrimitive="Request-response">
    <interaction-spec
className="oracle.tip.adapter.aq.outbound.AQEnqueueInteractionSpec">
      <property name="QueueName" value="CORRELATION_REQUEST"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

Using Correlation ID for Filtering Messages During Dequeue

Perform the following steps to set up an adapter that dequeues messages with a certain correlation ID only.

- Select **Dequeue** operation in Step 10.
- Enter the correlation ID in Step 15.

The adapter dequeues messages enqueued with that same correlation ID only.

For a working example of this use case where an Oracle AQ Adapter dequeues messages enqueued with that same correlation ID, refer to the following samples:

- `adapters-aq-106-messagerejection`
- `adapters-aq-109-nativecorrelation`
- `adapters-aq-112-prioritymessageselector`
- `adapters-aq-113-payloadbasedmessageselector`

You can obtain these samples by accessing the Oracle SOA Sample Code site.

Enqueuing and Dequeuing from Multisubscriber Queues

Multisubscriber queues are accessible by multiple users, and sometimes, those users are concerned only with certain types of messages within the queue. For example, you may have a multiuser queue for loan applications where loans below $100,000 can be approved by regular loan-approval staff, whereas loans over $100,000 must be approved by a supervisor. In this case, the BPEL process can use one adapter to enqueue loan applications for big loans for supervisors, and another adapter to enqueue loan applications for smaller loans for regular staff in the same multisubscriber queue.

Specify an adapter that enqueues to a multisubscriber queue, and include queue parameters in the **Recipients** field.

In Step 15, specify **Bob** in the **Recipients** field.

The following code is from a JCA file generated by defining an Oracle AQ Adapter that enqueues with a recipient list of `Bob`:

```
<adapter-config name="Inbound" adapter="AQ Adapter" xmlns="http://
platform.integration.oracle/blocks/adapter/fw/metadata">
    <connection-factory location="eis/AQ/aqSample" UIConnectionName="aqSample"
adapterRef=""/>
  <endpoint-interaction portType="Enqueue_ptt" operation="Enqueue">
    <interaction-spec
className="oracle.tip.adapter.aq.outbound.AQEnqueueInteractionSpec">
      <property name="QueueName" value="PURCHASEORDER_APPROVAL"/>
      <property name="RecipientList" value="Bob"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

When dequeuing from a multisubscriber queue, the Queue Parameters window is displayed.

The **Consumer** field is where you specify the consumer name, or the name of the queue subscriber. This must match the **Recipient** entry on the enqueue process for the message to be dequeued. When subscribing to a multiconsumer queue, this field is required.

The following code is from a JCA file generated by defining an Oracle AQ Adapter with a consumer name:

```
<adapter-config name="Dequer_Bob" adapter="AQ Adapter" xmlns="http://
platform.integration.oracle/blocks/adapter/fw/metadata">
    <connection-factory location="eis/AQ/manas" UIConnectionName="aqSample"
adapterRef=""/>
  <endpoint-activation portType="Dequeue_ptt" operation="Dequeue">
    <activation-spec
className="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec">
      <property name="QueueName" value="PURCHASEORDER_APPROVAL"/>
      <property name="Consumer" value="Bob"/>
      <property name="SchemaValidation" value="false"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

For a working example of this use case which demonstrates enqueuing and dequeuing from multisubscriber queues, refer to the following samples:

- `adapters-aq-114-multiconsumeroutbound`

You can obtain these samples by accessing the Oracle SOA Sample Code site.

Oracle AQ Adapter ADT Queue

In this sample, the business process receives a message from the AQ Adapter, copies the payload to an outbound message, and invokes the AQ Adapter with the outbound message.The queues involved are ADT queues. In this scenario, where the user has chosen to use whole ADT as the payload, the AQ Adapter Wizard has generated the schema in `SCOTT_CUSTOMER_TYPE.xsd`, according to the queue structure. During runtime, an XML file that matches the schema is created by the adapter for each message.

This section includes the following topics:

- Meeting Prerequisites
- Creating an Application and an SOA Project
- Creating an Inbound Oracle AQ Adapter
- Creating an Outbound Oracle AQ Adapter
- Wiring Services and Activities
- Configuring Routing Service
- Deploying with
- Configuring the Data Sources in the

Meeting Prerequisites

You must have access to a database with the SCOTT schema.

To perform this use case, you require the following SQL files from the `artifacts.zip` file contained in the `adapters-aq-102-adt` sample. These files are located in the `artifacts/sql` subdirectory of the `artifacts.zip` file. Execute the SQL files in the order shown below:

- `setup_user.sql`
- `create_type_customer.sql`

- `create_queues.sql`
- `enqueue_customer.sql`
- `dequeue_customer.sql`

To obtain the adapters-aq-102-adt sample code, access the Oracle SOA Sample Code site

Creating an Application and an SOA Project

You must create an JDeveloper application to contain the SOA composite. Use the following steps to create an application and an SOA project:

1. In the **Application Navigator** of JDeveloper, click **New Application**.

   The Create Generic Application Name your application page is displayed.

2. Enter `ADT` in the **Application Name** field, and click **Next**.

   The Create Generic Application Name your project page is displayed.

3. Enter `ADT` in the **Project Name** field.

4. In the Available list in the Project Technologies tab, double-click **SOA** to move it to the Selected list.

5. Click **Next**.

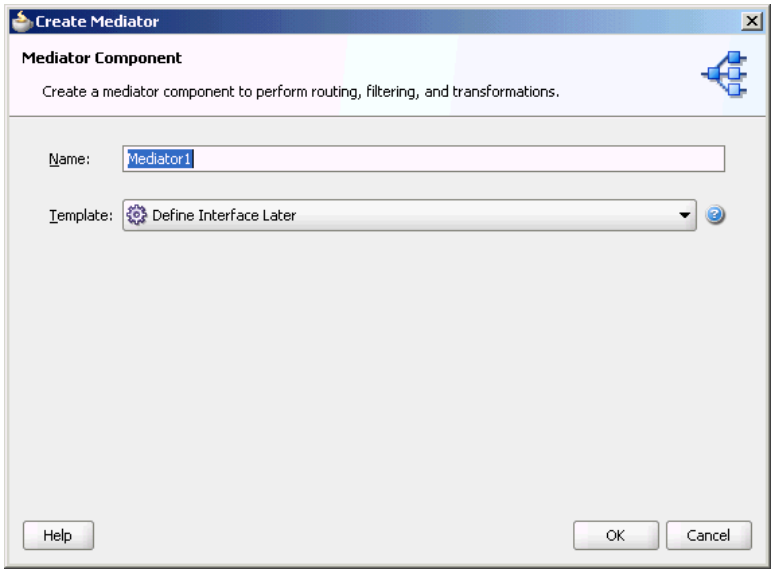
   The Create Generic Application Configure SOA settings page is displayed.

6. Select **Composite With Mediator** from the Composite Template list, and then click **Finish**.

   You have created a new application and an SOA project.

   The Create Mediator page is displayed, as shown in Figure 7-40.

Figure 7-40 The Create Mediator Page



7. Enter a name for the Mediator component in the **Name** field. In this example, retain the default name `Mediator1`.

8. Select **Define Interface Later** in the Template list, and then click **OK**.

   You have created a mediator component.

Creating an Inbound Oracle AQ Adapter

The following are the steps to create an inbound Oracle AQ Adapter service:

1. In the Components window, select **SOA**.
2. Drag and drop **AQ Adapter** from the Service Adapters list in the Components window to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard is displayed.
3. Click **Next**.

   The Service Name page is displayed.
4. Specify a name for the service in the Service Name page. In this example, type `dequeue`.
5. Click **Next**.

   The Service Connection page is displayed. A database connection is required to configure an Oracle AQ Adapter. You can either create a new connection or select an existing database connection.
6. Click the **Create a new database connection** icon to create a database connection.

   The Create Database Connection page is displayed.
7. Create a database connection, as mentioned in Step 6 of Defining an Service.
8. Click **OK** to complete the process of creating a new database connection.

   The Service Connection page is displayed, providing a summary of the database connection.
9. Click **Next**.

   The Adapter Interface page is displayed.
10. In the Adapter Interface page, select **Define from operation and schema (specified later)**.
11. Click **Next**.

    The Operation page is displayed.
12. Select **Dequeue**.
13. Accept the default operation name, and then click **Next**.

    The Queue Name page is displayed.
14. Select a database schema from the list, or click **Browse** to browse for the schema. In this example, click **Browse**.

    The Select Queue dialog is displayed.
15. In the Select Queue dialog, perform the following steps:

    a. For Queue Type, select **all types**.

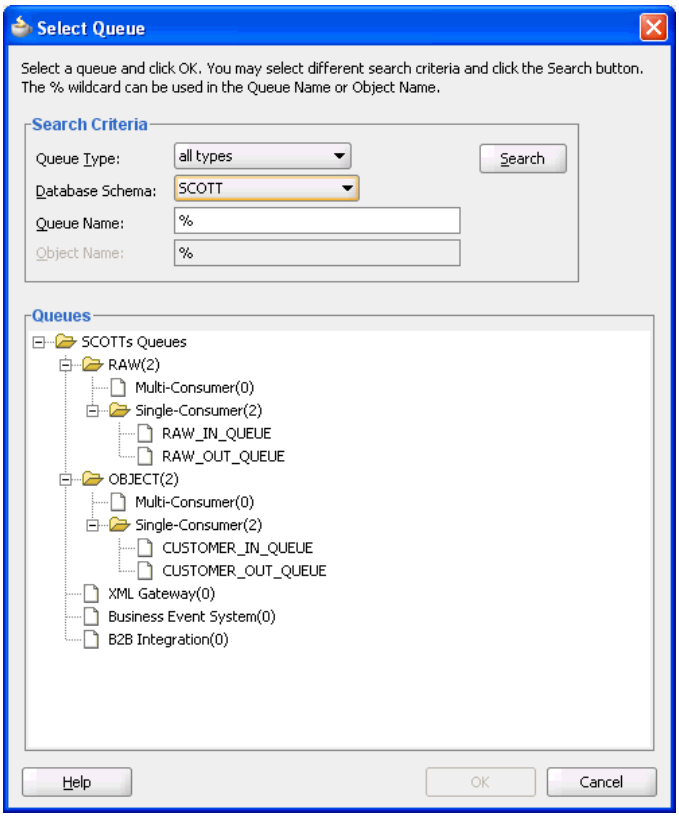
b. For Database Schema, select **Scott**.

c. Retain the default values for the other fields.

d. Under Queues, select **CUSTOMER_IN_QUEUE**.

Figure 7-41 shows the Select Queue dialog.

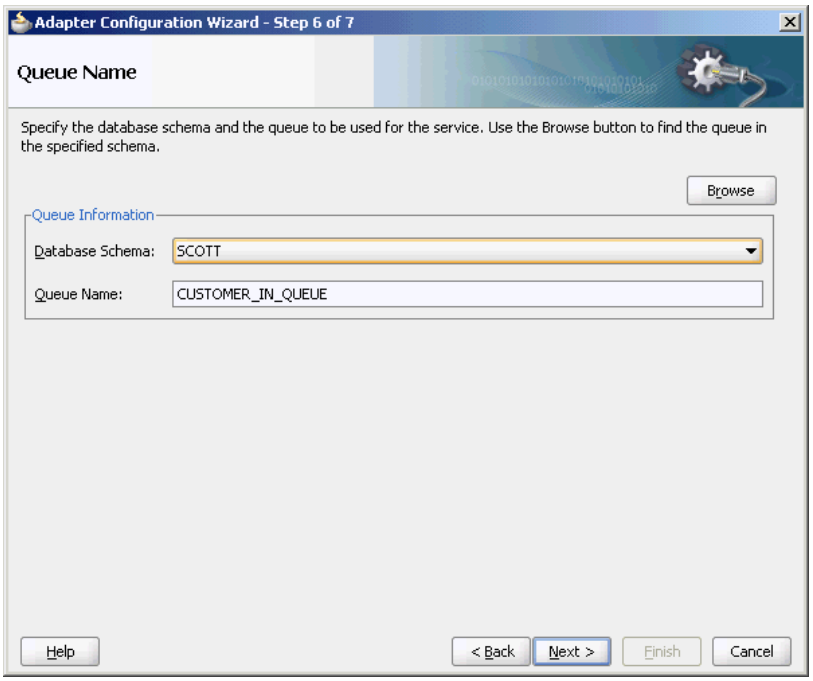
Figure 7-41 Selecting a Queue for the Inbound Operation



16. Click **OK**.

The Queue Name dialog with all the fields populated is displayed, as shown in Figure 7-42.

Figure 7-42 The Queue Name Page



17. Click **Next**.

    The Queue Parameters page is displayed.

18. In the Queue Parameters page, leave the fields empty, and then click **Next**.

    The Object Payload page is displayed.

19. Select a business payload: either the entire object, or just one field within the object.

    In this example, select **Whole Object CUSTOMER_TYPE**.

20. Click **Next**.

    The Finish screen is displayed. This page shows the path and name of the adapter file that the wizard creates.

21. Click **Finish**.

    You have defined an inbound Oracle AQ Adapter

Creating an Outbound Oracle AQ Adapter

The following are the steps to create an outbound Oracle AQ Adapter service:

1. In the Components window, select **SOA**.
2. Drag and drop **AQ Adapter** from the Service Adapters list in the Components window to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard is displayed.

3. Click **Next**.

   The Service Name page is displayed.

4. In the Service Name field, enter `enqueue` and click **Next**.

   The Service Connection page is displayed.

5. For Connection, select **MyConnection**, and then click **Next**.

   The Adapter Interface page is displayed.

6. In the Adapter Interface page, select **Define from operation and schema (specified later)**, and then click **Next**.

   The Operation page is displayed.

7. In the Operation page, select **Enqueue**, and accept the default operation name.

8. Click **Next**.

   The Queue Name page is displayed.

9. In the Queue Name page, select a database schema from the list, or click **Browse** to browse for the schema. In this example, click **Browse**.

   The Select Queue dialog is displayed.

10. In the Select Queue dialog, perform the following steps:

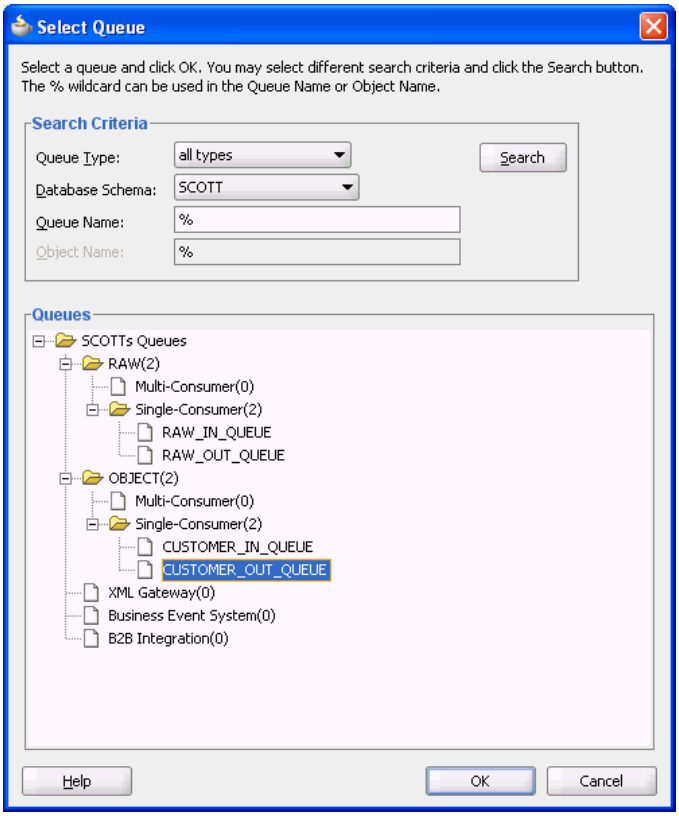
    a. For Queue Type, select **all types**.

    b. For Database Schema, select **Scott**.

    c. Retain the default values for the other fields.

    d. Under Queues, select **CUSTOMER_OUT_QUEUE**, as shown in Figure 7-43.

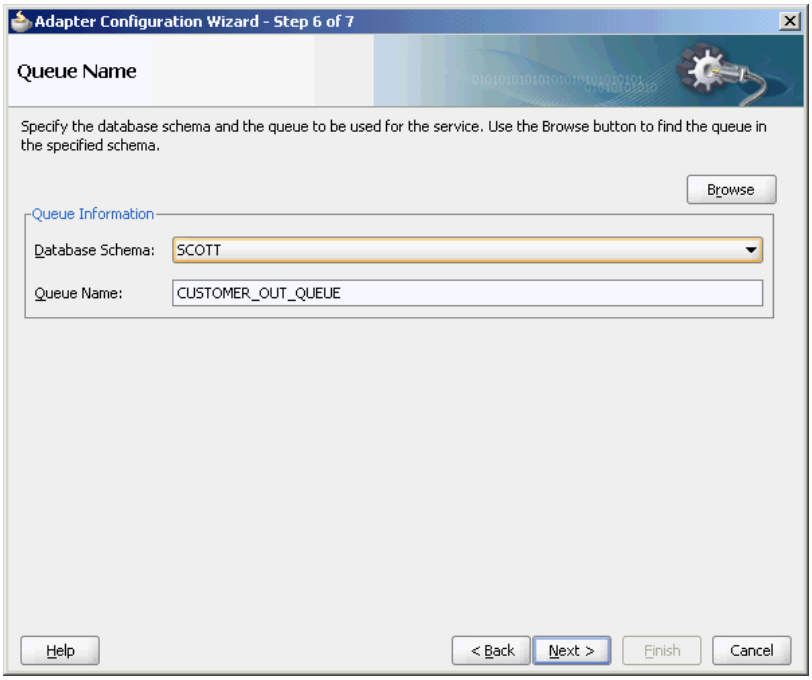
Figure 7-43 Selecting a Queue for the Outbound Operation



11. Click **OK**.

    The Queue Name page with all the fields populated is displayed, as shown in Figure 7-44.

Figure 7-44 The Queue Name Page



12. Click **Next**.

    The Queue Parameters page is displayed.

13. In the Queue Parameters page, leave the fields empty, and then click **Next**.

    The Object Payload page is displayed.

14. Select a business payload, either the entire object, or just one field within the object. In this example, select **Whole Object CUSTOMER_TYPE**.

15. Click **Next**.

    The Finish screen is displayed. This page shows the path and name of the adapter file that the wizard creates.

16. In the Finish window, click **Finish**.

    You have defined an outbound Oracle AQ Adapter.

Wiring Services and Activities

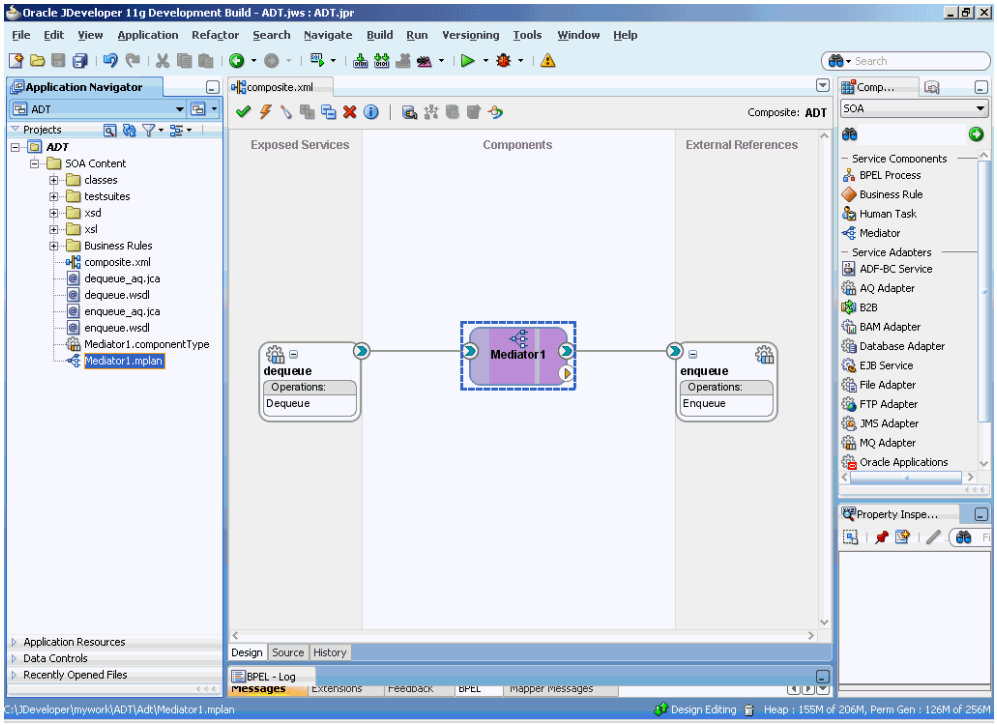
You must assemble or wire the three components that you have created: Inbound adapter service, Mediator component, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the Inbound adapter in the Exposed Services area to the drop zone that appears as a green triangle in the Mediator component in the Components area.

2. Drag the small triangle in the Mediator component in the Components area to the drop zone that appears as a green triangle in the Outbound adapter in the External References area.

   The JDeveloper composite.xml is displayed, as shown in Figure 7-45.

Figure 7-45 The JDeveloper composite.xml



3. Click **File**, **Save All**.

Configuring Routing Service

The following are the steps to configure the routing service:

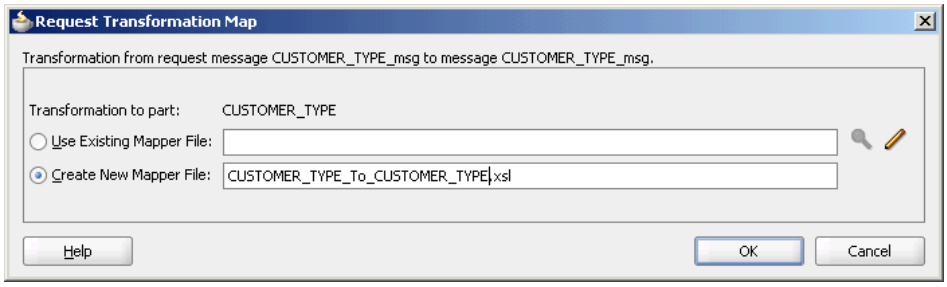
1. Double-click **Mediator1**.

   The Mediator1.mplan window is displayed.

2. Click the **Select an existing mapper file or create a new one...** icon that is displayed at the end of the Transform Using field.

   The Request Transformation Map dialog is displayed, as shown in Figure 7-46.

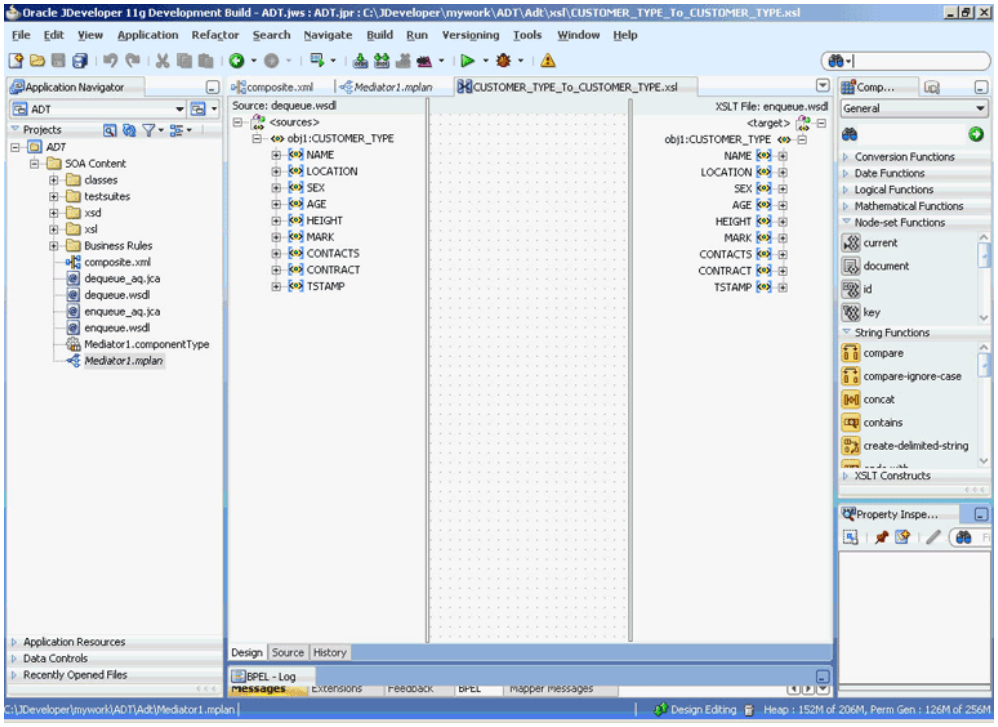
Figure 7-46 The Request Transformation Map Dialog



3. Select **Create New Mapper File**, and then click **OK**.

   The Transformation window is displayed, as shown in Figure 7-47.

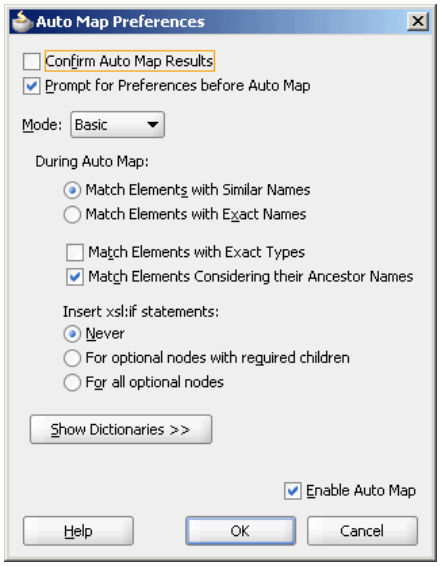
Figure 7-47 The Transformation Window



4. Select the source root elements on the left-hand side of the mapper and drag them over to the destination root elements on the right-hand side to set the map preferences.

   The Auto Map Preferences dialog is displayed, as shown in Figure 7-48.

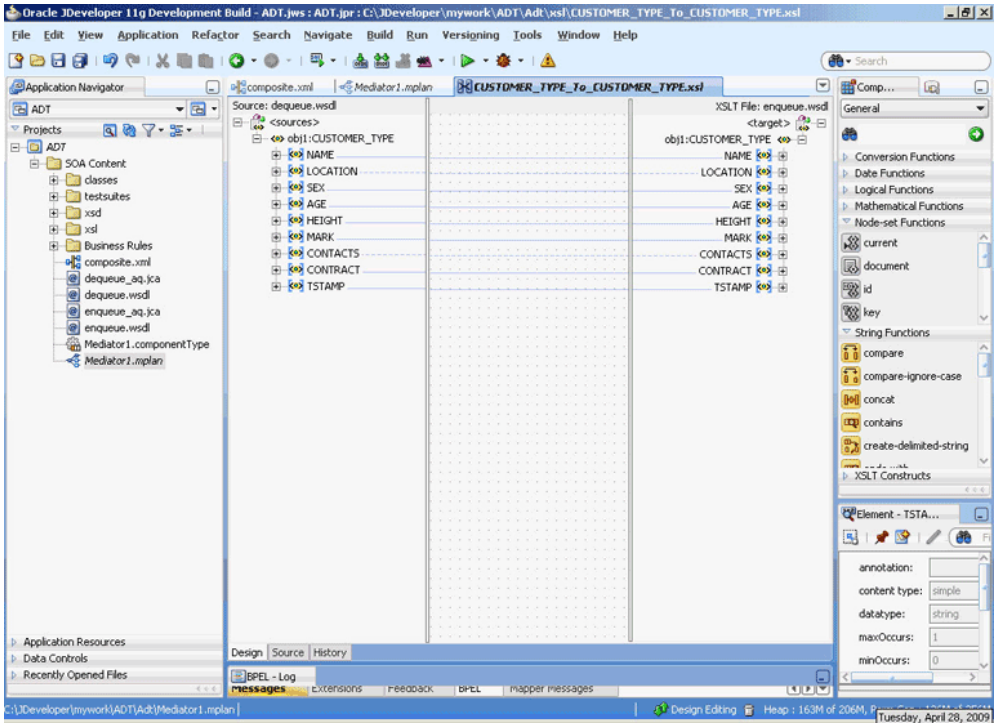
Figure 7-48 The Auto Map Preferences Dialog



5. Click **OK**.

   The middle pane of the application window appears as shown in Figure 7-49.

Figure 7-49 The Application Window After Setting the Map Preferences



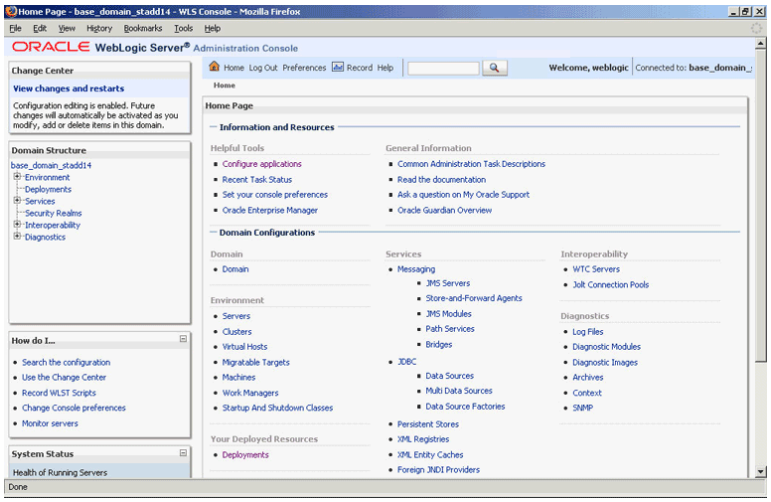
6. Save and close the tab for the mapper.
7. Save and close the tab for the routing service.

Configuring the Data Sources in the Oracle WebLogic Server Administration Console

1. Navigate to `http://servername:portnumber/console`.
2. Use the required credentials to open the Home page of the Oracle WebLogic Server Administration Console.

   The Home page of the Oracle WebLogic Server Administration Console is displayed, as shown in Figure 7-50.

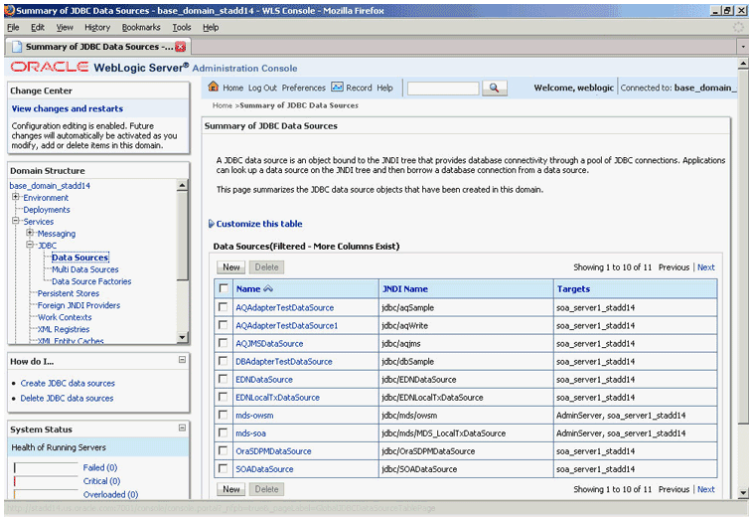
Figure 7-50 Oracle WebLogic Server Administration Console Home Page



3. Under Domain Structure, select **Services**, **JBDC**, and then click **DataSources**.

   The Summary of JDBC Data Sources page is displayed, as shown Figure 7-51.

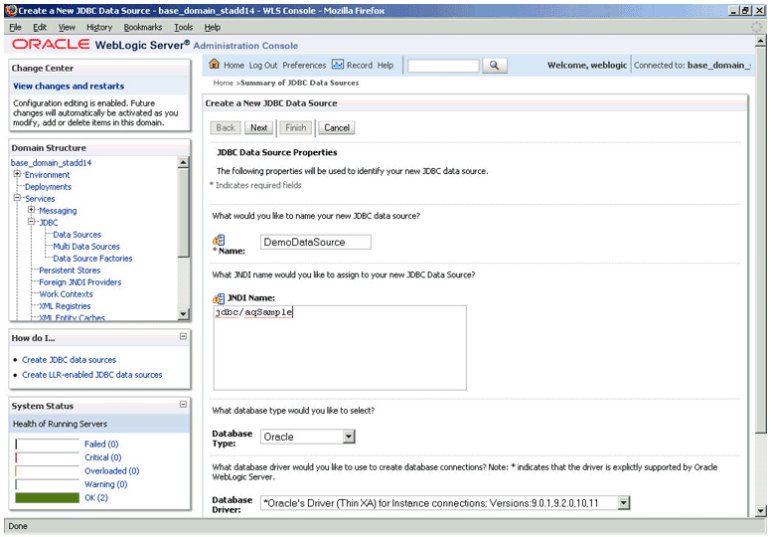
Figure 7-51 The Summary of JDBC Data Sources Page



4. Click **New**. The Create a New JDBC Data Source page is displayed.

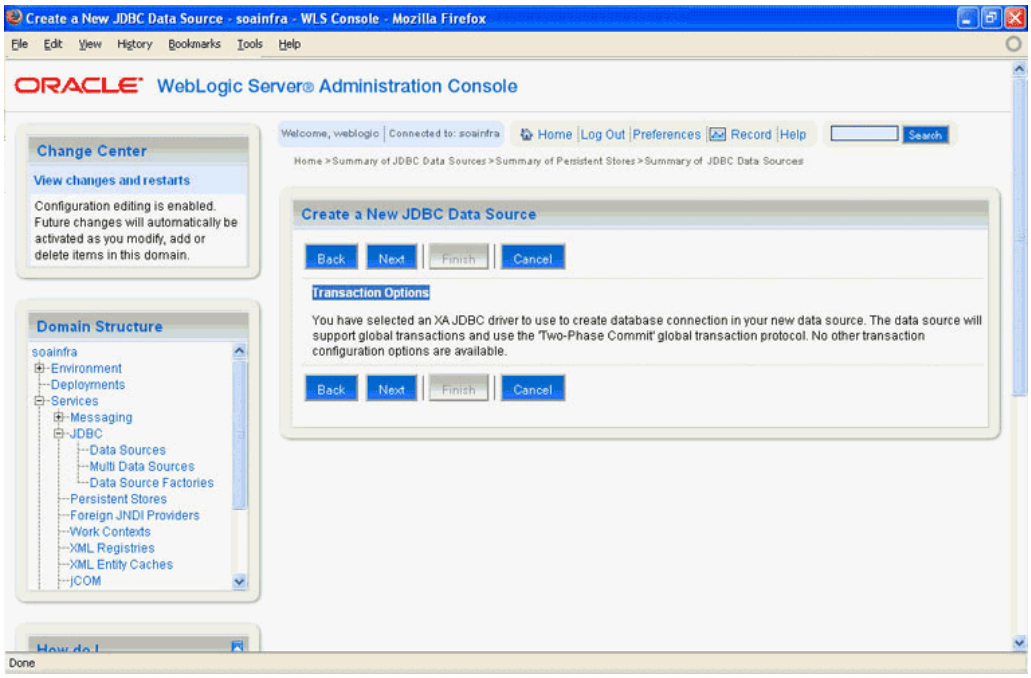
5. Enter the values for the properties to be used to identify your new JDBC data source, as shown in Figure 7-52.

Figure 7-52 The Create a New JDBC Data Source Page



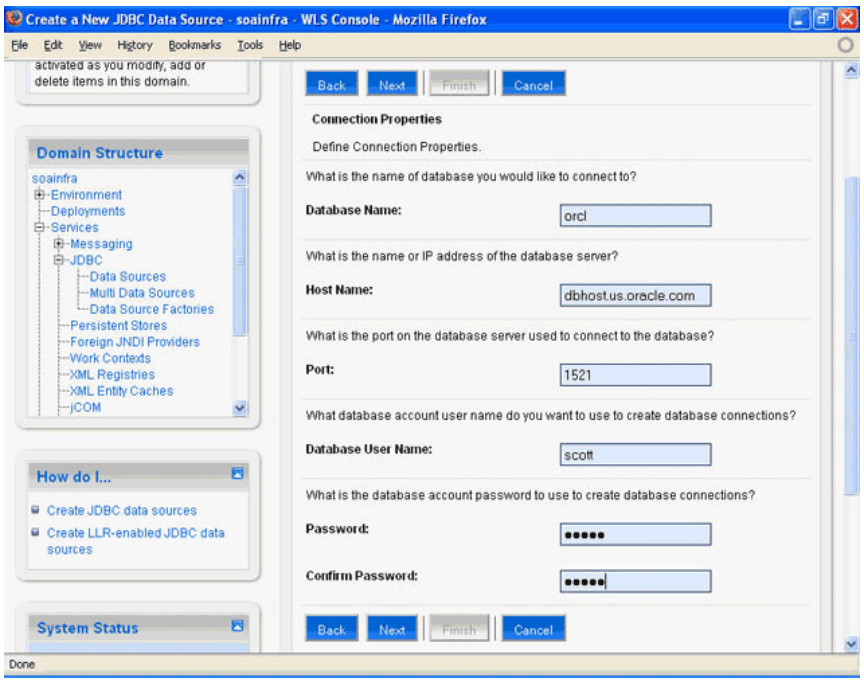
6. Click **Next**. The Create a New JDBC Data Source Transaction Options page is displayed, as shown in Figure 7-53.

Figure 7-53 The Create a New JDBC Data Source Transaction Options Page



7. Click **Next**. The Create a New JDBC Data Source Connection Properties page is displayed, as shown in Figure 7-54.

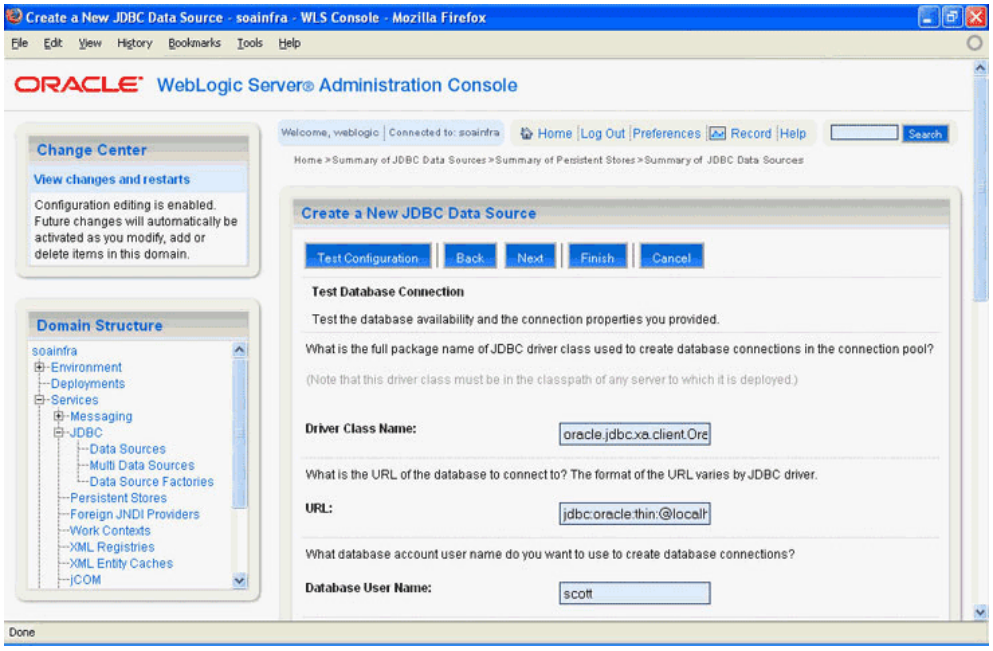
Figure 7-54 The Create a New JDBC Data Source Connection Properties Page



8. Enter the connection properties in the Connection Properties page.

9. Click **Next**. The Create a New JDBC Data Source Test Database Connection page is displayed, as shown in Figure 7-55.

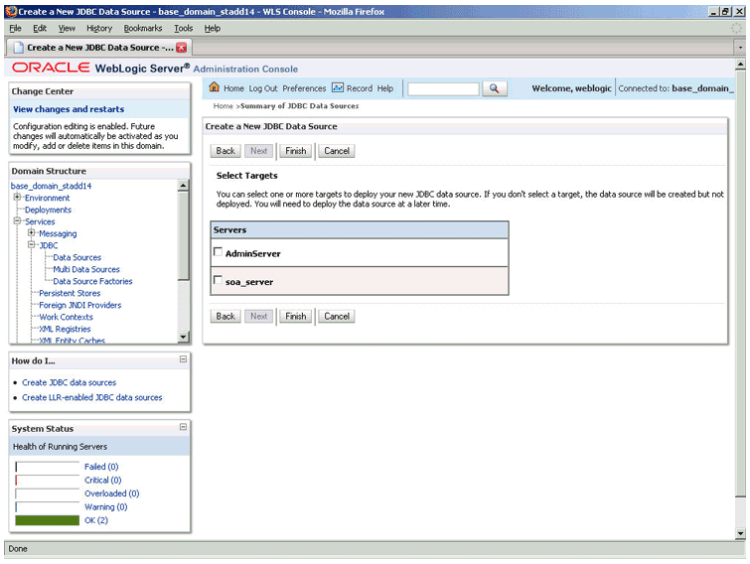
Figure 7-55 The Create a New JDBC Data Source Test Database Connection Page



10. Click **Test Configuration** to test the database availability and the connection properties you provided. A message stating that the connection test succeeded is displayed at the top of the Create a New JDBC Data Source Test Database Connection page.

11. Click **Next**. The Create a New JDBC Data Source Select Targets page is displayed, as shown in Figure 7-56.

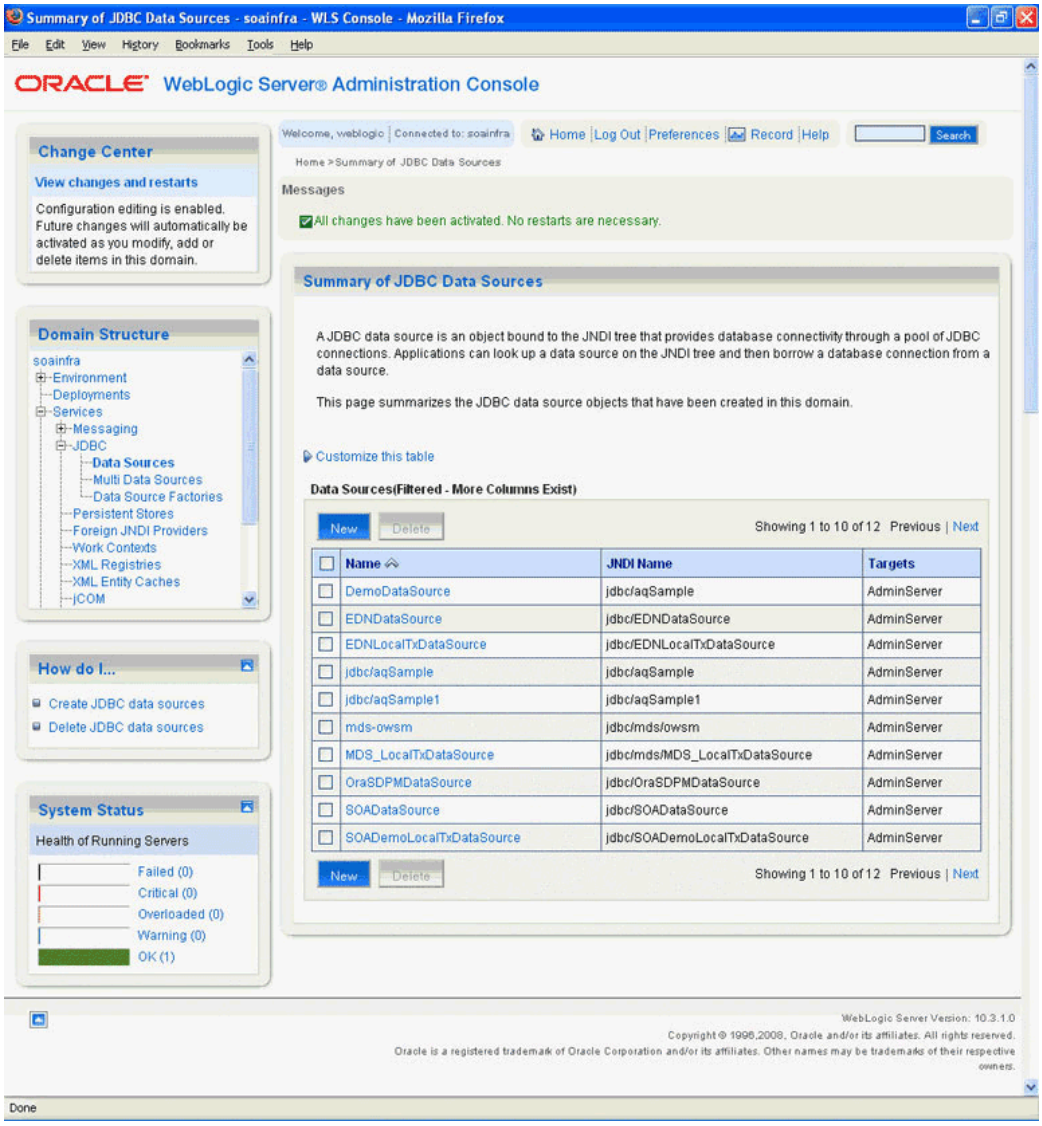
Figure 7-56 The Create a New JDBC Data Source Select Targets Page



12. Select a target, and then click **Finish**.

    The Summary of JDBC Data Sources page is displayed, as shown in Figure 7-57. This page summarizes the JDBC data source objects that have been created in this domain. The data source that you created appears in this list.

Figure 7-57 The Summary of JDBC Data Sources Page



13. Close the Oracle WebLogic Server Administration Console.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps.

The following are the steps to deploy the application profile by using JDeveloper:

1. Create an application server connection by using the procedure described in Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application by using the procedure described in Deploying Oracle JCA Adapter Applications from .

Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed composite by using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed is displayed in the Application Navigator.
2. In the Last 5 Instances pane, there is an entry of a new instance. This new instance is the instance that was triggered when you enqueued a message.
3. Click an instance. The Flow Trace page is displayed.
4. Click the **Mediator1** component instance. The Audit page is displayed.
5. Click the **Flow-Debug** tab to debug the instance.

Oracle AQ Adapter RAW Queue

This use case demonstrates how to use Oracle AQ Adapter to dequeue from and enqueue to an AQ RAW queue.

This section includes the following topics:

- Prerequisites
- Creating an Application and an SOA Project
- Creating an Inbound Adapter Service
- Creating an Outbound Adapter Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Configuring the Data Sources in the Oracle WebLogic Server Administration Console
- Monitoring Using the Fusion Middleware Control Console

Prerequisites

You must have access to a database with the `SCOTT` schema.

To perform this use case, you require the following SQL files from the `artifacts.zip` file contained in the `adapters-aq-101-raw` sample. These files are located in the `artifacts/sql` subdirectory of the `artifacts.zip` file. Execute the SQL files in the order shown below:

- `setup_user.sql`
- `create_queues.sql`
- `enqueue_raw.sql`
- `dequeue_raw.sql`
- `artifacts/schemas/emp.xsd`

To obtain the `artifacts.zip` contained in the `adapters-aq-101-raw` sample code, access the Oracle SOA Sample Code site.

To obtain the `adapters-aq-101-raw` sample code, access the Oracle SOA Sample Code site.

Creating an Application and an SOA Project

You must create an JDeveloper application to contain the SOA composite. To create an application and an SOA project, perform the following steps:

1. Open JDeveloper.
2. In the Application Navigator, click **New Application**.

   The Create Generic Application Name your Application page is displayed.
3. Enter `Rawqueue` in the **Application Name** field.
4. In the Application Template list, select **Generic Application**.
5. Click **Next**.

   The Create Generic Application Name your project page is displayed.
6. In the **Project Name** field, enter a descriptive name, for example, `Raw`.
7. In the Available list in the Project Technologies tab, double-click **SOA** to move it to the Selected list.
8. Click **Next**.

   The Create Generic Application Configure SOA settings page is displayed.
9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

   You have created a new application and an SOA project. This automatically creates an SOA composite.

   The Create BPEL Process page is displayed.
10. Enter a name for the BPEL process in the **Name** field. For example, `BPELRawqueue`.
11. Select **Define Service Later** in the Template list, and then click **OK**.

    The `Rawqueue` application and the `Raw` project appear in the design area.
12. Copy the `emp.xsd` file to the `XSD` folder in your project (see Prerequisites for the location of this file).

Creating an Inbound Adapter Service

Perform the following steps to create an inbound Oracle AQ Adapter service that dequeues the message to a queue:

1. In the Components window, select **SOA**.
2. Drag and drop **AQ Adapter** from the Service Adapters list in the Components window to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard is displayed.
3. Click **Next**.

   The Service Name page is displayed.

4. In the Service Name field, enter `Raw-Dequeuer`, and then click **Next**.

   The Service Connection page is displayed.

5. Create a database connection, as mentioned in Step 6 of Defining an Service.

6. Click **Next**.

   The Adapter Interface page is displayed.

7. In the Adapter Interface page, select **Define from operation and schema (specified later)**, and then click **Next**.

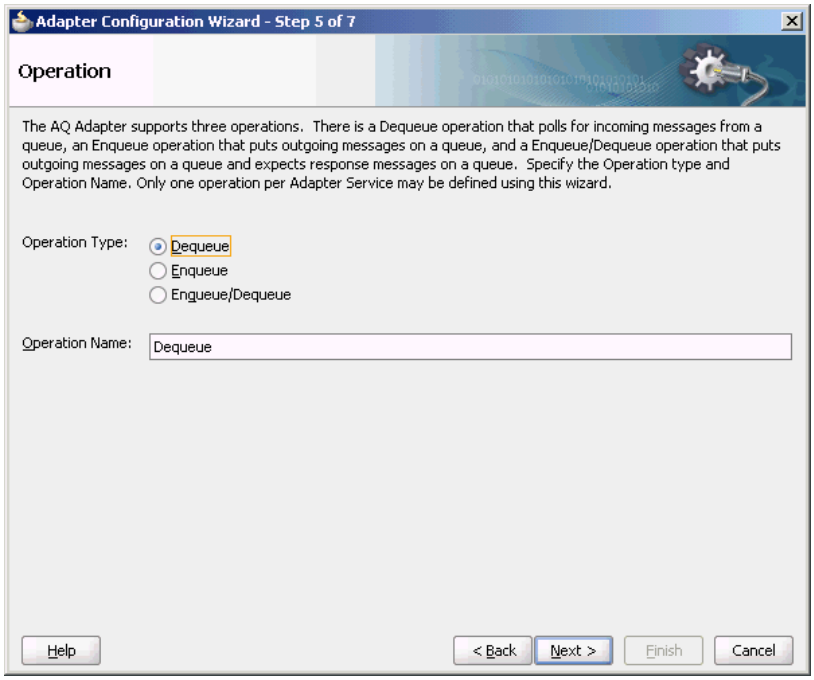
   The Operation page is displayed.

8. In the Operation page, select **Dequeue**, as shown in Figure 7-58.

9. Accept the default operation name, and click **Next**.

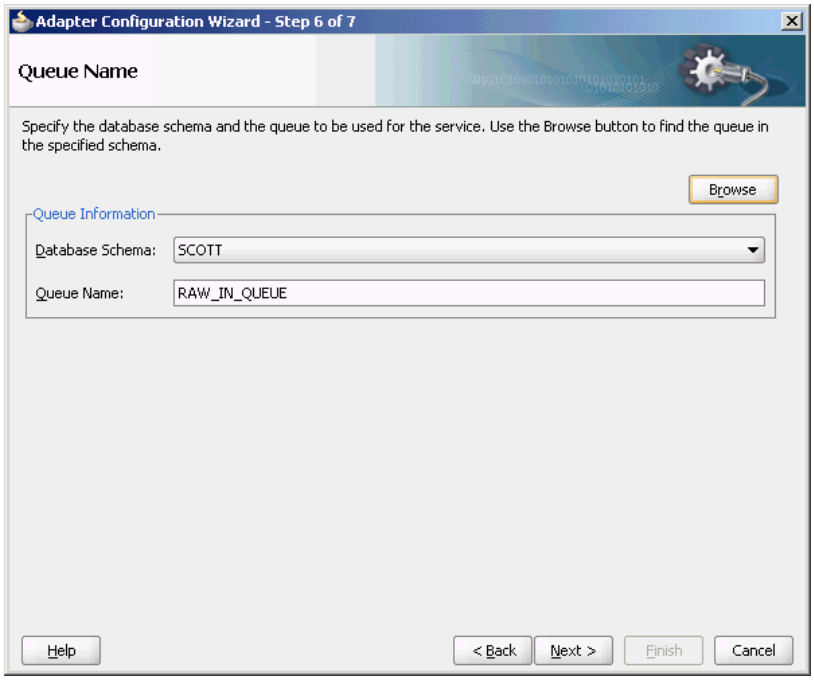
   The Queue Name page is displayed.

Figure 7-58 The Adapter Configuration Wizard Operation Page



10. In the Queue Name page, select **SCOTT** as Database Schema and **RAW_IN_QUEUE** as Queue Name, as shown in Figure 7-59.

Figure 7-59 The Adapter Configuration Wizard Queue Name Page



11. Click **Next**.

    The Queue Parameters page is displayed.

12. Enter the Correlation ID and a Dequeue condition, and then click **Next**.

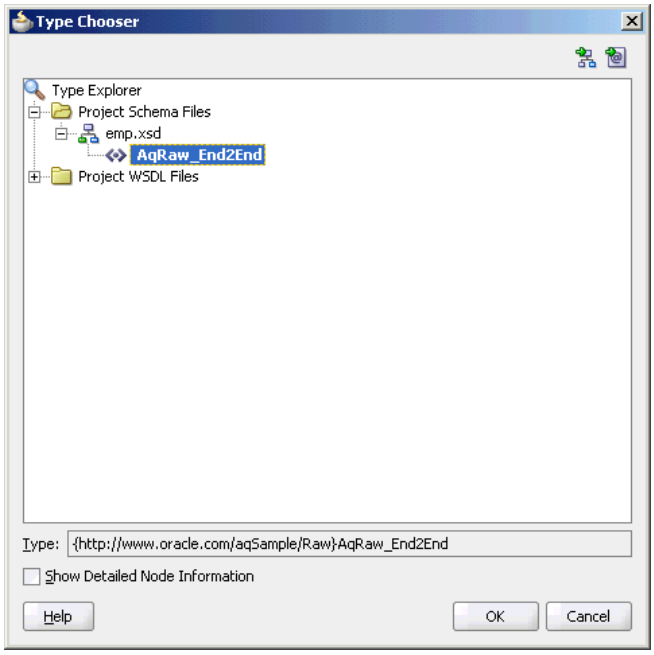
    The Messages page is displayed.

13. Click **Browse** at the end of the URL field.

    The Type Chooser dialog is displayed.

14. Select **Project Schema Files**, **emp.xsd**, and then **AQRaw_End2End**, as shown in Figure 7-60.

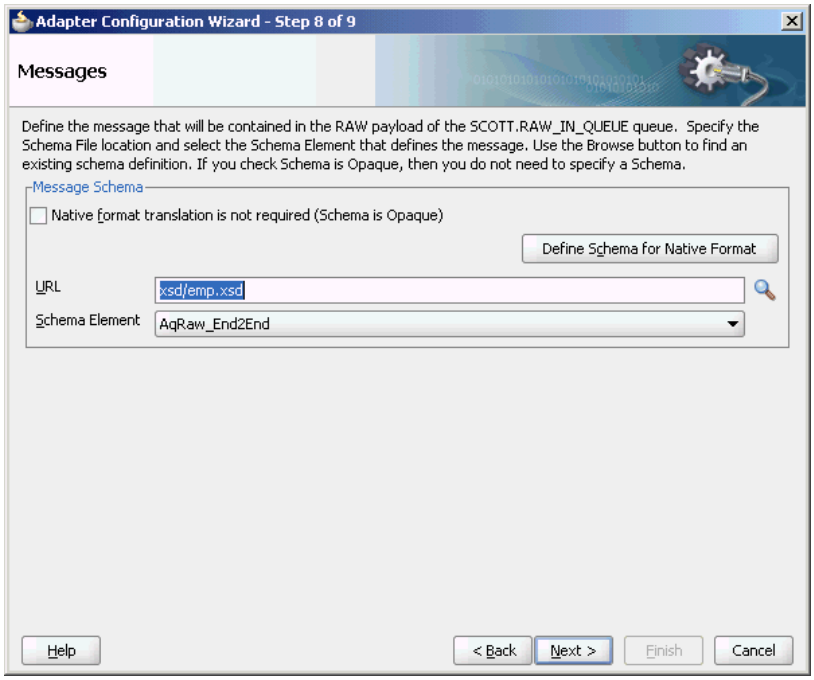
Figure 7-60 The Type Chooser Dialog



15. Click **OK**.

    The `emp.xsd` schema file is displayed in the URL field in the Message dialog, as shown in Figure 7-61.

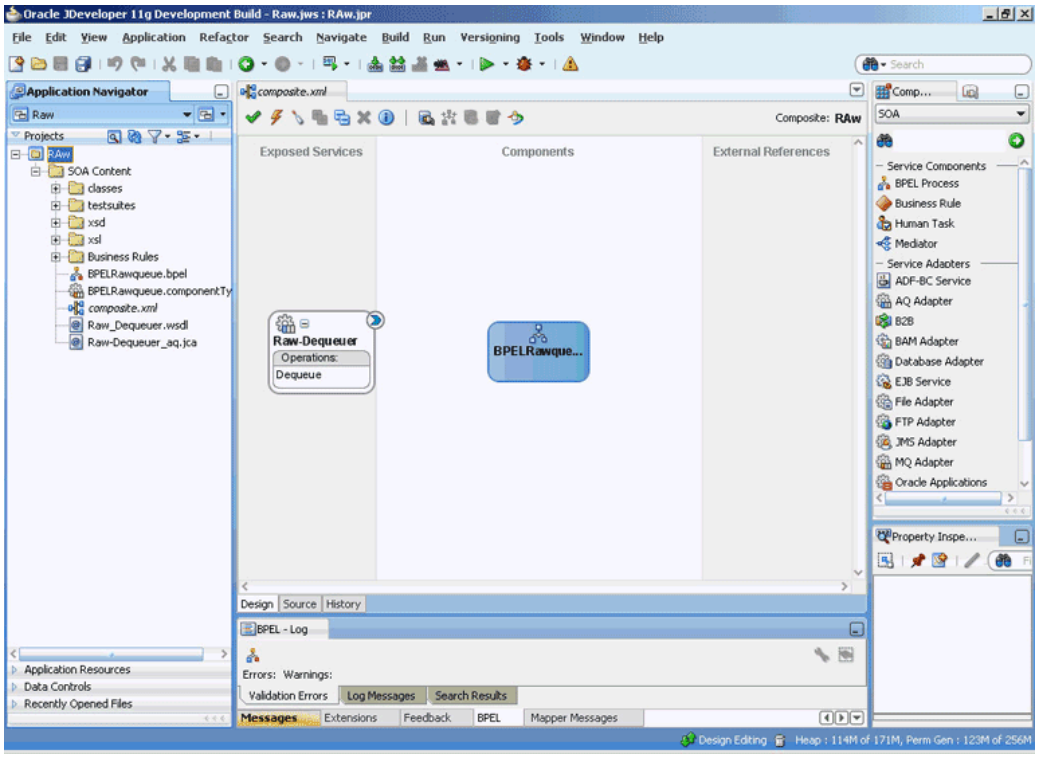
Figure 7-61 The Adapter Configuration Wizard Messages Page



16. Click **Next**. The Finish page is displayed.

17. Click **Finish**. You have configured the Oracle AQ Adapter service, and the `composite.xml` page is displayed, as shown in Figure 7-62.

Figure 7-62 The JDeveloper Window Composite.xml Page



Creating an Outbound Adapter Service

Perform the following steps to create an adapter service that enqueues the request messages and dequeue the corresponding response messages (report) from a queue:

1. Drag and drop **AQ Adapter** from the Service Adapters list in the Components window to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard Welcome page is displayed.

2. Click **Next**. The Service Name page is displayed.

3. Enter `Raw-Enqueuer` in the **Service Name** field, and click **OK**.

   The Service Connection page is displayed.

4. Select **XA Datasource**, and then click **Next**.

   The Operation page is displayed.

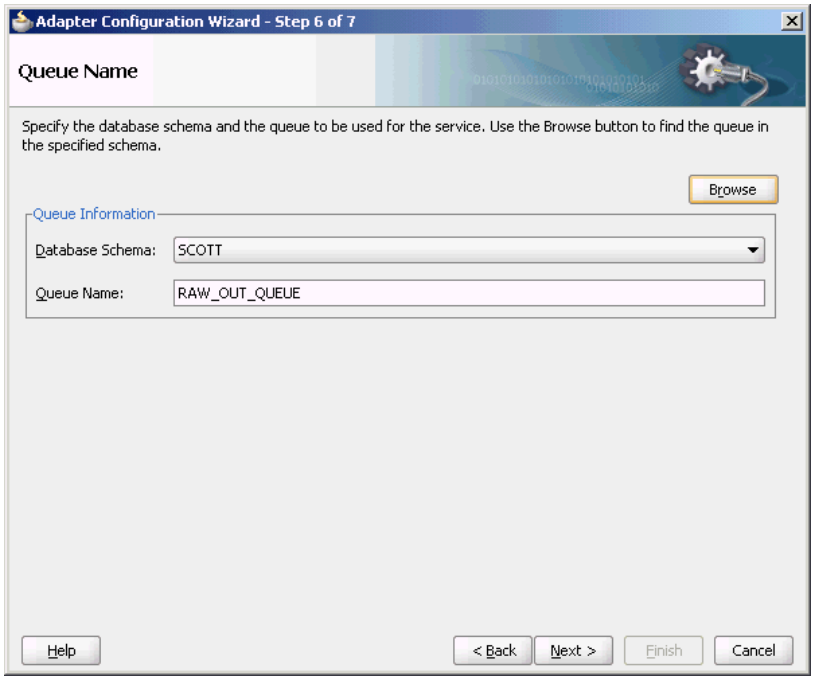
5. Select **Enqueue**.

6. Accept the default operation name, and click **Next**.

   The Queue Name page is displayed.

7. Select **SCOTT** as Database Schema and **RAW_OUT_QUEUE** as Queue Name, as shown in Figure 7-63.

Figure 7-63 The Adapter Configuration Wizard Queue Name Page



8. Click **Next**.

   The Queue Parameters page is displayed.

9. Enter the Correlation ID, and then click **Next**.

   The Messages page is displayed.

10. Click **Browse for schema file** at the end of the URL field.

    The Type Chooser dialog is displayed.

11. Select **Project Schema Files**, **emp.xsd**, and **AQRaw_End2End**, as shown in Figure 7-60.

12. Click **Next**.

    The emp.xsd schema file is displayed in the URL field in the Message dialog, as shown in Figure 7-61.

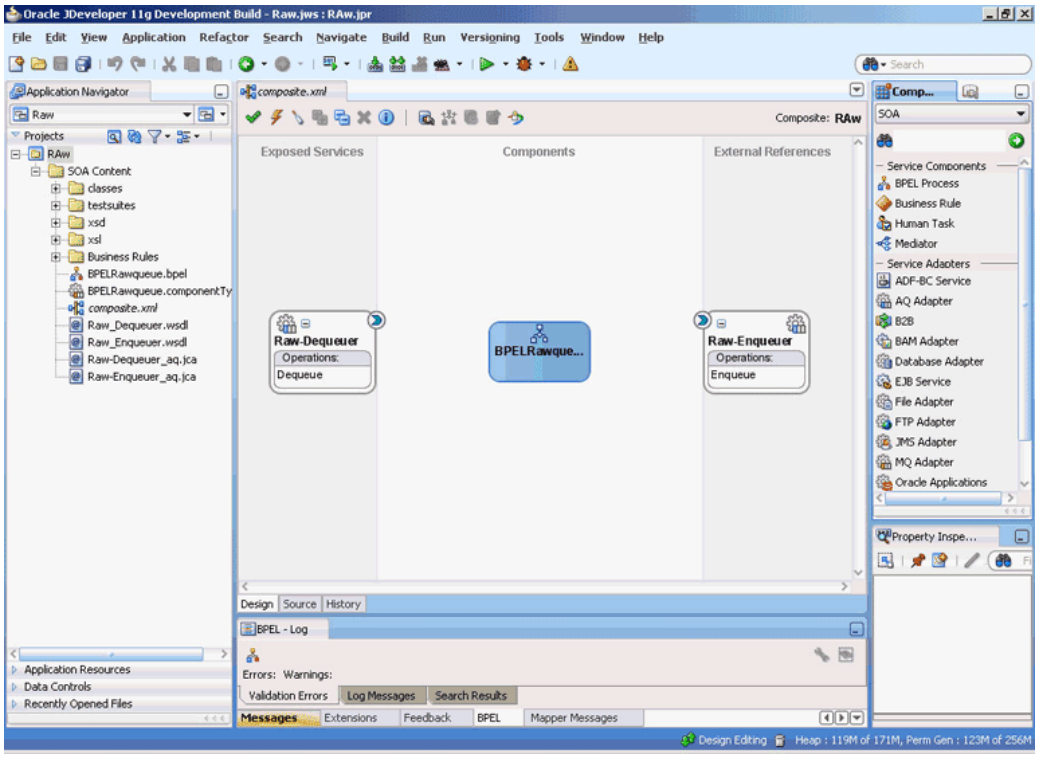
13. Click **Next**.

    The Finish page is displayed.

14. Click **Finish**.

    You have configured the Oracle AQ Adapter service, and the composite.xml page is displayed, as shown in Figure 7-64.

Figure 7-64 The JDeveloper Window Composite.xml Page



Wiring Services and Activities

You must assemble or wire the three components that you have created: Inbound adapter service, BPEL process, and Outbound adapter reference. Perform the following steps to wire the components:

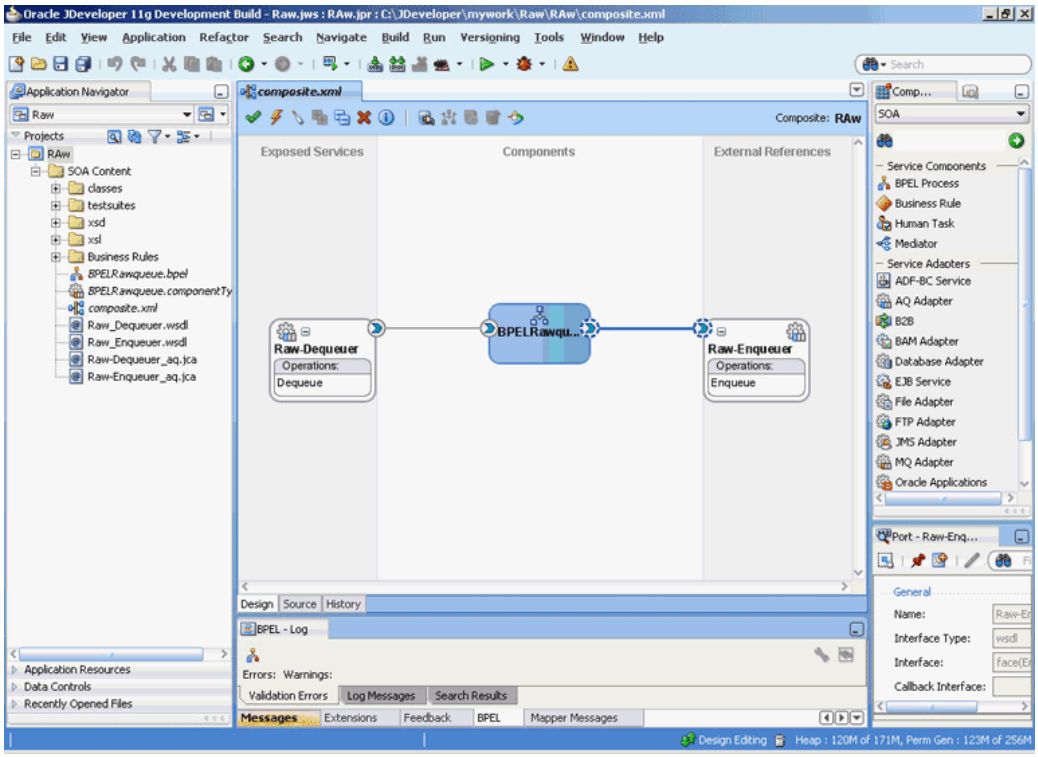
1. Drag the small triangle in the Raw-Dequeuer in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.

2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in Raw-Enqueuer in the External References area.

   Similarly, drag the small triangle in the BPEL process in the Components area to the drop zone in OutboundService in the External References.

   The JDeveloper `composite.xml` file is displayed, as shown in Figure 7-65.

Figure 7-65 The JDeveloper- Composite.xml

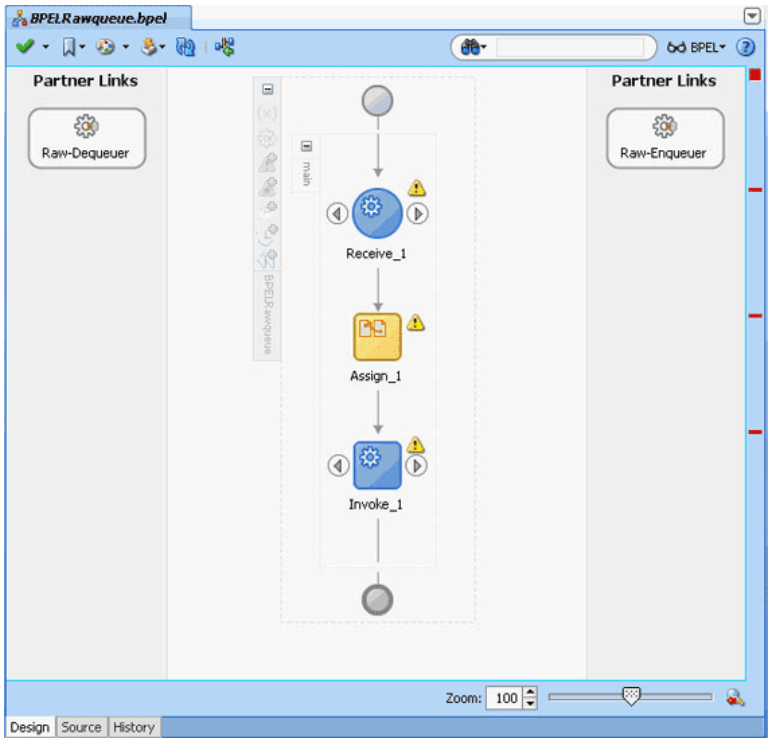


3. Click **File**, **Save All**.
4. Double-click **BPELRawqueue**.

   The BPELRawqueue.bpel page is displayed.
5. Drag and drop the **Receive**, **Assign**, and **Invoke** activities in the order mentioned, from the Components window to the Components area.

   The JDeveloper `BPELRawqueue.bpel` page is displayed, as shown in Figure 7-66.

Figure 7-66 The BPELRawqueue.bpel Page



6. Double-click the **Receive** activity.

   The Receive dialog is displayed.

7. Click the **Browse Partner Links** icon at the end of the Partner Link field.

   The Partner Link Chooser dialog is displayed.

8. Select **Raw-Dequeuer**, and then click **OK**.

   The Receive dialog is displayed with the Partner Link field populated with the value Raw-Dequeuer.

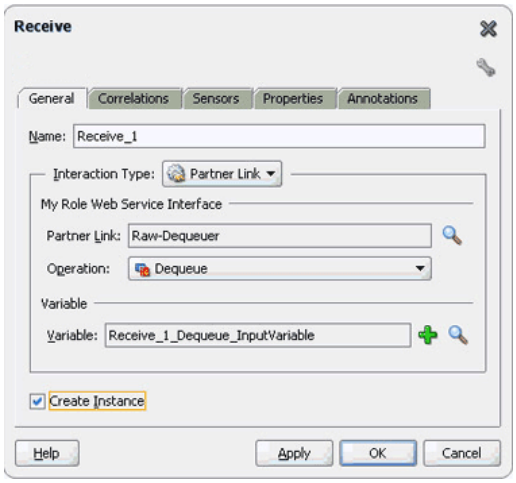
9. Click the **Auto-Create Variable** icon that is displayed at the end of the Variable field.

   The Create Variable dialog is displayed.

10. Accept the default values, and click **OK**.

11. Check the **Create Instance** box, as shown in Figure 7-67, and click **OK**.

Figure 7-67 The Receive Dialog



12. Double-click the **Invoke** activity.

    The Invoke dialog is displayed.

13. Click the **Browse Partner Links** icon at the end of the Partner Link field.

    The Partner Link Chooser dialog is displayed.

14. Select **Raw-Enqueuer**, and then click **OK**.

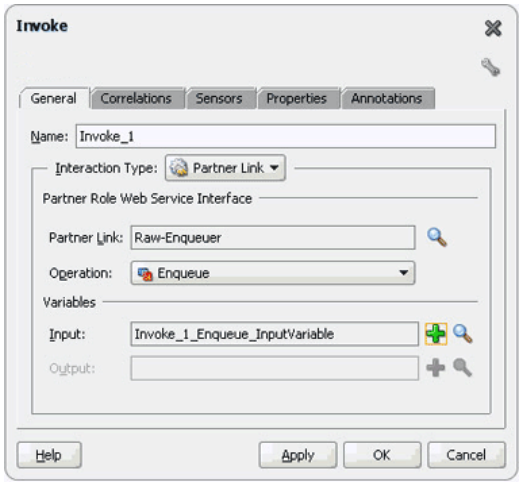
    The Invoke dialog is displayed with the Partner Link field populated with the value Raw-Enqueuer.

15. Click the **Automatically Create Input Variable** icon that is displayed at the end of the Input Variable field.

16. Accept the default values, and click **OK**.

    The Invoke dialog is displayed, as shown in Figure 7-68.

Figure 7-68 The Invoke Dialog



17. Click **OK**.

18. Double-click the **Assign** activity.

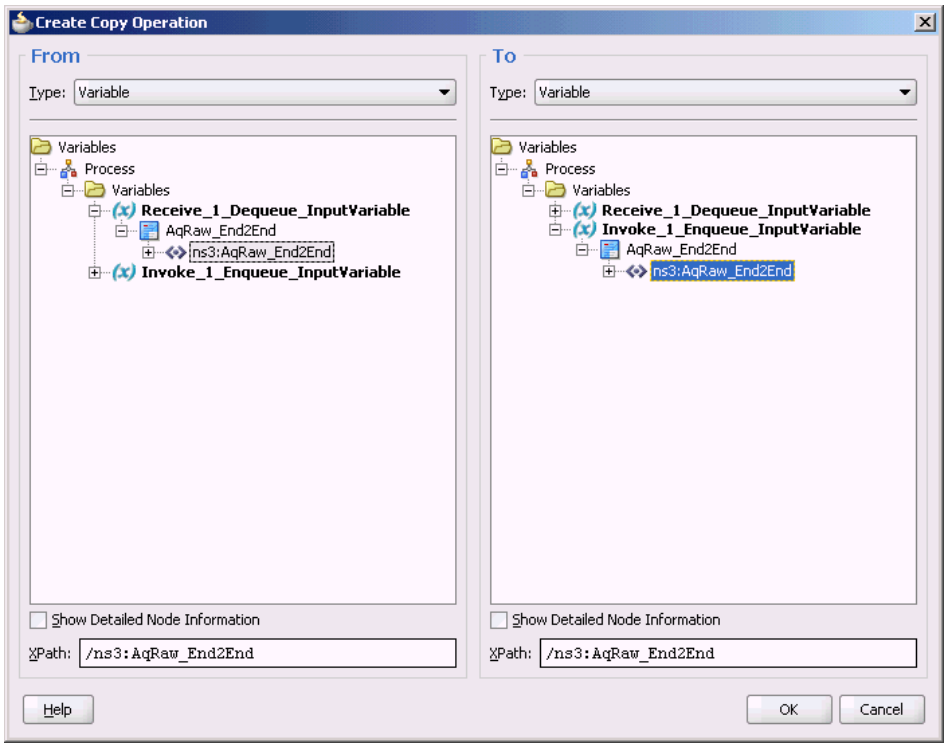
The Assign dialog is displayed.

19. Click the plus icon, and select **Copy Operation**.

    The Create Copy Operation dialog is displayed.

20. Select the variables, as shown in Figure 7-69, and click **OK**.

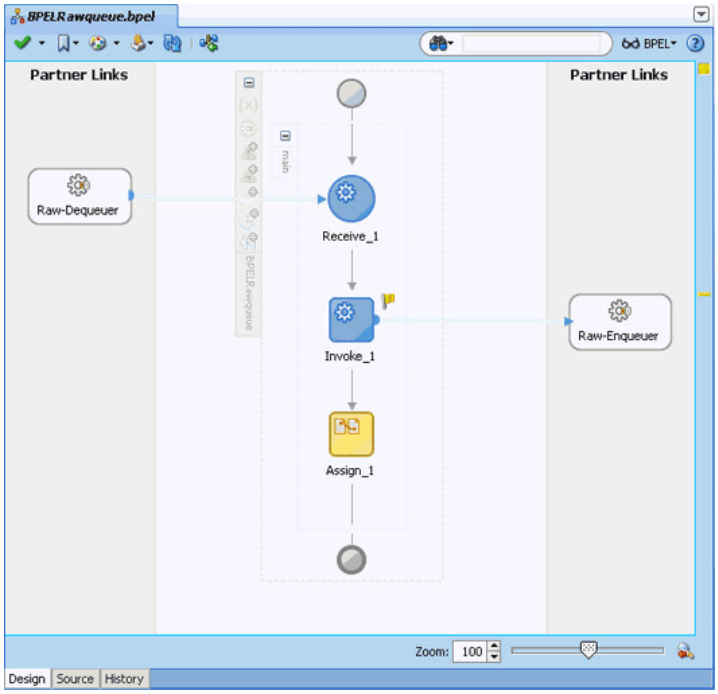
Figure 7-69 The Create Copy Operation Dialog



21. Click **OK** in the Assign dialog.

    The JDeveloper `BPELRawqueue.bpel` page is displayed, as shown in Figure 7-70.

Figure 7-70 The BPELRawqueue.bpel Page



22. Click **File**, **Save All**.

Configuring the Data Sources in the Oracle WebLogic Server Administration Console

1. Navigate to `http://servername:portnumber/console`.
2. Use the required credentials to open the Home page of the Oracle WebLogic Server Administration Console.
3. In the Home page, under Domain Structure, select **Services**, **JBDC**, and then click **DataSources**.

   The Summary of JDBC Data Sources page is displayed.
4. Click **New**. The Create a New JDBC Data Source page is displayed.
5. Enter the values for the properties to be used to identify your new JDBC data source.
6. Click **Next**. The Create a New JDBC Data Source Transaction Options page is displayed.
7. Click **Next**. The Create a New JDBC Data Source Connection Properties page is displayed.
8. Enter the connection properties in the Connection Properties page.
9. Click **Next**. The Create a New JDBC Data Source Test Database Connection page is displayed.
10. Click **Test Configuration** to test the database availability and the connection properties you provided. A message stating that the connection test succeeded is displayed at the top of the Create a New JDBC Data Source Test Database Connection page.

11. Click **Next**. The Create a New JDBC Data Source Select Targets page is displayed.
12. Select a target, and then click **Finish**.

    The Summary of JDBC Data Sources page is displayed. This page summarizes the JDBC data source objects that have been created in this domain. The Data Source that you created is displayed in this list.
13. Close the Oracle WebLogic Server Administration Console.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps.

The following are the steps to deploy the application profile using JDeveloper:

1. Create an application server connection by using the procedure described in Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application by using the procedure described in Deploying Oracle JCA Adapter Applications from .

Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed composite by using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`.

   The composite you deployed is displayed in the Application Navigator.
2. In the Last 5 Instances pane, there is an entry of a new instance.

   This is the instance that triggered when you enqueued a message.
3. Click an instance.

   The Flow Trace page is displayed.
4. Click the **BPELRawqueue** component instance.

   The Audit page is displayed.
5. Click the **Flow-Debug** tab to debug the instance.

8

Oracle JCA Adapter for JMS

This chapter describes how to use the Oracle JCA Adapter for JMS (Oracle JMS Adapter), which enables an Oracle BPEL process or an Oracle Mediator component to interact with Java Messaging Service.

This chapter includes the following topics:

Introduction to the Oracle JMS Adapter

The JMS architecture uses one client interface-to-many messaging servers. The JMS model has two messaging domains, point-to-point and publish-subscribe. In the point-to-point domain, messages are exchanged through a queue and each message is delivered to only one receiver. In the publish-subscribe model, messages are sent to a topic and can be read by many subscribed clients.

You can obtain JMS adapter sample files by accessing theOracle SOA Sample Code site.

This section includes the following topics:

Oracle JMS Adapter Integration with the Oracle BPEL Process Manager

The JCA Binding Component is used for the bidirectional integration of the JCA 1.5 resource adapters with BPEL Process Manager. The JCA Binding Component is based on standards and employs the Web service Invocation Framework (WSIF) technology for exposing the underlying JCA interactions as Web services.

For information on Oracle JMS Adapter architecture, adapter integration with Oracle BPEL Process Manager (Oracle BPEL PM), and adapter deployments, refer to Adapter Integration with Components .

Oracle JMS Adapter Integration with Oracle Mediator

Mediator supports Oracle JCA Adapters and enables you to define inbound and outbound adapter services for each. An inbound adapter service receives data from an external messaging system and transforms it into an XML message. An outbound

adapter service sends data to a target application by transforming an XML message into the native format of the given adapter.

In the case of the Oracle JMS Adapter service, by using Mediator, you can send or receive messages from a JMS queue or topic.

Oracle BPEL PM pre-dates Mediator, and most of this guide and the samples implicitly assume use with Oracle BPEL PM. However, the adapters work equally well with either Oracle BPEL PM or Mediator. For any mention of Oracle BPEL PM in this chapter, you may substitute Mediator, instead.

Oracle JMS Adapter Features

The Oracle JMS Adapter includes the following features:

- **Is based on JMS version 1.0.2b**
- **Is a generic Oracle JMS Adapter**

  Should work with any JMS provider. However, Oracle JMS Adapter is certified against AQ JMS (JMS providers OJMS 8.1.7, 9.0.1.4, and 9.2), TIBCO JMS, IBM Websphere MQSeries (IBM MQSeries JMS 6.0), Weblogic JMS, Apache, and Active MQ. Contact Oracle Support for additional Providers certification information.
- **Supports JMS topics and queues**
- **Supports byte, text, and map message types**.

  Supports these data types only for this release. The Adapter Configuration Wizard provides the Native Format Builder wizard for consuming native data payloads at runtime. The Native Format Builder wizard creates XSD definitions for the underlying native data.
- **Supports JMS headers and properties**
- **Supports WebLogic Server Unit-of-Order feature**

  The WebLogic Server Unit-of-Order feature enables a JMS message producer or group of message producers acting as one, to group messages into a single unit that is processed sequentially in the order the messages were created. The message processing of a single message is complete when a message is acknowledged, committed, recovered, or rolled back. Until message processing for a message is complete, the remaining unprocessed messages for that Unit-of-Order are blocked.

  This enhancement enables WebLogic Server Unit-of-Order support in SOA JMS adapter. Messages produced using the SOA JMS adapter would enable the user to specify a specific unit-of-order.
- **Supports jca.message.encoding property**

  The Oracle JMS Adapter supports the `jca.message.encoding` property for inbound and outbound payloads. If the `jca.message.encoding` property is used along with the `adapter.jms.encoding` property and the `nxsd:encoding` attribute, then the `jca.message.encoding` property takes precedence over the `adapter.jms.encoding` property, and the `nxsd:encoding` attribute is given the last preference. The `nxsd:encoding` value can be UTF-8, which is typically recommended for interoperability and Unicode support. However, you can specify any legal encoding supported by the Java runtime environment. For a complete listing of supported encodings, visit `http://docs.oracle.com/javase/7/docs/technotes/guides/intl/encoding.doc.html`. You can specify the

encoding in the (N)XSD associated with the adapter proxy meta data. For example, you can specify the following attribute, `nxsd:encoding="iso-8859-1".`

The `jca.message.encoding` property is supported as an endpoint defined in `composite.xml` You can define this property using the Properties tab of the Adapter Configuration Wizard or by editing the `composite.xml` file. The `jca.message.encoding` property can be passed as a normalized message property for both inbound and outbound interactions.

The following code snippet is an example of setting values in the composite.xml file for message encoding for an inbound service:

```
 <service name="jms_inbound" ui:wsdlLocation="jms_inbound.wsdl">
    <interface.wsdl
   interface="http://xmlns.oracle.com/pcbpel/adapter
              /jms/utf8/jcamessageencoding/
     jms_inbound#wsdl.interface(Consume_Message_ptt)"/>
    <binding.jca config="jms_inbound_jms.jca">
    <property name="jca.message.encoding"
                 type="xs:string" many="false"
                override="may">GBK</property>
  </binding.jca>
</service>
```

The following code snippet is an example of setting values in the composite.xml file for message encoding for an outbound reference:

```
<reference name="jms_outbound" ui:wsdlLocation=
         “jms_outbound.wsdl">
    <interface.wsdl
     interface="http://xmlns.oracle.com/pcbpel/adapter/jms/utf8/
jcamessageencoding/
     jms_outbound#wsdl.interface(Produce_Message_ptt)"/>
    <binding.jca config="jms_outbound_jms.jca">
    <property name="jca.message.encoding" type="xs:string" many="false"
                override="may">GBK</property>
  </binding.jca>
</reference>
```

- **Supports the JMS message selector**

  Supports the JMS message selector for performing filtering while subscribing to JMS topics and queues. This parameter is based on the SQL 92 language for filtering messages based on fields present in the JMS header and properties section.

- **Is DOM2 compliant**

  The Oracle JMS Adapter can process and generate document objects that are compliant with DOM2 specification.

- **Supports normalized message.**

  Header manipulation and propagation is a key business integration messaging requirement. Oracle BPEL PM, Mediator, Oracle JCA, and Oracle B2B rely extensively on header support to solve customers' integration requirements. For example, a user can preserve a file name from the source directory to the target directory by propagating it through message headers. Another example: the outbound Oracle JMS Adapter facilitates asynchronous request/response by propagating the `correlationId` and the `JMSReplyTo` address as JMS headers. In Oracle BPEL Process Manager and Mediator, users can access, manipulate, and set headers with varying degrees of UI support.

For more information, see Correlation Support Within Adapters.

Propagating Headers in a Normalized Message:

Normalized Message is simplified to have only two parts, properties and payload.Typically, properties are name-value pairs of scalar types. To fit the existing complex headers into properties, they are flattened into scalar types.

Manipulating Headers in Design-Time:

The user experience while manipulating headers in design time is simplified, because the complex properties are predetermined. In Mediator, or Oracle BPEL designer, you can manipulate the headers with some reserved key words. For example, in Mediator designer, you can access an inbound Oracle File Adapter, `fileName` header by using the following expression:

```
$nmproperty.InboundFileHeaderType.fileName
```

However, this method does not address the properties that are dynamically generated based on your input. For example, in the Oracle AQ Adapter Wizard, you are allowed to propagate some of the fields from an AQ object as headers. Based on this user choice, the header definitions are generated. These definitions are not predetermined and hence cannot be accounted for in the list of predetermined property definitions. The user cannot design header manipulation of the dynamic properties before they are defined. To address this limitation, you must generate all the necessary services (composite entry points) and references. This restriction applies only to those services that are expected to generate dynamic properties. Once dynamic properties are generated, they must be captured in some given location for each composite. Only then can the user manipulate the dynamic properties in the Oracle Mediator or Oracle BPEL designer.

- **Supports specifying a durable JMS subscriber**
- **Supports persistent and nonpersistent modes of a JMS publisher**
- **Does not support outbound retry functionality for AQJMS on Solaris**

Note:

When you use the Oracle JMS Adapter to connect to an AQ-JMS provider, and if the database that hosts the AQ destination is 10.1.0.4, then the adapter retry mechanism on the outbound direction fails to reconnect to the database server if the database server goes down. This is because of a client JDBC issue with `ojdbc14.jar`. To resolve this you must download the 10.1.0.4 JDBC drivers and use them in the mid tier by replacing the libraries, specifically ojdbc14.jar in `$MIDTIER_ORACLE_HOME/jdbc`. For a detailed explanation about how to resolve this issue, refer to Metalink Note 317385.1.

- The JMS API specifies three types of acknowledgments that can be sent by the JMS publisher:
  - `DUPS_OK_ACKNOWLEDGE`, for consumers that are not concerned about duplicate messages
  - `AUTO_ACKNOWLEDGE`, in which the session automatically acknowledges the receipt of a message

  - `CLIENT_ACKNOWLEDGE`, in which the client acknowledges the message by calling the message's acknowledge method

- **Supports tracking message size**

  The Oracle JMS Adapter is message size aware. The Oracle JMS Adapter calculates the message size and reports the size back to the JCA Binding Component. The API, related to size, exposed by the JCA Binding Component can be used for reporting purposes.

- **Configuring MapMessage Support in JMS Adapter MapMessage Data Type**

  A `MapMessage` is used to send a set of name-value pairs where names are strings and values are Java primitive types. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined. It inherits from a message and adds a map message body.

  Oracle JMS adapter provides support for processing `MapMessage`. It now supports one new ActivationSpec and InteractionSpec property, namely `JmsMapMessageConsumeActivationSpec` and `JmsMapMessageProduceInteractionSpec`.

  The following use cases are supported:

  - `Use Case 1.` Entire JMS MapMessage object is processed as payload.

    If both PayloadEntry and AttachmentList properties are not defined, the entire JMS MapMessage is converted to XML and the XML file is transferred as the payload. For this use case, both `PayloadEntry` and `AttachmentList` properties are optional. The following schema is used for conversion:

```
<schema targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/jms/MapMessage"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/jms/MapMessage"
        elementFormDefault="qualified">
 <element name="MapMessage">
  <complexType>
   <choice maxOccurs="unbounded">
    <element name="entry">
     <complexType>
      <simpleContent>
       <extension base="string">
        <attribute name="name" type="string"/>
        <attribute name="dt" type="string"/>
       </extension>
      </simpleContent>
     </complexType>
    </element>
   </choice>
  </complexType>
 </element>
    </schema>
```

    The attribute `name` is the name part of the name value pair for a JMS MapMessage entry.

    The attribute `dt` can have one of the following string values.

    - Boolean
    - Byte

- Short
- Integer
- Long
- Float
- Double
- String
- Char
- ByteArray
- **Use Case 2.** A MapMessage entry (name-value pair) is processed as payload

The `PayloadEntry` property specifies that the JMS MapMessage entry name (referring to the name-value pair that should be processed) to be is used as the payload. You have the option to send the payload as an attachment if the `AttachmentList` property is defined instead of `PayloadEntry`. For this use case, either `PayloadEntry` or `AttachmentList` property should be specified but not both.

All other MapMessage entries are converted to adapter properties identified by `jca.jms.Map.xxxx`, where `xxxx` is name of the JMS MapMessage entry (name-value pair).

For the JMS MapMessage sample, see the following samples by accessing the latest SOA Sample code under **Middleware & Tools** from Sample Code for Developers and Admins page: adapters-jms-104-wlsjms-mapmessage.

- **Supports Enterprise Information System (EIS) Credentials**

  The Oracle JMS Adapter supports securing of the Enterprise Information System (EIS) credentials such as the user name and password, whenever it establishes an outbound connection with EIS. You can secure the user name and password for Oracle JMS Adapter by using Oracle WebLogic Server container-managed sign-on.

  For more information about support for securing of the Enterprise Information System (EIS) credentials, see Securing Enterprise Information System Credentials.

- **Supports Streaming Large Payload**

  Oracle JMS Adapter provides support to stream payload. When you enable this feature, the payload is streamed to a database instead of getting manipulated in the SOA runtime as in a memory DOM. This feature can be used while handling large payloads. To enable support to stream payload, ensure that you select the **Enable Streaming** check box while defining the consume operation parameters on the Consume Operation Parameters page in Oracle JDeveloper. When the **Enable Streaming** check box is selected, a corresponding Boolean property `EnableStreaming` is appended to the `ActivationSpec` properties defined in the respective `.jca` file, as shown in the following example. If the `EnableStreaming` property does not exist, then the default value of false is assumed.

```
<activation-spec className="oracle.tip.adapter.jms.inbound.
                 JmsConsumeActivationSpec">
  <property name="DestinationName" value="jms/DemoInQueue"/>
  <property name="UseMessageListener" value="false"/>
```

```
  <property name="PayloadType" value="TextMessage"/>
  <property name="EnableStreaming" value="true"/>
</activation-spec>
```

- **Supports Transactions**

  A transaction enables an application to coordinate a group of messages for production and consumption, treating messages sent or received as a single unit. When an application commits a transaction, all messages it received within the transaction are removed by the JMS provider. The messages it sent within the transaction are delivered as one unit to all JMS consumers. If the application rolls back the transaction, then the messages it received within the transaction are returned to the messaging system and the messages it sent are discarded. The Oracle JMS Adapter supports JMS transactions. A JMS-transacted session supports transactions that are located within the session. A JMS-transacted session's transaction does not have any effects outside of the session.

- **Supports Error Handling**

  For information about error handling, refer to Error Handling.

- **Supports Multiple Consumer Threads**

  The Oracle JMS Adapter supports an activation endpoint property, `adapter.jms.receive.threads`. Setting this property in composite.xml is a preferred way to spawn multiple poller threads for the inbound message flow between the adapter and the Enterprise Information System (EIS). This helps improve performance because multiple poller threads dequeue messages in a round robin fashion; this assists in Distributed scenarios as well.

- **Supports Performance Tuning**

  The Oracle JMS Adapter supports performance tuning. For more information, see Oracle JCA Adapter Tuning Guide and Oracle JCA Adapter Properties.

Note:

Oracle JMS Adapter cannot be used programmatically inside an EJB or JMS client.

Oracle JMS Adapter Concepts

Messaging is any mechanism that enables communication between programs. Messages are structured data that one application sends to another. Message-oriented middleware (MOM) is an infrastructure that supports scalable enterprise messaging. MOM ensures fast, and reliable asynchronous communication, guaranteed message delivery, receipt notification, and transaction control.

JMS is a Java interface developed by Sun Microsystems for producing, sending, and receiving messages of an enterprise messaging system. JMS is an API that JMS vendors implement. Oracle provides two implementations of JMS, WLS JMS and Oracle JMS based on Oracle advanced queues. A JMS producer creates JMS messages and a JMS consumer consumes JMS messages.

JMS supports two messaging paradigms, point-to-point (queues) and publish/subscribe (topics).

This section includes the following topics:

Point-to-Point

In point-to-point messaging, the messages are stored in a queue until they are consumed. One or more producers write to the queue and one or more consumers extract messages from the queue. The JMS consumer sends an acknowledgment after consumption of a message; this results in purging of the message from the queue.

Publish/Subscribe

In publish/subscribe messaging, producers publish messages to a topic, and the consumer subscribes to a particular topic. Many publishers can publish to the same topic, and many consumers can subscribe to the same topic. All messages published to the topic by the producers are received by all consumers subscribed to the topic. By default, subscribers receive messages only when the subscribers are active. However, JMS API supports durable subscriptions that ensure that consumers receive messages that were published even when the subscribers are not up and running. The durable subscription involves registering the consumer with a unique ID for retrieving messages that were sent when the consumer was inactive. These messages are persisted by the JMS provider and are either sent to the consumer when it becomes active again or purged from storage if the message expires. The JMS producer can be set either to persistent or nonpersistent mode. The messages are not persisted in the latter case and can be used only for nondurable subscriptions.

For scenarios that requires you to work with durable subscriptions on Oracle WebLogic Server, a connector factory with `ClientID` property defined is required, as shown in the following example:

```
<FactoryProperties>ClientID=uniquename</FactoryProperties>
```

When defining multiple durable subscriber it would entail you to define multiple connector factory each with a unique `ClientID` property specified. You must take care to not use the same connector factory for any other adapter interaction (such as outbound interaction if it is used for processing inbound messages) because Oracle WebLogic Server allows a `clientid` to be bound only once. For a scenario in which a connector factory with `ClientId` defined is used on the inbound to process incoming messages a different connector factory should be used for the outbound adapter interactions.

Note:

You must manually remove durable subscribers that are not used by the BPEL partner link. Oracle JMS Adapter does not automatically remove these durable subscriptions.

The JMS API supports both synchronous and asynchronous communication for message consumption. In the synchronous case, the consumer explicitly calls the

`receive()` method on the topic or queue. In the asynchronous case, the JMS client registers a message listener for the topic or queue and the message is delivered by calling the listener's `onMessage()` method.

Destination, Connection, Connection Factory, and Session

The destination property contains the addressing information for a JMS queue or topic.Connections represent a physical connection to the JMS provider. The connection factory is used to create JMS connections. A session is used to create a destination, JMS producer, and JMS consumer objects for a queue or topic.

Structure of a JMS Message

The JMS message has a mandatory standard header element, an optional properties element, and an optional standard payload element. The payload can be a text message, byte message, map message, stream message, or object message. The properties element is JMS provider-specific and varies from one JMS provider to another.

Oracle JMS Adapter Header Properties

For information about the Oracle JMS Adapter header properties, see Oracle JMS Adapter Properties.

Connecting with Third-Party Service Providers

JMS Adapter can communicate with TIBCO,IBM MQ Series, and other certified third-party Service Providers. If these service providers are stopped and subsequently restarted, the JMS Adapter can successfully connect to them and process any pending messages.

Perform the following steps to ensure connection occurs correctly:

1. In `setDomainEnv.sh`, a startup script for SOA and the WebLogic Console, set

   ```
   -Dweblogic.transaction.blocking.commit=false
   ```

   and

   ```
   -Dweblogic.transaction.blocking.rollback=false
   ```

2. In the WebLogic Console, proceed to -->**domain** --- > **JTA** ----- > **Advanced** ----> **CompletionTimeoutSeconds**, and set `CompletionTimeoutSeconds` to 0.

Binding

A property,

"adapter.jms.DistributedDestinationConnectionEveryMember" is available as the Service binding property for the JMS Adapter.

This property takes boolean values [true | false]. Default value is "true ".

When true, the JMS Adapter creates a consumer/subscriber for each member of the Distributed Destinations. If set to false, the JMS Adapter creates a consumer/subscriber for only local members of the distributed destination.When the JMS Adapter is connecting to distributed destination on a remote cluster or a server on remote domain, the property

'adapter.jms.DistributedDestinationConnectionEveryMember' should always be set to true. When the JMS Adapter is connecting to distributed destination on a local cluster, the property can be set to either true or false. If set to true, the JMS Adapter behavior remains the same as before (that is, there is a consumer for each Distributed Destination is created). If set to false, the JMS Adapter only creates consumer/subscriber for the local members.

You will not see this property in the JDeveloper User Interface as a binding property on Service side. Consequently, you must manually add it in the composite.xml as shown below under the `<service>/<binding.jca>` tag :

```
<property name="adapter.jms.DistributedDestinationConnectionEveryMember"
type="xs:string" many="false" override="may">false</property>
```

Oracle JMS Adapter Use Cases

This section includes the following topics:

- Configuring Oracle JMS Adapter
- Configuring the Oracle JMS Adapter with TIBCO JMS
- Configuring Oracle JMS Adapter with IBM WebSphere MQ JMS
- WebLogic Server JMS Text Message
- Accessing Queues and Topics from WLS JMS Server in a Remote Oracle WebLogic Server Domain
- Synchronous/Asynchronous Request Reply Interaction Pattern
- AQ JMS Text Message
- Accessing Queues and Topics Created in 11g from the OC4J 10.1.3.4 Server
- Configuring the 11G Server or Later Server to Access Queues Present in 10.1.3.X OC4J .
- Accessing Distributed Destinations (Queues and Topics) on the WebLogic Server JMS .
- Configuring Oracle JMS Adapter with IBM WebSphere Default JMS Provider
- Configuring Request-Reply in the JMS Adapter .
- Using the WLS JMS Unit-Of-Order with the JMS Adapter.

Configuring Oracle JMS Adapter

The following use case demonstrates the procedure for configuring Oracle JMS Adapter and examines the resulting WSDL files and associated `weblogic-ra.xml` files.

This section includes the following topics:

- Creating an Application and a SOA Project
- Using the Adapter Configuration Wizard to Configure
- Generated Files

- weblogic-ra.xml file
- Produce Message Procedure

Creating an Application and a SOA Project

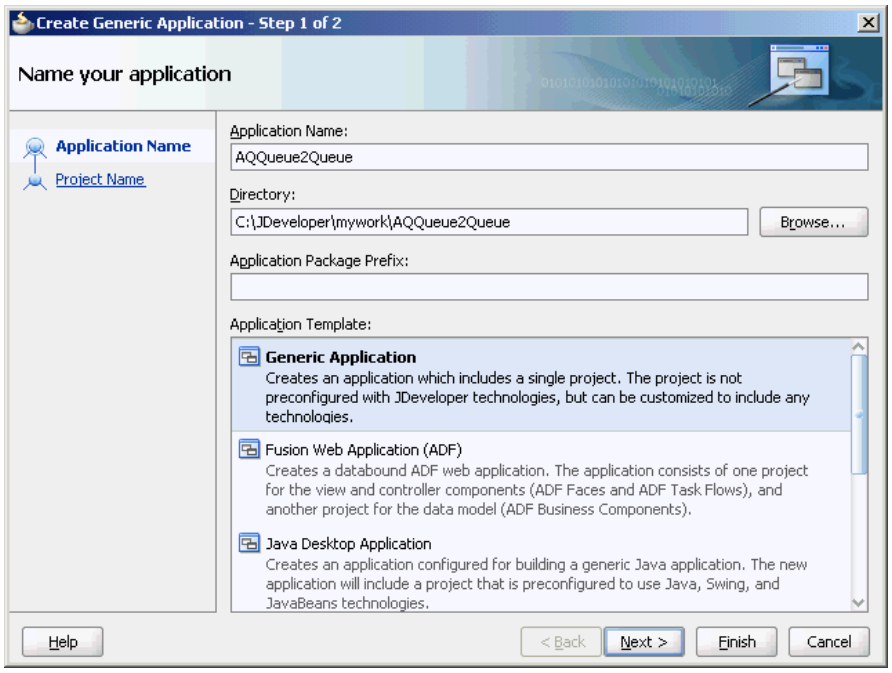
You must first create an JDeveloper application to contain the SOA composite. Use the following steps to create a new application and a SOA project:

1. Open JDeveloper.
2. In the Application Navigator, click **New Application**.

   The Create Generic Application - Name your Application page is displayed, as shown in Figure 8-1.

3. Enter a name for the application in the **Application Name** field. For example, `AQQueue2Queue`.
4. In the Application Template list, choose **Generic Application**.

Figure 8-1 The Create Generic Application - Name your application Page

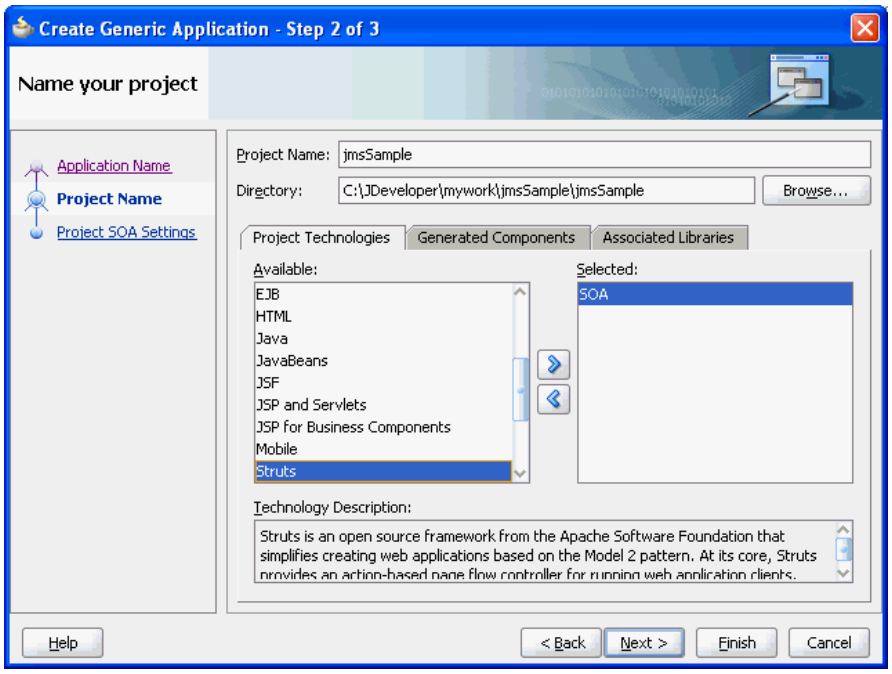


5. Click **Next**.

   The Name your project dialog is displayed, as shown in Figure 8-2.

6. In the **Project Name** field, enter a descriptive name. For example, `AQQueue2Queue`.
7. In the Available list in the **Project Technologies** tab, double-click **SOA** to move it to the Selected list.

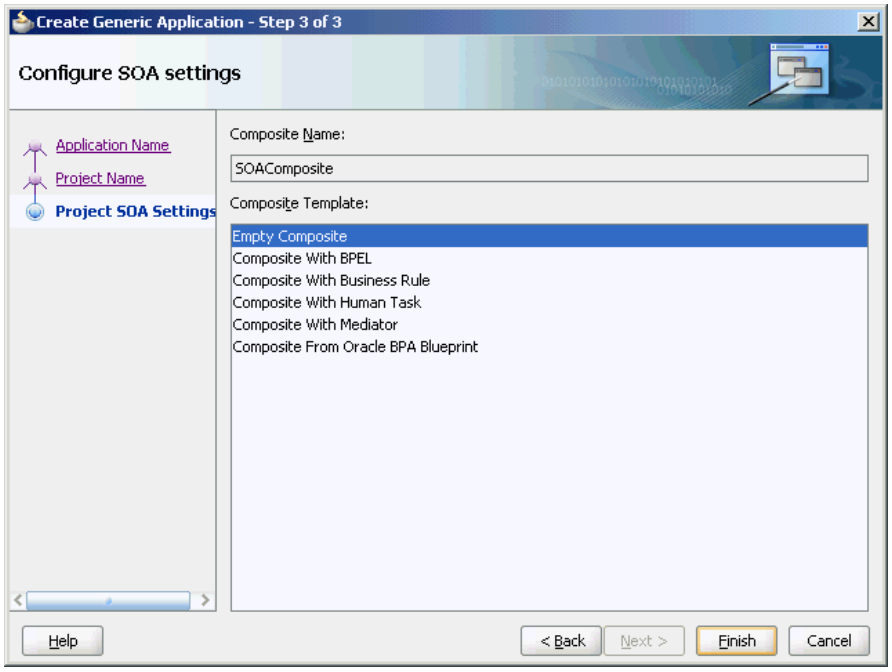
Figure 8-2 The Create Generic Application - Name your Generic project Page



8. Click **Next**.

   The Create Generic Application - Configure SOA settings page is displayed, as shown in Figure 8-3.

Figure 8-3 The Create Generic Application - Configure SOA Settings Page

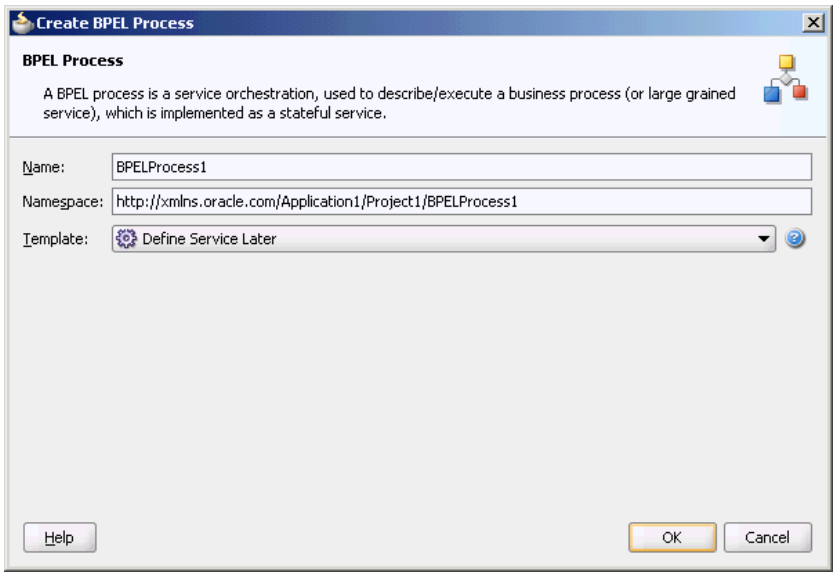


9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

You have created a new application, and an SOA project. This automatically creates an SOA composite.

The Create BPEL Process page is displayed, as shown in Figure 8-4.

Figure 8-4 The Create BPEL Process Page



10. Enter a name for the BPEL process in the **Name** field. In this example, use the default name.
11. Select **Define Service Later** in the Template list, and then click **OK**.

    You have created a BPEL process.

Using the Adapter Configuration Wizard to Configure Oracle JMS Adapter

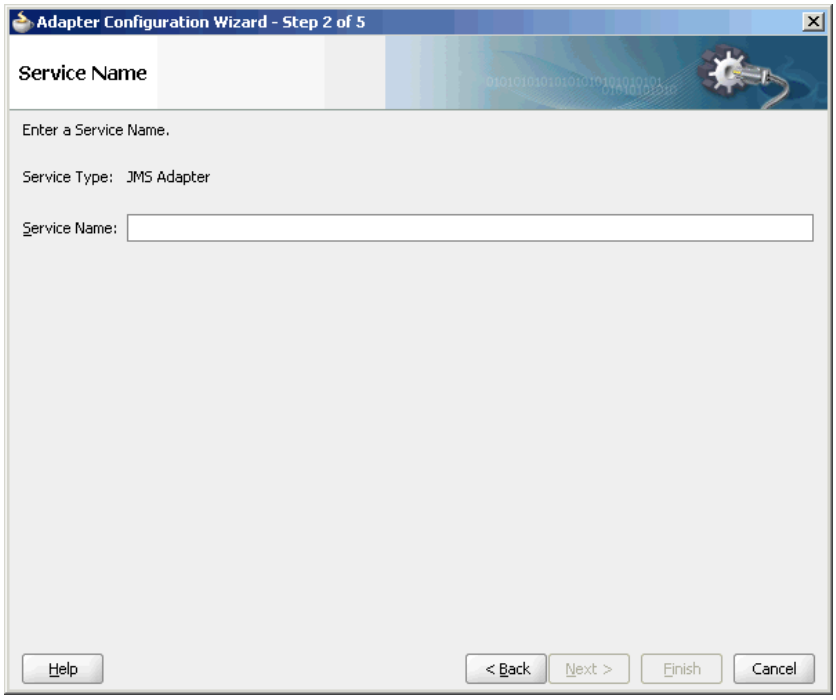
The following are the steps to configure an Oracle JMS Adapter by using the Adapter Configuration Wizard:

1. In the Components window, select **SOA**.
2. Drag and drop **JMS Adapter** from the Service Adapters list to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard is displayed.
3. Click **Next**.

   The Adapter Configuration Wizard - Service Name page is displayed, as shown in Figure 8-5.

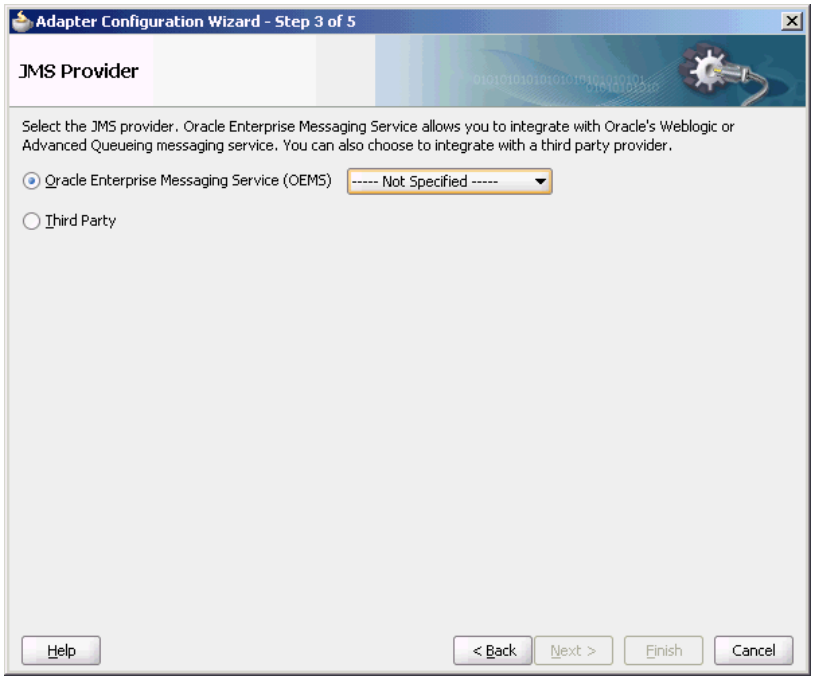
Figure 8-5 Service Name Page



4. Enter a name for the service, and then click **Next**.

   The Adapter Configuration Wizard - JMS Provider page is displayed, as shown in Figure 8-6.

Figure 8-6 The Adapter Configuration Wizard - JMS Provider Page



5. Select any one operation. In this example, select **Oracle Weblogic JMS**.

- **Oracle Enterprise Messaging Service (OEMS)**: This enables you to integrate with the Weblogic service or Advanced Queueing messaging service.
- **Third Party**: Select this option to integrate with a third-party provider.

6. Click **Next**.

   The Adapter Configuration Wizard - Service Connection page is displayed.

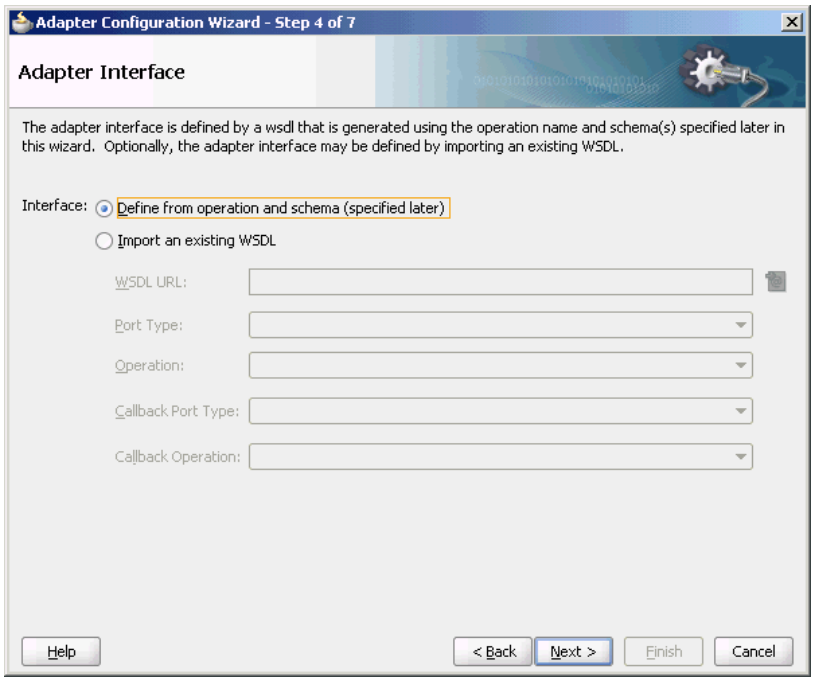
7. You must establish connectivity between the design-time environment and the server you want to deploy it to.

   Perform the steps mentioned in Creating an Application Server Connection for Oracle JCA Adapters to create an application server connection.

8. Click **Next**. The Adapter Interface page is displayed, as shown in Figure 8-7.

9. In the Adapter Interface page, select **Define from operation and schema (specified later)**.

Figure 8-7 The Adapter Configuration Wizard - Adapter Interface Page



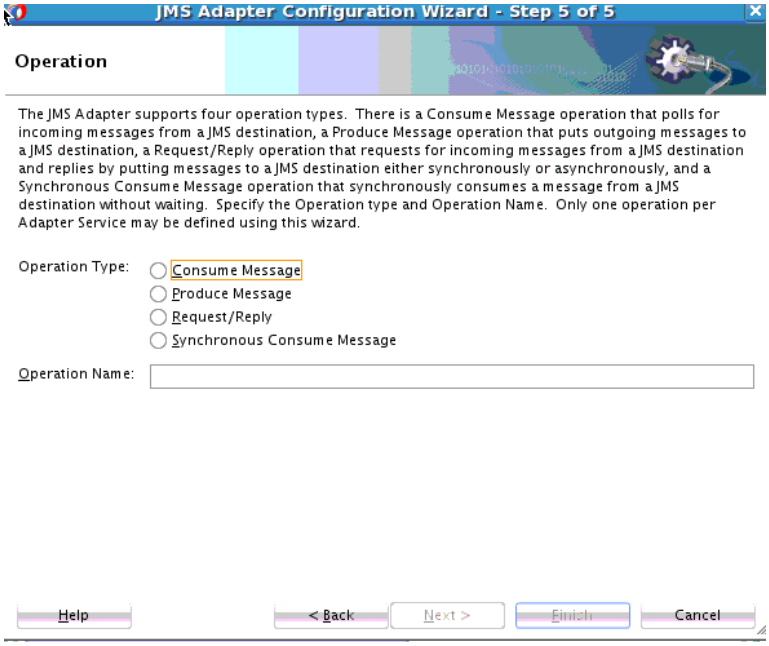
10. Click **Next**.

    The Adapter Configuration Wizard- Operation page is displayed.

11. Select **Consume Message**, **Produce Message**, or **Request/Reply, or Synchronous Consume a Message**. In this example, select **Consume Message**.

    The operation name is filled in automatically, as shown in Figure 8-8.

Figure 8-8 The Adapter Configuration Wizard - Operation Page

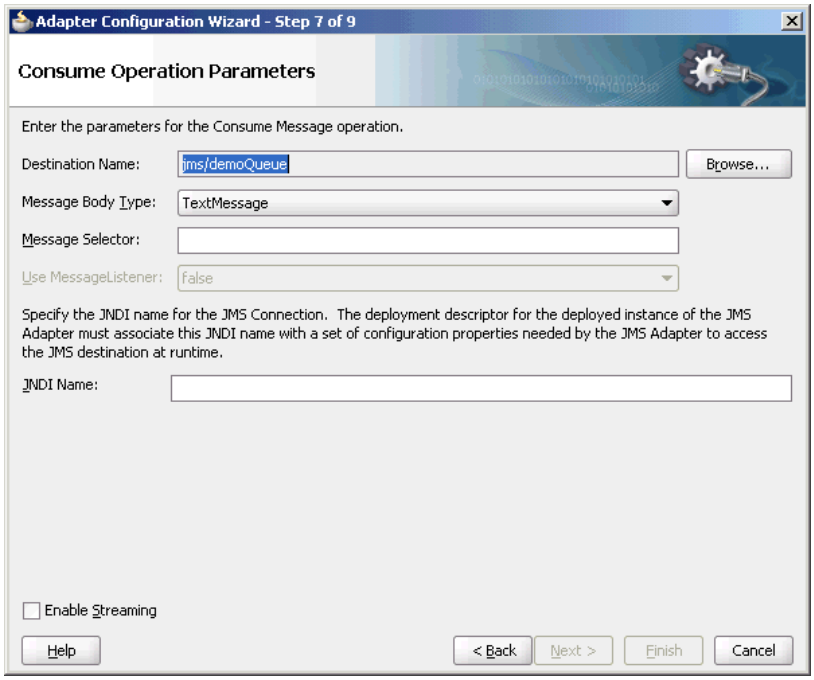


The **Consume Message** option enables the adapter to consume (receive) inbound messages from a JMS destination.

12. Click **Next**.

The Adapter Configuration Wizard - Consume Operation Parameters page is displayed, as shown in Figure 8-9.

Figure 8-9 The Adapter Configuration Wizard - Consume Operation Parameters Page



13. Enter values for the following fields:

- **Destination Name**

  This is the JNDI name of the JMS queue or topic from which to receive the message. This is not an editable field. You must click **Browse** to browse for the queue or topic. The queue or topic to be chosen is based on the type of JMS provider you are using.

  For more information, see the following sections:

  – Configuring the Oracle JMS Adapter with TIBCO JMS

  – Configuring Oracle JMS Adapter with IBM WebSphere MQ JMS

- **Message Body Type**

  The supported values are `TextMessage`, `BytesMessage`, `MapMessage`. The `StreamMessage` message type is not supported in this release.

- **Durable Subscriber ID**

  This field is optional. If you are setting up a durable subscriber, then the durable subscriber ID is required. Generally, a subscriber loses messages if the subscriber becomes disconnected, but a durable subscriber downloads stored messages when it reconnects.

  Note:

  When the JMS provider is Oracle Weblogic JMS or Oracle Advanced queueing messaging service, then the Durable Subscriber option appears only when a topic is selected. However, the Durable Subscriber option always appears when the JMS provider is a third-party.

- **Message Selector**

  This field is also optional. It filters messages based on header and property information. The message selector rule is a Boolean expression. If the expression is `true`, then the message is consumed. If the expression is `false`, then the message is rejected.

  For example, you can enter logic, such as:

  – **JMSPriority > 3**. Based on this, messages with a priority greater than 3 are consumed; all other messages are rejected.

  – **JMSType = 'car' AND color = 'blue' AND weight > 2500**

  – Country in ('UK', 'US', 'France')

- **Use MessageListener**

  This field is always set to `False` by default.

- **JNDI Name**

  The value specified in the JNDI name should exist in the Oracle JMS Adapter weblogic-ra.xml file to ensure that the adapter runs in managed mode.

Note:

This example shows a consume message operation. For a produce message operation, this page is different. See Produce Message Procedure to see how this part of the procedure differs.

After you enter the appropriate parameters, click **Next**.

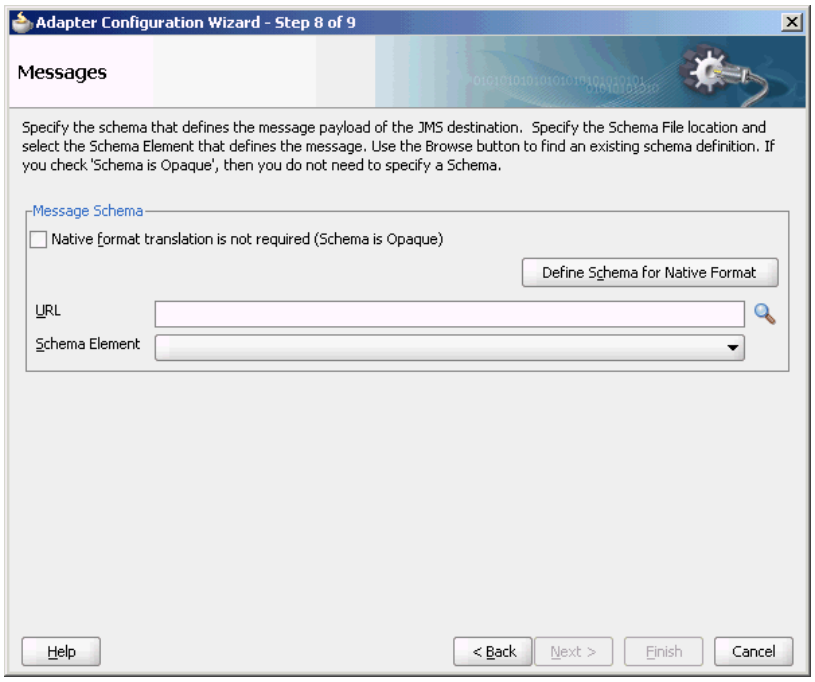
14. The Adapter Configuration Wizard - Messages page is displayed, as shown in Figure 8-10. The settings in this page define the correct schema for the message payload.

    You can perform one of the following:

    - Check **Native format translation is not required (Schema is Opaque)**, which disables the rest of the fields.
    - Click **Define Schema for Native Format** to start the Native Format Builder wizard, which guides you through the process of defining the native format.
    - Enter the path for the schema file URL (or browse for the path).

    The following steps demonstrate the last option: browsing for the schema file URL.

Figure 8-10 The Adapter Configuration Wizard - Messages Page

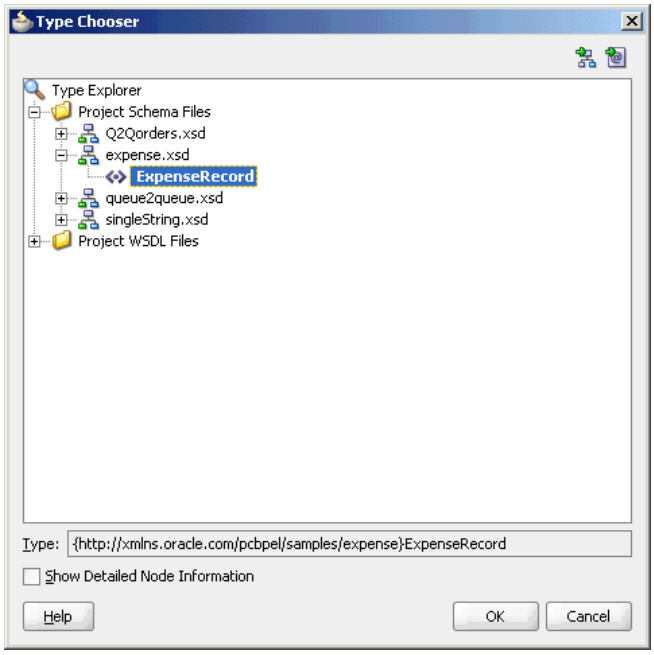


15. Click the **Browse** button.

    The **Type Chooser** dialog is displayed, with the **Type Explorer** navigation tree, as shown in Figure 8-11.

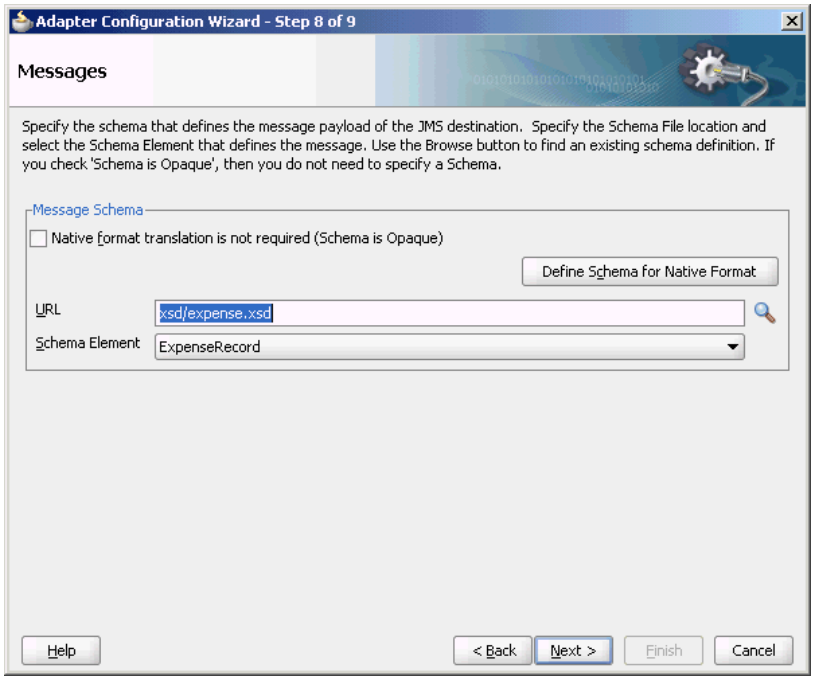
16. Browse the tree and select the appropriate schema type, and then click **OK**.

Figure 8-11 Selecting a Schema from the Type Chooser Dialog



The Messages page is displayed again, this time with the **Schema File URL** field and the **Schema Element** field filled up, as shown in Figure 8-12.

Figure 8-12 Completed Messages Dialog



17. Click **Next**.

    The **Finish** page is displayed. This box shows the path and name of the adapter file that the wizard creates.

18. Click **Finish**.

The `composite.xml` page is displayed.

Generated Files

The following composite file is generated by the Adapter Configuration Wizard:

```
<composite name="AQQueue2Queue"revision="1.0"
           label="2007-09-04_11-58-50_914"mode="active"state="on"
           xmlns="http://xmlns.oracle.com/sca/1.0"           xmlns:xs="http://
www.w3.org/2001/XMLSchema"           xmlns:wsp="http://schemas.xmlsoap.org/ws/
2004/09/policy"           xmlns:orawsp="http://schemas.oracle.com
                 /ws/2006/01/policy">
  <import namespace="http://xmlns.oracle.com/pcbpel/adapter
               /jms/Inbound/"location="Inbound.wsdl"
               importType="wsdl"/>
  <import namespace="http://xmlns.oracle.com/pcbpel/
               adapter/jms/Outbound/"location="Outbound.wsdl"
              importType="wsdl"/>
  <service name="Inbound">
    <interface.wsdl interface="http://xmlns.oracle.com/pcbpel/
          adapter/jms/Inbound/#wsdl.interface
            (Consume_Message_ptt)"/>
    <binding.jca config="Inbound_jms.jca"/>
  </service>
  <component name="BPELProcess1">
    <implementation.bpel src="BPELProcess1.bpel"/>
  </component>
  <reference name="Outbound">
    <interface.wsdl interface=
                "http://xmlns.oracle.com/pcbpel/adapter
                /jms/Outbound/#wsdl.interface (Produce_Message_ptt)"/>
    <binding.jca config="Outbound_jms.jca"/>
  </reference>
  <wire>
    <source.uri>Inbound</source.uri>
    <target.uri>BPELProcess1/Inbound</target.uri>
  </wire>
  <wire>
    <source.uri>BPELProcess1/Outbound</source.uri>
    <target.uri>Outbound</target.uri>
  </wire>
</composite>
```

The following code segment defines the name of the adapter and the locations of various necessary schemas and other definition files.

```
 <import namespace="http://xmlns.oracle.com/pcbpel
          /adapter/jms/Inbound/"location="Inbound.wsdl"
                 importType="wsdl"/>
  <import namespace="http://xmlns.oracle.com
                   /pcbpel/adapter/jms/Outbound/"location=
                   "Outbound.wsdl" importType="wsdl"/>
```

This code segment imports the necessary namespace.

```
<definitions name="Inbound" targetNamespace="http://xmlns.oracle.com/pcbpel/
adapter/jms/Inbound/" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://
xmlns.oracle.com/pcbpel/adapter/jms/Inbound/" xmlns:plt="http://
schemas.xmlsoap.org/ws/2003/05/partner-link/" xmlns:imp1="http://xmlns.oracle.com/
pcbpel/samples/expense">
    <types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
```

```
            <import namespace="http://xmlns.oracle.com/pcbpel/samples/expense"
schemaLocation="xsd/expense.xsd"/>
        </schema>
    </types>
    <message name="ExpenseRecord_msg">
        <part name="ExpenseRecord" element="imp1:ExpenseRecord"/>
    </message>
    <portType name="Consume_Message_ptt">
        <operation name="Consume_Message">
            <input message="tns:ExpenseRecord_msg"/>
        </operation>
    </portType>
```

This code segment defines the message type, name, and the port type for the partner link.

```
<adapter-config name="dequeue" adapter="Jms Adapter   xmlns="http://
platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/wls/Queue" UIConnectionName="wls3"
UIJmsProvider="WLSJMS" adapterRef=""/>
  <endpoint-activation portType="Consume_Message_ptt" operation="Consume_
Message">
    <activation-spec
className="oracle.tip.adapter.jms.inbound.JmsConsumeActivationSpec">
      <property name="DestinationName" value="jms/DemoInQueue"/>
      <property name="UseMessageListener" value="false"/>
      <property name="PayloadType" value="TextMessage"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

weblogic-ra.xml file

The `weblogic-ra.xml` file defines the endpoints for the JMS connection factories. The connection factories include configuration properties for each endpoint. Endpoints are added to accommodate different types of connections, as demonstrated in the following sections. The following example is from the generic `weblogic-ra.xml` file:

```
<connection-instance>
  <jndi-name>eis/wls/Queue</jndi-name>
    <connection-properties>
       <properties>
           <property>
               <name>ConnectionFactoryLocation</name>
               <value>weblogic.jms.XAConnectionFactory</value>
           </property>
           <property>
               <name>FactoryProperties</name>
               <value></value>
           </property>
           <property>
               <name>AcknowledgeMode</name>
               <value>AUTO_ACKNOWLEDGE</value>
           </property>
           <property>
              <name>IsTopic</name>
              <value>false</value>
           </property>
           <property>
               <name>IsTransacted</name>
```

```
            <value>false</value>
        </property>
        <property>
            <name>Username</name>
            <value></value>
        </property>
        <property>
            <name>Password</name>
            <value></value>
        </property>
        </properties>
    </connection-properties>
</connection-instance>
```

Creating a New Connection by Using the Oracle WebLogic Server Administration Console

You can also create a new connection by using the Oracle WebLogic Server Administration Console. The following are the steps for creating a new connection by using the Oracle WebLogic Server Administration Console:

1. Navigate to the Oracle WebLogic Server Administration Console: `http://servername:portnumber/console`.
2. Use the required credentials to open the Home page of the Oracle WebLogic Server Administration Console.

   The Home page of the Oracle WebLogic Server Administration Console is displayed.
3. Select **Deployments** in the Domain Structure pane.

   The Oracle WebLogic Server Administration Console - Summary of Deployments page is displayed.
4. Under **Deployments**, click any JMS adapter that you have deployed. For example, click **JmsAdapter**.

   The Oracle WebLogic Server Administration Console - Settings for JmsAdapter page is displayed.
5. Click the **Configuration** tab, and then click the **Outbound Connection Pools** tab.

   The Outbound Connection Pool Configuration Table is displayed.
6. Click **Next**.

   The Create a New Outbound Connection page is displayed.
7. Select the default outbound connection group, and then click **Next**.
8. Click **Next**.
9. In the **JNDI Name** field, enter the JNDI name that you want to use to obtain the new connection instance. For example, `eis/wls/Queue`.

   You can specify any name for the JNDI field. However, you must ensure that you use the same JNDI name while defining the consume or produce operation parameters in JDeveloper.
10. Click **Finish**.

The Save Deployment Plan Assistant page is displayed.

The configuration changes that you made must be stored in a new deployment plan.

11. In the **Path** field, select or enter the path of a deployment plan file. The path must end with '.xml'.

12. Click **OK**.

    You have created a new connection. After you have done this, you must verify whether the properties you have created are correct.

13. In the Settings for JmsAdapter page, click the **Configuration** tab, and then click the **Properties** tab.

    The connection that you created is listed in this page. Verify whether this value is correct. For example, if you are connecting to a third-party JMS server, then ensure that the Connection Factory Location field has the correct value applicable for a third-party JMS server.

    Note:

    In this example, you created a new connection for Oracle JMS Adapter by using the Oracle WebLogic Server Administration Console. To create connection for other adapters, you must follow the same steps. However, ensure that you select the appropriate adapter for which you want to create a connection in Step 4.

14. Click **Save**.

Adding a Third-Party JMS Provider

You can specify that the adapter use a third-party JMS Provider for non-Web Logic Server JMS and non-AQJMS connection instances, by supplying a value to the FactoryProperties parameter in the `weblogic-ra.xml` file. Specifically, you can provide the `ThirdPartyJMSProvider` value to the FactoryProperties parameter. This property is required only when you deploy an adapter to the WebLogic Server.

When this value is set to true, the JMS Adapter does not use `DestinationAvailabilityListener` for creating consumers for processing of JMS messages. The default is false. You must ensure you employ code similar to the following snippet:

```
<property>
    <name>FactoryProperties</name>
    <value>ThirdPartyJMSProvider=true</value>
</property>
```

Note:

All pre-populated connection instances on a WebLogic Server reflect the change and consequently, no further tuning is required for those instances. Only when a new non WLS JMS or AQJMS provider access is required do you must add new a connection instance and therefore the `ThirdPartyJMSProvider` property.

Produce Message Procedure

A produce message operation has certain differences in the definition procedure, particularly in Step 13 of Using the Adapter Configuration Wizard to Configure . Instead of specifying consume operation parameters, you specify the following produce operation parameters. This enables the adapter to produce (send) outbound messages for a JMS destination. The Produce Operation Parameters page is shown in Figure 8-13.

- **Destination Name:**

  The JNDI name of the JMS queue or topic to which the message must be delivered. The name to enter is based on the type of JMS provider you use.

  For more information about destination name, see the following:

  – Configuring the Oracle JMS Adapter with TIBCO JMS

  – Configuring Oracle JMS Adapter with IBM WebSphere MQ JMS

- **Message Body Type:**

  The supported values are `TextMessage`, `BytesMessage`, and `MapMessage`. `StreamMessage` is not supported in this release.

- **Delivery Mode:**

  The values are `Persistent` or `Nonpersistent`. A persistent delivery mode specifies a persistent JMS publisher; that is, a publisher that stores messages for later use by a durable subscriber. A durable subscriber is a consume message with a durable subscriber ID in the corresponding field in Step 15 of Using the Adapter Configuration Wizard to Configure . A nondurable subscriber loses any messages that are produced when the adapter is not active. A durable subscriber downloads messages that have been stored in the persistent publisher, and therefore does not have to remain active at all time to receive all the messages.

- **Priority:**

  Select a priority value, with `9` representing the highest priority and `0` representing the lowest priority. The default is `4`.

- **TimeToLive:**

  The amount of time before the message expires and is no longer available to be consumed.

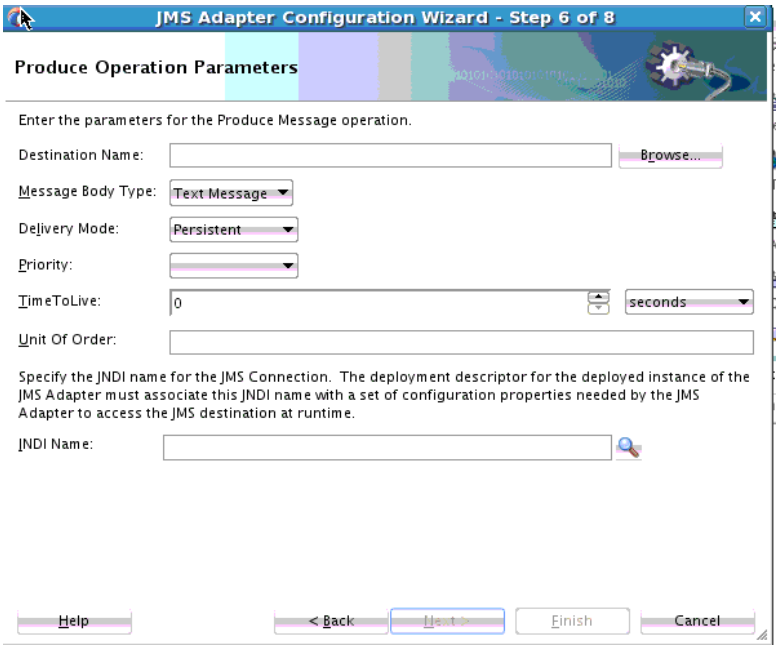
- **Unit of order**

  This parameter only applies when the selected JMS Provider is a WebLogic Server JMS provider. It enables a message producer or group of message producers acting as one, to group messages into a single unit that is processed sequentially in the order the messages were created. The message processing of a single message is complete when a message is acknowledged, committed, recovered, or rolled back. Until message processing for a message is complete, the remaining unprocessed messages for that Unit of Order are blocked. This `Unit of order` property enables you to set the unit-of-order value for the MessageProducer when producing a message. Refer to the Use Case “Using the WLS JMS Unit-of-Order with the JMS Adapter", and “Using Message Unit-of-Order" in *Developing JMS Applications for Oracle WebLogic Server*.

- JNDI Name

  This field enables you to specify the JNDI name for the JMS connection. The deployment descriptor for the deployed instance of the JMS Adapter must associate this JNDI name with a set of configuration properties required by the JMS Adapter to access the JNDI destination at runtime.

Figure 8-13 Produce Operation Parameters Page



Configuring the Oracle JMS Adapter with TIBCO JMS

This section describes how to configure Oracle JMS Adapter with Tibco JMS for direct connection and non direct connection.

Note:

Read the entire section before making configuration changes, as not all information is available in each subsection. Information related to TIBCO Enterprise Message Service is based on TIBCO Enterprise Message Service User's Guide, Software Release 6.3, February 2012. Refer to TIBCO's documents for latest updates.

Also, note that in the following discussion `EMS_HOME` stands for Tibco Enterprise Message Service installation directory.

Using Preconfigured Tibco Connection Factory for non-SSL Connections

Use the TIBCO Enterprise Message Service Administration Tool to create a connection factory and queue/topic.

1. For example,

```
create factory QueueConnectionFactory queue url=tcp://localhost:7222
create queue TibcoQueue
```

2. Open the Weblogic Administration Console to modify an existing connection pool for connection to a Tibco Server. Navigate to **Deployment->JmsAdapter->Configuration-> oracle.tip.adapter.jms.IJmsConnectionFactory->Outbound Connection Pools->eis/tibjms/Queue**

3. Set the `ConnectionFactoryLocation` to `QueueConnectionFactory`. This is the name of the connection factory created earlier using TIBCO Enterprise Message Service Administration Tool.

4. Set the `FactoryProperties` to the following (separated into multiple lines for easier reading):

```
java.naming.factory.initial=com.tibco.tibjms.naming.
TibjmsInitialContextFactory;
java.naming.provider.url=tibjmsnaming://localhost:7222;
java.naming.security.principal=<user name on Tibco server>;
java.naming.security.credentials=<password>;
  ThirdPartyJMSProvider=true
```

5. You can also create your own connection pool by modifying the `weblogic-ra.xml` file in `AS11gR1SOA/soa/connectors/JmsAdapter.rar` as shown in the example below. (You can use this as an example for creating a Topic connection pool, also):

   Example - Creating Your Own Connection Pool

```
<connection-instance>
     <jndi-name>eis/tibjms/QueueConnectionPool</jndi-name>
         <connection-properties>
             <properties>
                 <property>
                     <name>ConnectionFactoryLocation</name>
                     <value>QueueConnectionFactory</value>
                 </property>
                 <property>
                     <name>FactoryProperties</name>
                     <value>java.naming.factory.initial=com.
                          tibco.tibjms.naming.
                          TibjmsInitialContextFactory;
                            java.naming.provider.
                                 url=tibjmsnaming://localhost:7222;
                                    java.naming
                             .security.principal=<user name on
                                        Tibco server>;
                              java.naming.security.credentials=
                                        <password>;
                                 ThirdPartyJMSProvider=true
                         </value>
                     </property>
                     <property>
                     <name>AcknowledgeMode</name>
                     <value>AUTO_ACKNOWLEDGE</value>
                 </property>
                 <property>
                     <name>IsTopic</name>
                     <value>false</value>
                 </property>
                 <property>
                     <name>IsTransacted</name>
                     <value>true</value>
                 </property>
```

```
                <property>
                    <name>Username</name>
                    <value></value>
                </property>
                <property>
                    <name>Password</name>
                    <value></value>
                </property>
            </properties>
        </connection-properties>
 </connection-instance>
```

6. Update the JMSAdapter connection pool deployment after making these changes. Copy `EMS_HOME/lib/tibjms.jar` to the `<SOA Installation>/user_projects/domains/<DOMAIN_NAME>/lib` directory. Restart the Weblogic Server.

7. In the project you deploy, the JMS jca file for the JMS adapter might look like the following. Note that the value of DestinationName is `TibcoQueue` which is the queue created using TIBCO Enterprise Message Service Administration Tool.

```
<connection-factory UIJmsProvider="THIRDPARTY"
                         location="eis/tibjms/Queue"
                      UIConnectionName="SOADEV"/>
  <endpoint-interaction portType="Produce_Message_ptt"
                       operation="Produce_Message">
     <interaction-spec
       className="oracle.tip.adapter.jms.
outbound.JmsProduceInteractionSpec">
       <property name="TimeToLive" value="0"/>
       <property name="PayloadType" value="TextMessage"/>
       <property name="DeliveryMode" value="Persistent"/>
       <property name="DestinationName" value="TibcoQueue"/>
     </interaction-spec>
   </endpoint-interaction>
```

8. After messages are published, you can use the TIBCO Enterprise Message Service Administration Tool to verify the number of messages in a Tibco queue, for example:

```
show queue TibcoQueue
  Queue:                TibcoQueue
  Type:                 static
  Properties:           *prefetch=5
  JNDI Names:           <none>
  Bridges:              <none>
  Receivers:            0
  Pending Msgs:         2, (2 persistent)
  Delivered Msgs:       0
  Pending Msgs Size:    1.2 Kb, (1.2 Kb persistent)
```

Using Dynamically Created Tibco Connection Factory for non-SSL Connections

A preconfigured connection factory is often the preferred mechanism; however, Tibco does provide support for dynamically created connection factories. To use these,

1. Use the TIBCO Enterprise Message Service Administration Tool to create a queue or topic.

```
create queue TibcoQueue
```

2. Set the `ConnectionFactoryLocation` to `com.tibco.tibjms.TibjmsQueueConnectionFactory`.
3. Set the `FactoryProperties` to `ServerUrl=tcp://<HOST>:<PORT>;UserName=<USERNAME>;UserPassword=<PASSWORD>`
4. If the Tibco server does not require a user name and password to connect remove the user name and password fields.
5. Save and update the JMS Adapter connection pool deployment.
6. Copy `EMS_HOME/lib/tibjms.jar and EMS_HOME/lib/jms-2.0.jar to <SOA Installation>/user_projects/domains/<DOMAIN_NAME>/lib` directory.
7. Restart the managed Weblogic Server.
8. In the project to be deployed, the JMS jca file for the JMS Adapter might look like the following example.

   Note that value of DestinationName is `TibcoQueue,` which is the queue created using TIBCO Enterprise Message Service Administration Tool.

   Example - JMS jca File for JMS Adapter with Dynamically-Created Connection Factory

```
<connection-factory UIJmsProvider="THIRDPARTY"
location="eis/tibjmsDirect/Queue" UIConnectionName="SOADEV"/>
 <endpoint-interaction portType="Produce_Message_ptt"
operation="Produce_Message">
    <interaction-spec
className="oracle.tip.adapter.jms.outbound.JmsProduceInteractionSpec">
      <property name="TimeToLive" value="0"/>
      <property name="PayloadType" value="TextMessage"/>
      <property name="DeliveryMode" value="Persistent"/>
      <property name="DestinationName" value="TibcoQueue"/>
    </interaction-spec>
  </endpoint-interaction>
```

Using a Preconfigured Tibco Connection Factory for SSL Connections

You can use a pre-configured Tibco Connection Factory for SSL Connections. To so, you must configure the Tibco server to support SSL.

1. Create a `certs` directory under `EMS_HOME/bin`.
2. Copy server SSL certificates to the `EMS_HOME/bin/certs` directory. Tibco supplies, for the purposes of testing, a self-signed server certificate and private keys which you can find in `EMS_HOME\bin\samples\certs`.
3. In `EMS_HOME/bin/tibemsd.conf`, add the following lines:

```
listen = ssl://localhost:7243
ssl_server_identity = certs/server.cert.pem
ssl_server_key = certs/server.key.pem
ssl_password = password
listen = tcp://localhost:7222
```

These lines explicitly set the TCP and SSL listen ports and specify the three required server-side SSL parameters: identity, private key, and password.

4. Save the file and stop the TIBCO Enterprise Message Service server.
5. Start the TIBCO EMS server with the `-ssl_debug_trace` option: `tibemsd -ssl_debug_trace`.
6. Verify server is listening on the SSL port in server console

```
2014-04-29 16:14:20.997 Secure Socket Layer is enabled, using OpenSSL 0.9.8y5
Feb 2013
2014-04-29 16:14:20.998 Accepting connections on ssl://localhost/[::]:7243.
2014-04-29 16:14:20.998 Accepting connections on
ssl://localhost/0.0.0.0:7243.
2014-04-29 16:14:20.998 Accepting connections on tcp://localhost/[::]:7222.
2014-04-29 16:14:20.998 Accepting connections on
tcp://localhost/0.0.0.0:7222.
```

7. Use the TIBCO Enterprise Message Service Administration Tool to create an SSL ConnectionFactory, disable host verification (do not disable host verification if you want to verify the client host), and create a queue:

```
create factory SSLQueueConnectionFactory queue url=ssl://localhost:7243
setprop factory SSLQueueConnectionFactory ssl_verify_host=disabled
create queue TibcoQueue
```

8. Open the Weblogic Administration Console to modify an existing connection pool for connection to Tibco Server. Navigate to **Deployment->JmsAdapter->Configuration-> oracle.tip.adapter.jms.IJmsConnectionFactory->Outbound Connection Pools->eis/tibjms/Queue**.
9. Set the `ConnectionFactoryLocation` to `SSLQueueConnectionFactory`. This is the name of the connection factory you created earlier using the TIBCO Enterprise Message Service Administration Tool.
10. Set the `FactoryProperties` to (separated into multiple lines for easy reading):

```
java.naming.factory.initial=com.tibco.tibjms.naming.
TibjmsInitialContextFactory;
 java.naming.provider.url=tibjmsnaming://localhost:7243;
 java.naming.security.principal=<user name on Tibco server>;
 java.naming.security.credentials=<password>;ThirdPartyJMSProvider=true;
 com.tibco.tibjms.naming.security_protocol=ssl;
 com.tibco.tibjms.naming.ssl_enable_verify_host=false;
 com.tibco.tibjms.naming.ssl_vendor=j2se-default
```

11. This enables the client to connect without a certificate. Save and update the deployed connection pool.
12. Copy the following jar files located in `EMS_HOME\lib` to `<SOA Installation>/user_projects/domains/<DOMAIN_NAME>/lib` directory:

```
tibcrypt.jar
slf4j-api-1.4.2.jar
slf4j-simple-1.4.2.jar
tibjms.jar
```

13. Restart Weblogic Server.

14. In the project to be deployed, the JMS jca file for the JMS adapter might look like the following example.

    Notice that value of DestinationName is `TibcoQueue` which is the queue created using TIBCO Enterprise Message Service Administration Tool.

    Example - JMS jca File for Prefconfigured TIBCO Connection Factory Scenario

```
<connection-factory UIJmsProvider="THIRDPARTY" location="eis/tibjms/Queue"
          UIConnectionName="SOADEV"/>
  <endpoint-interaction portType="Produce_Message_ptt"
       operation="Produce_Message">
     <interaction-spec
         className="oracle.tip.adapter.jms.outbound.JmsProduceInteractionSpec">
       <property name="TimeToLive" value="0"/>
       <property name="PayloadType" value="TextMessage"/>
       <property name="DeliveryMode" value="Persistent"/>
       <property name="DestinationName" value="TibcoQueue"/>
     </interaction-spec>
   </endpoint-interaction>
```

15. To enable client certificate verification, you must modify `EMS_HOME\bin\tibemsd.conf` to add the CA certificate which was used to sign the client certificate, for example: `ssl_server_trusted = certs/test_client_ca.pem`, where `test_client_ca.pem` is a Tibco-supplied test CA certificate used to sign client `certificate test_client_identity.p12`. Both files can be found in `EMS_HOME\bin\samples\certs`. The modified EMS_HOME\bin\tibemsd.conf now looks like the following:

```
listen = ssl://localhost:7243
 ssl_server_identity = certs/server.cert.pem
 ssl_server_key = certs/server.key.pem
 ssl_password = password
 ssl_server_ciphers = all
 ssl_server_trusted = certs/test_client_ca.pem
 listen = tcp://localhost:7222
```

16. Start the TIBCO EMS server with the -ssl_debug_trace option: `tibemsd -ssl_debug_trace`

17. Modify the `FactoryProperties` (separated into multiple lines for easy reading):

```
java.naming.factory.initial=com.tibco.tibjms.naming.
TibjmsInitialContextFactory;
 java.naming.provider.url=tibjmsnaming://localhost:7243;
 java.naming.security.principal=<user name on Tibco server>;
 java.naming.security.credentials=<password>;
 ThirdPartyJMSProvider=true;
 com.tibco.tibjms.naming.security_protocol=ssl;
 com.tibco.tibjms.naming.ssl_enable_verify_host=false;
 com.tibco.tibjms.naming.ssl_vendor=j2se-default;
 com.tibco.tibjms.naming.ssl_identity=<EMS_HOME>/bin/certs/
test_client_identity.p12;
 com.tibco.tibjms.naming.ssl_trusted=<EMS_HOME>/bin/certs/test_client_ca.pem;
 com.tibco.tibjms.naming.ssl_ciphers=all
```

18. With these changes, the client can connect to the server with a certificate as shown in the server console:

```
2014-04-30 13:39:47.485 Peer certificate:
2014-04-30 13:39:47.485 Certificate=[/C=US/ST=California/L=us-english/O=Test
Company/OU=test_client
Unit/CN=test_client/emailAddress=test_clienttestcompany.com]
Issuer=[/C=US/ST=California/L=us-english/O=Test Company/OU=test_client_root
Unit/CN=test_client_root/emailAddress=test_client_roottestcompany.com]
2014-04-30 13:39:47.485 Peer certificate chain:
2014-04-30 13:39:47.485 Certificate=[/C=US/ST=California/L=us-english/O=Test
Company/OU=test_client_root
Unit/CN=test_client_root/emailAddress=test_client_roottestcompany.com]
Issuer=[/C=US/ST=California/L=us-english/O=Test Company/OU=test_client_root
Unit/CN=test_client_root/emailAddress=test_client_roottestcompany.com]
2014-04-30 13:39:47.485 SSL accepted cipher=RC4-SHA
```

19. To limit the server to accept SSL connections only, add the following to `EMS_HOME\bin\tibemsd.conf`:

```
ssl_require_client_cert = enable
```

Using Dynamically Created Tibco Connection Factory for SSL Connections

The Oracle JMS Adapter currently does not support this feature.

Configuring Oracle JMS Adapter with IBM WebSphere MQ JMS

This section describes how to configure Oracle JMS Adapter with IBM WebSphere MQ JMS for non-XA and XA data sources.

Non-XA Data Sources

Perform the following steps:

1. Copy the following files to the `<SOAInstall_DIR>/user_projects/domains/<DOMAIN_NAME>/lib` folder:
   - `/<YOUR-MQSERIES-INSTALL-LOCATION>/java/lib/com.ibm.mq.jar`
   - `/<YOUR-MQSERIES-INSTALL-LOCATION>/java/lib/com.ibm.mqjms.jar`
   - `/<YOUR-MQSERIES-INSTALL-LOCATION>/java/lib/dhbcore.jar`
2. Configure the connector factory by modifying the `weblogic-ra.xml` file in `AS11gR1SOA/soa/connectors/JmsAdapter.rar`, as shown in the following example:

```
<connection-instance>
<jndi-name>eis/webspheremq/Queue</jndi-name>
   <connection-properties>
      <properties>
         <property>
           <name>ConnectionFactoryLocation</name>
           <value>com.ibm.mq.jms.
                MQQueueConnectionFactory</value>
         </property>
         <property>
            <name>FactoryProperties</name>
            <value>QueueManager=<QUEUEMANAGER>;
                   TransportType=1;HostName=
               <YOUR-HOST>;Port=<YOUR-PORT>;Channel=<CHANNEL>;
```

```
                    ThirdPartyJMSProvider=true
</value>
          </property>
          <property>
              <name>AcknowledgeMode</name>
              <value>AUTO_ACKNOWLEDGE</value>
          </property>
          <property>
             <name>IsTopic</name>
             <value>false</value>
          </property>
          <property>
             <name>IsTransacted</name>
             <value>true</value>
         </property>
         <property>
             <name>Username</name>
             <value><USERNAME></value>
         </property>
         <property>
            <name>Password</name>
            <value><PASSWORD></value>
        </property>
        </properties>
    </connection-properties>
</connection-instance>
```

Note that the default <USERNAME> and <PASSWORD> are `MUSR_MQADMIN` and `password`, respectively.

Alternatively, to configure a new connection factory by using the Oracle WebLogic Server Administration Console, use the steps mentioned in Adding an Adapter Connection Factory.

Using a Multi-Instance Queue Manager

You can configure the JMS Adapter to connect with IBM WebSphere MQ JMS and to use a Multi-Instance Queue Manager.

To do so, use a custom property called `connectionNameList`. You can use this property to specify the names and ports of the different instances.

```
QueueManager=QM1;TransportType=1;
ConnectionNameList=slj01aml.us.domain.com(141
4),acb113114.us.domain.com(1414);
Channel=SYSTEM.DEF.SVRCONN;ThirdPartyJMSProv
ider=true;ClientReconnectOptions=0
```

XA Data Sources

Perform the following steps:

1. Copy the following files to `the <SOAInstall_DIR>/user_projects/domains/<DOMAIN_NAME>/lib` folder:
   - `/<YOUR-MQSERIES-INSTALL-LOCATION>/java/lib/com.ibm.mq.jar`
   - `/<YOUR-MQSERIES-INSTALL-LOCATION>/java/lib/com.ibm.mqjms.jar`
   - `/<YOUR-MQSERIES-INSTALL-LOCATION>/java/lib/dhbcore.jar`

- `com.ibm.mqetclient.jar`

  This IBM-extended transactional client (com.ibm.mqetclient.jar) is required for MQ Series versions prior to 7.5In v7.5 of IBM MQ Series, this `com.ibm.mqetclient.jar` JAR file is not required to perform XA transactions. XA transactions work acceptably without this JAR file.

2. Configure the connector factory by modifying the `weblogic-ra.xml` file in `AS11gR1SOA/soa/connectors/JmsAdapter.rar`, as shown in the following example:

```
<connection-instance>
<jndi-name>eis/webspheremq/Queue</jndi-name>
   <connection-properties>
        <properties>
           <property>
              <name>ConnectionFactoryLocation</name>
              <value>com.ibm.mq.jms.
MQXAQueueConnectionFactory</value>
           </property>
           <property>
              <name>FactoryProperties</name>
              <value>QueueManager=<QUEUEMANAGER>;
TransportType=1;
                 HostName=<YOUR-HOST>;
                       Port=<YOUR-PORT>;                  Channel=<CHANNEL>;
|
                       ThirdPartyJMSProvider
                              =true</value>
           </property
           <property>
             <name>AcknowledgeMode</name>
             <value>AUTO_ACKNOWLEDGE</value>
           </property>
           <property>
             <name>IsTopic</name>
             <value>false</value>
           </property
           <property>
              <name>IsTransacted</name
              <value>false</value>
           </property
           <property>
              <name>Username</name>
              <value><USERNAME></value>
           </property>
           <property>
              <name>Password</name>
              <value><PASSWORD></value>
            </property>
         </properties>
    </connection-properties>
</connection-instance>
```

Note that the default <USERNAME> and <PASSWORD> are `MUSR_MQADMIN` and `password`, respectively.

Alternatively, to configure a new connection factory by using the Oracle WebLogic Server Administration Console, use the steps mentioned in Adding an Adapter Connection Factory.

Configuring Oracle JMS Adapter with Active MQ JMS

This section describes how to configure Oracle JMS Adapter with Active MQ JMS (non-XA only).

Perform the following steps:Copy the following files to the `<SOAInstall_DIR>/user_projects/domains/<DOMAIN_NAME>/lib` folder:

- `/<YOUR-ACTIVEMQ-INSTALL-LOCATION>//activemq-all-5.8.0.jar`

Configure the connector factory by modifying the `weblogic-ra.xml` file in `AS11gR1SOA/soa/connectors/JmsAdapter.rar` as shown in the following example:

```
<connection-instance>
      <jndi-name>eis/activemq/Queue</jndi-name>
      <connection-properties>
      <properties>
            <property>
                  <name>ConnectionFactoryLocation</name>
                  <value>org.apache.activemq.
                        ActiveMQConnectionFactory</value>
            </property>
            <property>
                  <name>FactoryProperties</name>
<value>BrokerURL=tcp://<YOUR-HOST>:
                  <YOUR-PORT>;ThirdPartyJMSProvider=true</value>
            </property>
            <property>
                  <name>AcknowledgeMode</name>
                  <value>AUTO_ACKNOWLEDGE</value>
            </property>
            <property>
                  <name>IsTopic</name>
                  <value>false</value>
            </property>
            <property>
                  <name>IsTransacted</name>
                   <value>true</value>
            </property>
            <property>
                  <name>Username</name>
                  <value></value>
           </property>
           <property>
                 <name>Password</name>
                 <value></value>
            </property>
      </properties>
      </connection-properties>
</connection-instance>
```

Alternatively, to configure a new connection factory by using the Oracle WebLogic Server Administration Console, use the steps mentioned in Adding an Adapter Connection Factory.

For ActiveMQ Series v5.8, there is XA Support and the Connection Factory to be configured for XA is `org.apache.activemq.ActiveMQXAConnectionFactory`. This has been certified with the current release of the JMS Adapter.

WebLogic Server JMS Text Message

This WebLogic Server JMS text message use case for Oracle BPEL PM demonstrates how the Oracle JMS Adapter dequeues from and enqueues to the WebLogicServer JMS Queue.

In the case of a WLS JMS text message scenario for a Mediator business process, you must have the following files from the `artifacts.zip` file contained in the `adapters-jms-101-wlsjms-textmessageusingqueues` sample:

- `artifacts/schemas/expense.xsd`

You can obtain the `adapters-jms-101-wlsjms-textmessageusingqueues` sample by accessing the Oracle SOA Sample Code site.

This section includes the following topics:

- Meeting Prerequisites
- Creating an Application Server Connection
- Creating an Application and an SOA Project
- Creating an Inbound Adapter Service
- Creating an Outbound Adapter Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using the Fusion Middleware Control Console

Meeting Prerequisites

You must perform the following prerequisite for the WebLogic Server JMS text message use case for Oracle BPEL PM.

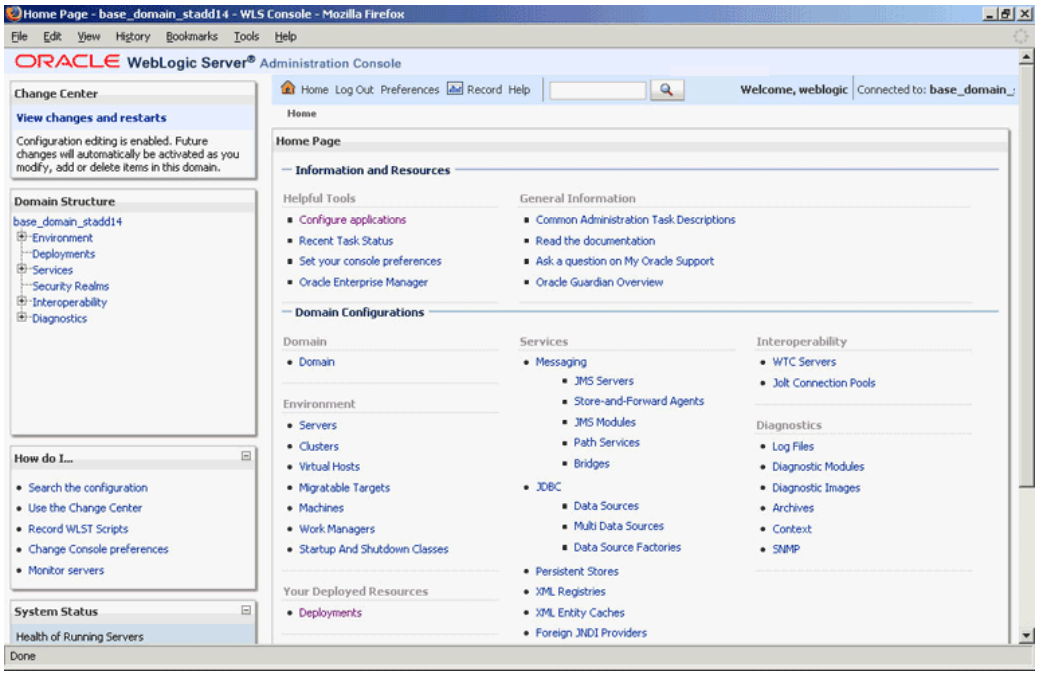
Creating Queues in the Oracle WebLogic Server Administration Console:

Perform the following steps to create queues required for this use case:

1. Navigate to the Oracle WebLogic Server Administration Console: `http://servername:portnumber/console`
2. Use the required credentials to open the Home page of the Oracle WebLogic Server Administration Console.

   The Home page of the Oracle WebLogic Server Administration Console is displayed, as shown in Figure 8-14.

Figure 8-14 The Oracle WebLogic Server Administration Console Home Page



3. Navigate to **Services**, **Messaging**, **JMS Modules** in the **Domain Structure** pane.

   The Oracle WebLogic Server Administration Console - JMS Modules page is displayed.

4. Click one of the existing modules. In this example, click **SOAJMSModule**.

   The **Oracle WebLogic Server Administration Console - Settings for SOAJMSModule** page is displayed.

5. Under the **Summary of Resources** section, click **New**.

   The Oracle WebLogic Server Administration Console - **Create a New JMS System Module Resource** page is displayed.

6. Select **Queue**, and then click **Next**.

7. Enter the following queue details:

   - Name
   - JNDI Name
   - Template

8. Click **Next**.

9. Select the subdeployment you want to use from the Subdeployments list.

10. Click **Finish**.

    You have created the queue, `ReceiveQueue`.

11. Repeat steps 1 through 10, and create a queue named `SendQueue`.

Creating the Q2Qorders.xsd file

Create the Q2Qorders.xsd file by using the following code:

```
<?xml version="1.0" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            targetNamespace="http://xmlns.oracle.com/
                 pcbpel/nxsd/extensions/FileInbound"
            xmlns:tns="http://xmlns.oracle.com/
            pcbpel/nxsd/extensions/FileInbound"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
              nxsd:encoding="US-ASCII" nxsd:stream="chars"
                        nxsd:version="NXSD">
  <xsd:element name="Items">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="item" minOccurs="1"
                              maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Name" type="xsd:string"
                nxsd:style="terminated"
                 nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Type" type="xsd:string"
                 nxsd:style="terminated"
                  nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Quantity"
                  type="xsd:string" nxsd:style="terminated"
                           nxsd:terminatedBy=","
                             nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Rate"
                type="xsd:string"
                  nxsd:style="terminated"
                  nxsd:terminatedBy="${eol}" nxsd:quotedBy="&quot;">
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
<!--NXSDWIZ:C:\errors\inputFiles\orders.txt:-->
```

Creating an Application Server Connection

You must establish connectivity between the design-time environment and the server you want to deploy to. Perform the steps mentioned in Creating an Application Server Connection for Oracle JCA Adapters to create an application server connection.

Creating an Application and an SOA Project

You must create an JDeveloper application to contain the SOA composite. Use the following steps to create an application and an SOA project:

1. Open JDeveloper.
2. In the **Application Navigator**, click **New Application**. The New Gallery dialog is displayed.
3. Select **SOA Application** and click **OK**. Enter a name for the application in the **Application Name** field. For example, `queue2queue`.
4. Click **Next**.

   The **Name your project** page is displayed.
5. In the **Project Name** field, enter a descriptive name. For example, `queue2queue`.
6. Click **Next**. The **Configure SOA settings** page is displayed.
7. Select **Composite With BPEL Process** from the Composite Template list, and then click **Finish**.

   You have created a new application, and an SOA project.

   The **Create BPEL Process** page is displayed.
8. Enter a name for the BPEL process in the **Name** field. For example, `queue2queue`.
9. Select **Define Interface Later** in the Template list, and then click **OK**.

   You have created a BPEL process.

   The `queue2queue` application, `queue2queue` project, and the SOA composite appear in the design area.
10. Copy the `Q2Qorders.xsd` file to the XSD folder in your project.

Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message to a queue:

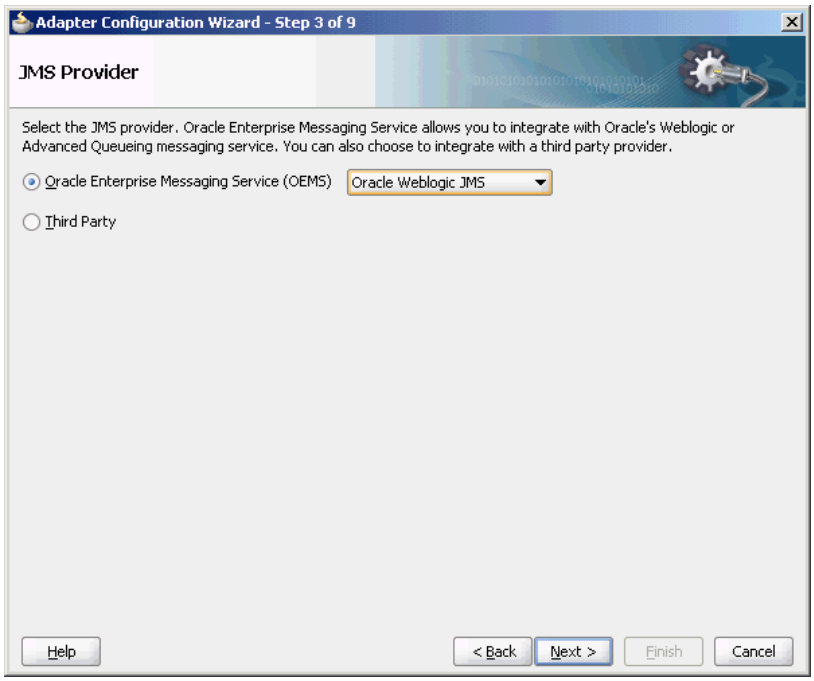
1. Drag and drop **JMS Adapter** from the Service Adapters list to the Exposed Services swim lane in the composite.xml page.

   The **Adapter Configuration Wizard Welcome** page is displayed.
2. Click **Next**.

   The **Service Name** page is displayed.
3. Enter `Inbound` in the **Service Name** field, and click **OK**.

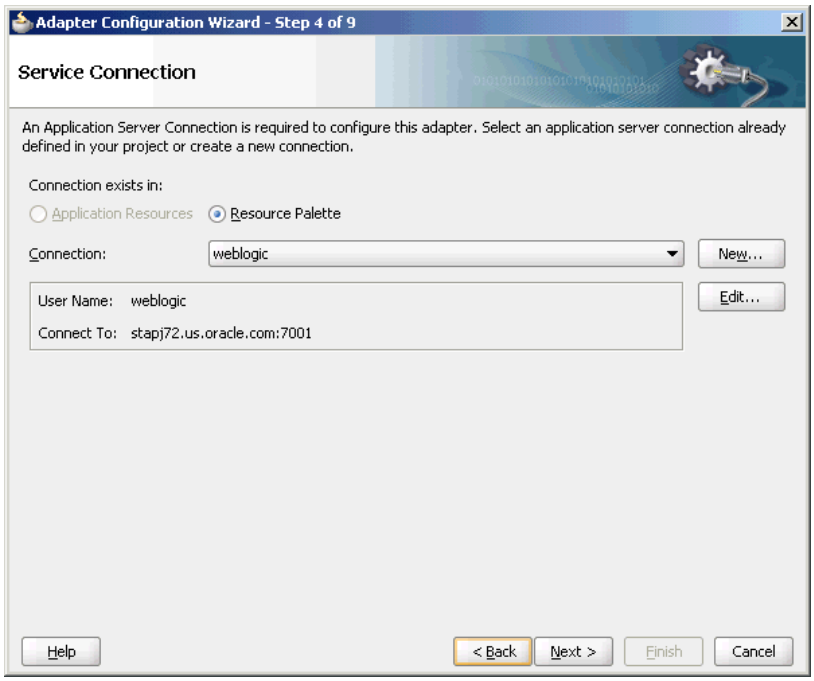
   The **JMS Provider** page is displayed.
4. Select **Oracle Weblogic JMS** in the Oracle Enterprise Messaging Service (OEMS) box, as shown in Figure 8-15, and click **Next**. The Service Connection page is displayed.

Figure 8-15 The Adapter Configuration Wizard JMS Provider Page



5. Select the connection created in Creating an Application Server Connection, as shown in Figure 8-16.

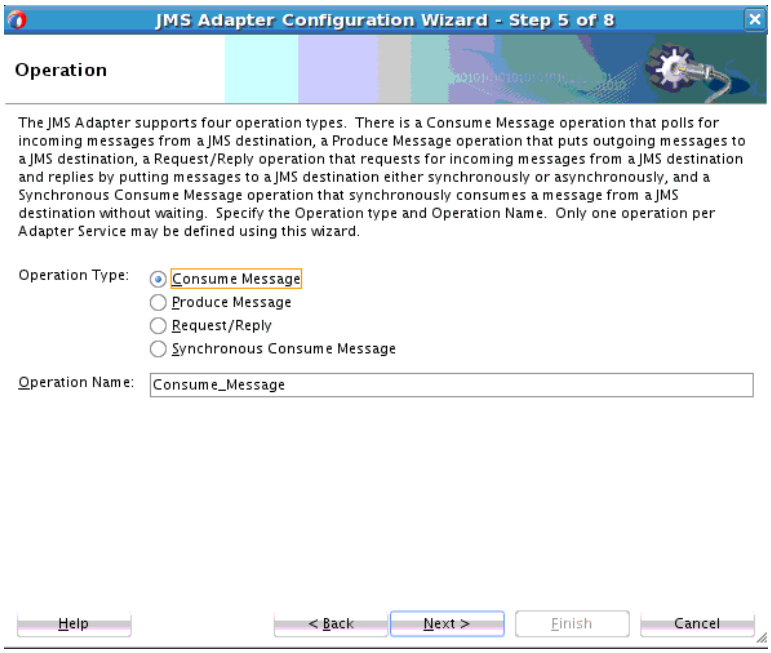
Figure 8-16 The Adapter Configuration Wizard Service Connection Page



6. Click **Next**. The **Adapter Interface** page is displayed.
7. Select **Define from operation and schema (specified later)**, and click **Next**. The **Operation** page is displayed.
8. Select **Consume Message**, as shown in Figure 8-17, and click **Next**.

The **Consume Operation Parameters** page is displayed.

Figure 8-17 The Adapter Configuration Wizard Operation Page



9. Click **Browse** and select **ReceiveQueue** in the Destination field.

   The **Consume Operation Parameters** page is displayed.

10. Enter the parameters for the consume operation, and then click **Next**.

    The **Messages** page is displayed.

    Note:

    The value specified in the JNDI name should exist in the Oracle JMS Adapter `weblogic-ra.xml` file to ensure that the Sdapter runs in managed mode.

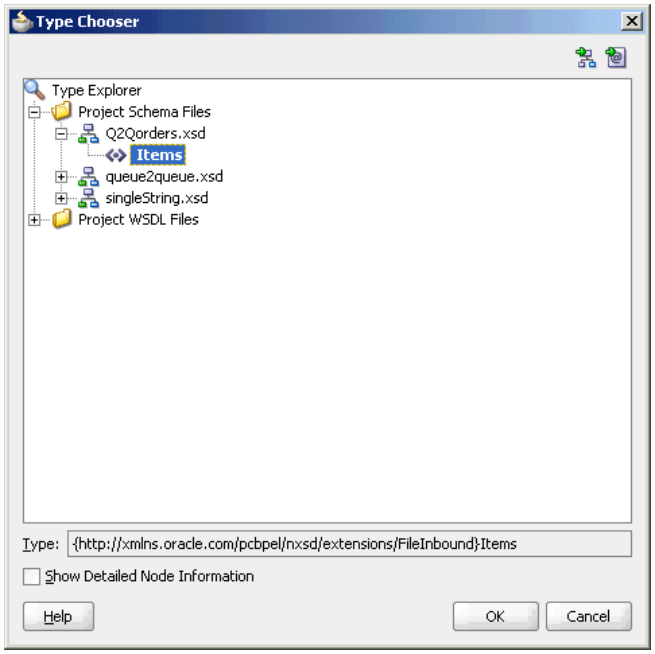
    ---

11. Click **Browse** at the end of the URL field.

    The Type Chooser dialog is displayed.

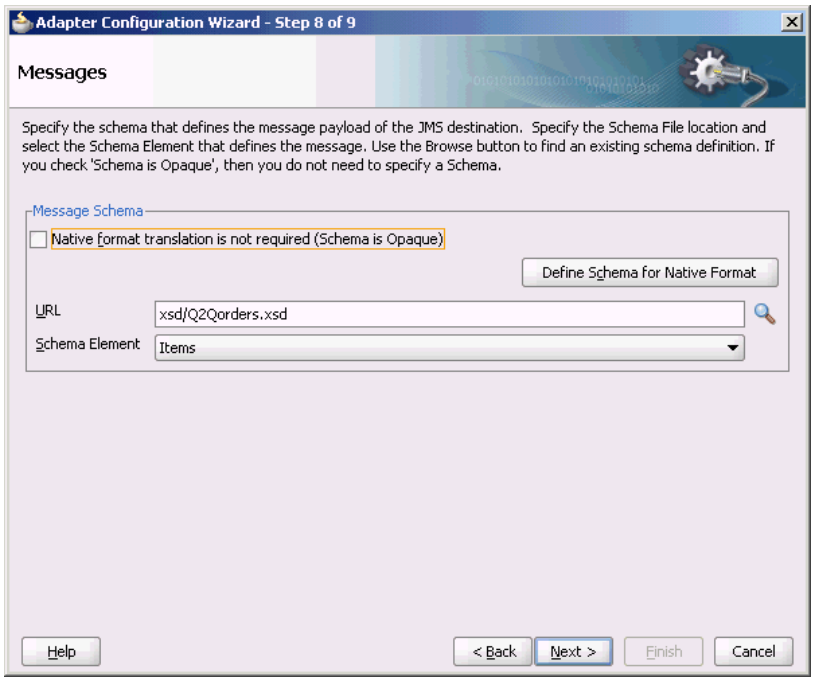
12. Select **Project Schema Files**, **Q2Qorders.xsd**, and **Items**, as shown in Figure 8-18.

Figure 8-18 The Type Chooser Dialog



13. Click **Next**. The `Q2Qorders.xsd` schema file is displayed in the URL in the **Messages** page, as shown in Figure 8-19.

Figure 8-19 The Adapter Configuration Wizard - Message Page



14. Click **Next**. The **Finish** page is displayed.

15. Click **Finish**. You have configured a JMS inbound adapter service.

Creating an Outbound Adapter Service

Perform the following steps to create an adapter service to enqueue the request messages and dequeue the corresponding response messages (report) from a queue:

1. Drag and drop **JMS Adapter** from the Components window into the Exposed Services swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter **Outbound** in the **Service Name** field, and click **OK**. The **JMS Provider** page is displayed.
4. Select **Oracle Weblogic JMS** in the Oracle Enterprise Messaging Service (OEMS) box, and click **Next**. The Service Connection page is displayed.
5. Select the connection created in Creating an Application Server Connection, and click **Next**. The Adapter Interface page is displayed.
6. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
7. Select **Produce Message**, and click **Next**. The Produce Operation Parameters page is displayed.
8. Click **Browse** and select **SendQueue** in the Destination field. The Produce Operation Parameters page is displayed.
9. Click **Next**. The **Messages** page is displayed.
10. Click **Browse** at the end of the URL field. The **Type Chooser** dialog is displayed.
11. Select **Project Schema Files**, **Q2Qorders.xsd**, and **Items**.
12. Click **Next**. The `Q2Qorders.xsd` schema file is displayed in the URL in the Message dialog.
13. Click **Next**. The **Finish** page is displayed.
14. Click **Finish**. You have configured the JMS adapter service, and the `composite.xml` page is displayed.

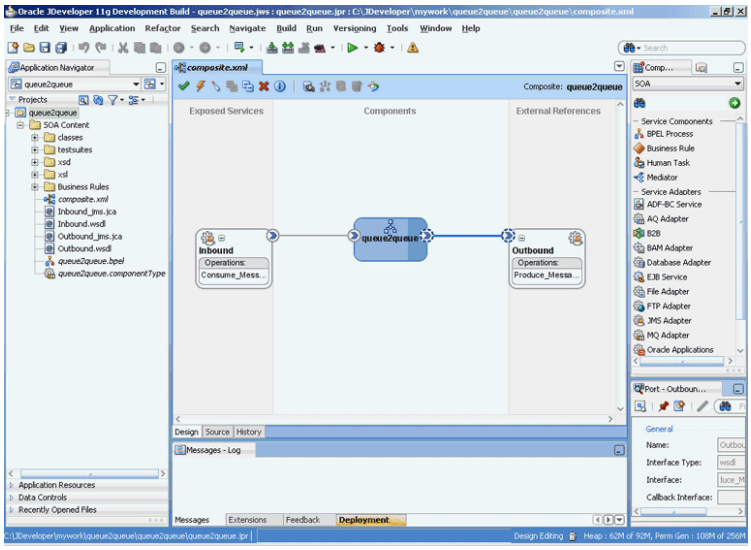
Wiring Services and Activities

You must wire the three components that you have created, Inbound adapter service, BPEL process, and Outbound adapter reference. Perform the following steps to wire components:

1. Drag the small triangle in the inbound Oracle JMS Adapter component in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the outbound Oracle JMS Adapter in the External References area.

   The JDeveloper Composite.xml is displayed, as shown in Figure 8-20.

Figure 8-20 The JDeveloper - Composite.xml



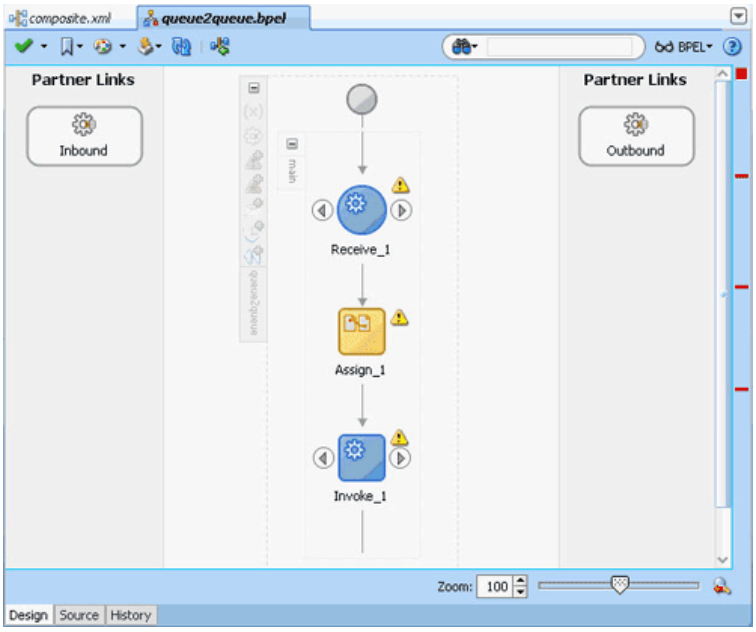
3. Click **File**, **Save All**.

4. Double-click **queue2queue**.

   The queue2queue.bpel page is displayed.

5. Drag and drop the **Receive**, **Assign**, and **Invoke** activities in the order mentioned from the Components window to the Components area, as shown in Figure 8-21.

Figure 8-21 The queue2queue.bpel Page



6. Double-click **Receive**.

   The **Receive** dialog is displayed.

7. Click the **Browse Partner Links** icon at the end of the Partner Link field.

The **Partner Link Chooser** dialog is displayed.

8. Select **Inbound**, and then click **OK**.

   The **Receive** dialog is displayed with the Partner Link field populated with the value Inbound.

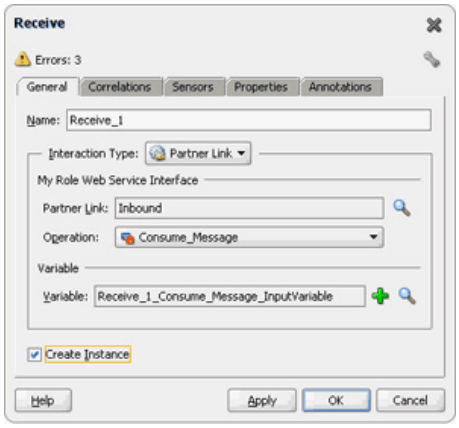
9. Click the **Auto-Create Variable** icon that is displayed at the end of the Variable field.

   The Create Variable dialog is displayed.

10. Accept the defaults, and click **OK**.

11. Select the **Create Instance** box, as shown in Figure 8-22, and click **OK**.

Figure 8-22 The Receive Dialog



12. Double-click the **Invoke** activity.

    The **Invoke** dialog is displayed.

13. Click the **Browse Partner Links** icon at the end of the Partner Link field.

    The **Partner Link Chooser** dialog is displayed.

14. Select **Outbound**, and then click **OK**.

    The **Invoke** dialog is displayed with the Partner Link field populated with the value Outbound.

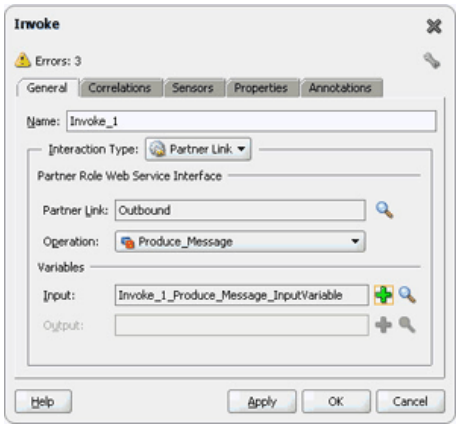
15. Click the **Automatically Create Input Variable** icon that is displayed at the end of the Input Variable field.

    The **Create Variable** dialog is displayed.

16. Accept the defaults, and click **OK**.

    The **Invoke** dialog is displayed, as shown in Figure 8-23.

Figure 8-23 The Invoke Dialog



17. Click **OK**.

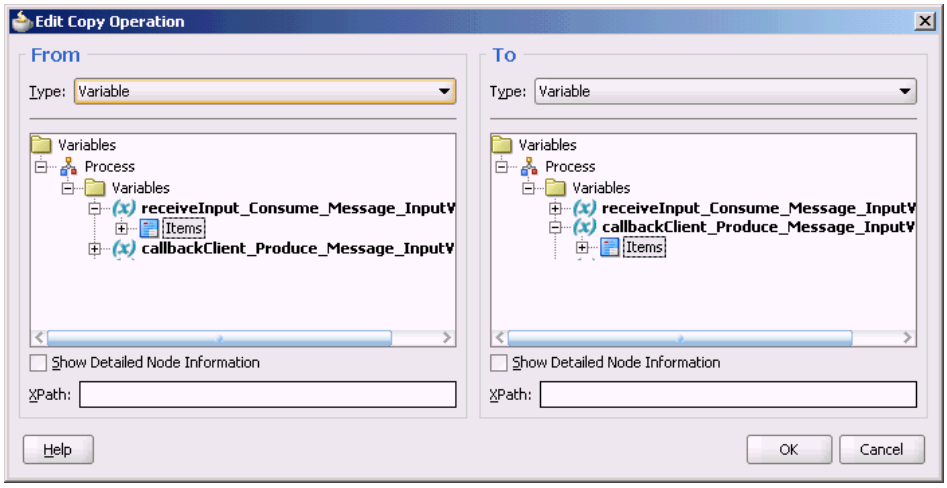
18. Double-click the **Assign** activity.

    The Assign dialog is displayed.

19. Click the plus icon, and select **Copy Operation**. The **Create Copy Operation** dialog is displayed.

20. Select the variables, as shown in Figure 8-24, and click **OK**.

Figure 8-24 The Create Copy Operation Dialog



21. Click **OK** in the Assign dialog.

22. Click **File**, **Save All**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, use the following steps:

1. Create an application server connection by using the procedure described in Creating an Application Server Connection for Oracle JCA Adapters.

2. Deploy the application by using the procedure described in Deploying Oracle JCA Adapter Applications from .

Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed composite by using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`.

   The composite you deployed is displayed in the Application Navigator.

2. In the Last 5 Instances pane, there is an entry of a new instance. This is the instance that was triggered when you enqueued a message using `queue2queue.java`.

3. Click one of the instances. The **Flow Trace** page is displayed.

4. Click the **TextMessage** component instance. The **Audit** page is displayed.

5. Click the **Flow-Debug** tab to debug the instance.

Accessing Queues and Topics from WLS JMS Server in a Remote Oracle WebLogic Server Domain

You can use the Oracle JMS Adapter to access remote WLS JMS destinations. Remote destinations refer to queues or topics that are defined in a WLS JMS server, which is part of a remote Oracle WebLogic Server domain.

To do so, ensure that you use the connector factory configured to interact to the remote WLS JMS server. You can achieve this by setting the <FactoryProperties> property of the connector factory defined in `weblogic-ra.xml` to remote server configuration, as shown in the following example:

```
<property>
  <name>FactoryProperties</name>
     <value>java.naming.factory.initial
                =weblogic.jndi.WLInitialContextFactory;
                       java.naming.provider.url= t3://<HOST>:<PORT>;
                    java.naming.security.principal=
                        <USERNAME>;java.naming.security.credentials
                                    =<PASSWORD>
     </value>
</property>
```

To enable the Oracle JMS Adapter to read from a remote queue that is present in a remote WLS JMS server, you must configure the following:

1. You must have a unique domain name and JMS server name in both the servers.

2. You must enable global trust between the two servers.

   Refer to the following link for information about how to enable global trust between servers:

   `http://download.oracle.com/docs/cd/E13222_01/wls/docs100/ConsoleHelp/taskhelp/security/EnableGlobalTrustBetweenDomains.html`

   This configuration is appropriate when you connect to queues or topics present in WLS9.2 server.

Note:

When using the JMS adapter to access WebLogic Server secure queues (local or remote domains), ensure that `java.naming.security.principal` and `java.naming.security.credentials` in the FactoryProperties property are setup correctly with a user who has access to the WLS secure queues.

JMS Adapter Limitations When a Remote Server is Used

The JMS Adapter enables you to interact with WebLogic Server JMS destination locations in a domain that are remote to the WebLogicServer domain where SOA is installed.

Two options are supported that enable you to access remote destinations using the JMS adapter:

- Direct access using specification of the `FactoryProperties` property in the `weblogic-ra.xml` file, with access parameters indicating the remote domain.
- Configuring the foreign server to access the remote domain.

For inbound use cases, both options are supported. For outbound use cases only, direct access is supported, but configuring the foreign server is not supported.

Synchronous/Asynchronous Request Reply Interaction Pattern

The Oracle JMS Adapter supports both synchronous and asynchronous request reply interaction patterns.

Synchronous Request Reply Pattern

You can use the Adapter Configuration Wizard to model a process that enables the Oracle JMS Adapter to be used in a synchronous request reply interaction pattern. In this case, the Oracle JMS Adapter sends a request to the request queue and waits for a response from the reply queue before further execution continues.

Underneath, the Oracle JMS Adapter uses a new interaction pattern, `JmsRequestReplyInteractionSpec`. This interaction spec allows for a request and reply destination name to be configured.

A variation enables usage of a temporary destination as part of the reply queue. Basically, this pattern enables an Oracle JMS Adapter to send a message to a JMS destination. In turn, the Adapter sets the `JMSReplyTo` header to the reply destination.

This value is then used by a third-party client to send the message to the reply destination which is then dequeued by the Oracle JMS Adapter.

When using the Oracle JMS Adapter in a synchronous pattern, you can use an XA ConnectionFactory for JMS or BPEL incoming events, and set the connector factory `isTransacted` property to `true` in `weblogic-ra.xml`. However, if you are using a web service, ensure that you use a non-XA ConnectionFactory, and set the connector factory `isTransacted` property to `true` in `weblogic-ra.xml`.

When you use the Oracle JMS Adapter in a synchronous pattern with Oracle WebLogic Server JMS, the connection factory must be `weblogic.jms.ConnectionFactory` or any other non-XA connection factory. Also, if Oracle WebLogic Server JMS is running in the local JVM (the same JVM as the adapter), then you must ensure that the connector factory `isTransacted` property is

set to `false` in `weblogic-ra.xml`. You can obtain the following samples by accessing the Oracle sample code site:

- `adapters-jms-106-wlsjms-syncrequestreply`
- `adapters-jms-107-wlsjms-syncrequestreplywithtemporaryreplydestination`

Asynchronous Request Reply Pattern

You can use the Adapter Configuration Wizard to model a process that allows Oracle JMS Adapter to be used in an asynchronous request reply interaction pattern.

Basically this pattern allows an JMS Oracle JMS Adapter to send a message to a JMS destination. When a message is received on the reply queue, the Oracle JMS Adapter can route messages to the correct composite or the component instance. The correlation is done based on the `JMSMessageID` of the request message, which becomes the `JMSCorrelationID` of the reply message, and the conversation ID of the underlying component.

For more information, see the following samples by accessing the Oracle sample code site:

- `adapters-jms-105-wlsjms-nativecorrelation`

AQ JMS Text Message

This use case demonstrates how the Oracle JMS Adapter dequeues from and enqueues to the AQ JMS Queue.

You can obtain the `adapters-jms-108-aqjms-textmessageusingqueues` sample by accessing the Oracle SOA Sample Code site.

This section includes the following topics:

- Meeting Prerequisites
- Create an Application Server Connection
- Creating an Application and an SOA Project
- Creating an Inbound Adapter Service
- Creating an Outbound Adapter Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using the Fusion Middleware Control Console

Meeting Prerequisites

You must perform the following prerequisites to complete this use case:

- Configuring AQ JMS in Oracle WebLogic Server Administration Console
- Creating Queues in Oracle Database

Configuring AQ JMS in Oracle WebLogic Server Administration Console

To configure AQ JMS in Oracle WebLogic Server Administration Console, you must perform the following steps:

- Adding an Oracle WebLogic JMS Module
- Adding an AQJMS Foreign Server to the JMS Module
- Configuring the AQJMS Foreign Server
- Adding Connection Factories to the AQ JMS Foreign Server
- Adding Destinations to the AQJMS Foreign Server

Adding an Oracle WebLogic JMS Module

Note that adding an Oracle WebLogic JMS module is optional. You can also create an AQJMS foreign server in a preexisting JMS module.

1. Navigate to the Oracle WebLogic Server JMS: `http://servername:portnumber/console`.
2. Use the required credentials to open the Home page of the Oracle WebLogic Server Administration Console.

   The **Home** page of the Oracle WebLogic Server Administration Console is displayed.
3. Navigate to **Services**, **Messaging**, **JMS Modules** in the Domain Structure pane.

   The **Oracle WebLogic Server Administration Console - JMS Modules** page is displayed.
4. Click **New** to create a new WebLogic JMS module.

   The **Oracle WebLogic Server Administration Console - Create JMS System Module** page is displayed.
5. Enter a name for the JMS module, and then click **Next**.

   The **Oracle WebLogic Server Administration Console-Create JMS System Module** page is displayed.
6. Select a target server where your SOA component is running, and then click **Next**.

   The **Oracle WebLogic Server Administration Console - Create JMS System Module** page is displayed.
7. Click **Finish**.

   You have created a JMS module.

Adding an AQJMS Foreign Server to the JMS Module

The next step is to add an AQ JMS foreign server to the JMS module by performing the following:

1. Click the JMS module that you created.

   The **Oracle WebLogic Server Administration Console - Settings for AQJMS Module** page is displayed.
2. Click **New** in the Summary of Resources table to create a new JMS system module resource.

The **Oracle WebLogic Server Administration Console - Create a New JMS System Module Resource** page is displayed.

3. Under **Choose the type of resource you want to create**, select **Foreign Server**, and click **Next**.

   The **Oracle WebLogic Server Administration Console - Create a New JMS System Module Resource** page is displayed.

4. In the **Name** field, enter a name for the foreign server, and then click **Finish**.

   The **Oracle WebLogic Server Administration Console - Settings for <JMS Module Name>** page is displayed.

Configuring the AQJMS Foreign Server

The next step is to configure the AQJMS foreign server that you created:

1. Click the AQ JMS Foreign Server listed under the Summary of Resources table.

   The **Oracle WebLogic Server Administration Console - Settings for TestAQJMS_ForeignServer** page is displayed.

2. Enter the following values:

   - **JNDI Initial Context Factory**:
     `oracle.jms.AQjmsInitialContextFactory`

     If the AQJMS Foreign Server is used by the WebLogic server side components, then you must configure a data source with this AQ JMS Foreign Server, by specifying the following values:

     In the **JNDI Properties** field, enter **datasource=<datasource jndi location>**. Replace the place holder with the JNDI location of your data source.

     However, if the AQJMS Foreign Server is used by WebLogic application client, then you must configure the JDBC URL with the AQ JMS foreign server you created.

   - **JNDI Connection URL**: Specify the URL that WebLogic Server uses to contact the JNDI provider.

     This value is required only if the AQJMS foreign server is used by the WebLogic application client.

   - **JNDI Properties Credential:** Specify any Credentials that must be set for the JNDI provider.

     This value is required only if the AQJMS foreign server is used by the Weblogic application client.

Note:

If you want to use an Oracle RAC database as adapter endpoint, the datasource pointed by the JNDI property, mentioned in the preceding step, must point to a multi data source.

Individual data sources and multi data sources used for such endpoints must use the recommended setting listed in Recommended Setting for Data Sources Used by Oracle JCA Adapters.

Adding Connection Factories to the AQ JMS Foreign Server

To add connection factories to the AQJMS foreign server:

1. In the Connection Factories tab in the Settings for <Foreign Server Name> page, select the AQJMS foreign server that you created.
2. Click **New**.

   The Oracle WebLogic Server Administration Console - Create a New Foreign JMS Connection Factory page is displayed.
3. In the **Name** field, enter a name for this connection factory. This is a logical name that would be referenced by Oracle WebLogic Server.
4. In the **Local JNDI Name** field, enter the local JNDI name that you would use in your application to look up this connection factory.

   Note:

   Ensure that you specify `aqjms/XAQueueConnectionFactory` for local JNDI name if you are connecting to a queue with JNDI name eis/aqjms/Queue that is provided with the sample use case, `AQQueuetoQueue`.

   Else, specify `aqjms/XATopicConnectionFactory` if you are connecting to a topic with JNDI name `eis/aqjms/Topic`.
5. In the **Remote JNDI Name** field, enter one of the following values depending on your requirement. If you use this connection factory in a global transaction, then use an XA-based connection factory, else use non-XA based connection factory.
   - `QueueConnectionFactory`
   - `TopicConnectionFactory`
   - `ConnectionFactory`
   - `XAQueueConnectionFactory`
   - `XATopicConnectionFactory`
   - `XAConnectionFactory`
6. Click **OK**.

Adding Destinations to the AQJMS Foreign Server

To add destinations to the AQJMS foreign server:

1. Click the **Destinations** tab in the Settings for <Foreign Server Name> page.
2. Click **New** and specify a name for this destination. This is a logical name that is referenced by the Oracle WebLogic Server and has nothing to do with the destination name.
3. In the **Local JNDI Name** field, enter the local JNDI name you would use in your application to look up this destination.

4. In the **Remote JNDI Name** field, enter `Queues/<queue name>`. If the destination is a queue, or enter `Topics/<topic name>` if the destination is a topic.
5. Click **OK**.
6. Restart the Oracle WebLogic Server Administration Console.

You have configured AQJMS in an Oracle WebLogic Server.

Creating Queues in Oracle Database

To create queues:

1. Run the `setup_user.sql` script.
2. Run the `create_start_queues.sql` script.

These scripts are located in the `adapters-jms-108-aqjms-textmessageusingqueues` sample `artifacts/sql` directory. You can obtain the `adapters-jms-108-aqjms-textmessageusingqueues` sample by accessing the Oracle SOA Sample Code site.

Create an Application Server Connection

You must establish connectivity between the design-time environment and the server you want to deploy to. Perform the steps mentioned in Creating an Application Server Connection for Oracle JCA Adapters to create an application server connection.

Creating an Application and an SOA Project

You must create an JDeveloper application to contain the SOA composite. Use the following steps to create a new application and an SOA project:

1. Open JDeveloper.
2. In the Application Navigator, click **New Application**. The Create Generic Application - Name your Application dialog is displayed.
3. Enter a name for the application in the **Application Name** field. For example, `AQQueue2Queue`.
4. In the Application Template list, choose **Generic Application**.
5. Click **Next**.

   The Name your project page is displayed.
6. In the **Project Name** field, enter a descriptive name. For example, `AQQueue2Queue`.
7. In the Available list in the **Project Technologies** tab, double-click **SOA** to move it to the Selected list.
8. Click **Next**. The Create Generic Application - Configure SOA Settings page is displayed.
9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

You have created a new application and an SOA project.

The Create BPEL Process page is displayed.

10. Enter a name for the BPEL process in the **Name** field.
11. Select **Define Interface Later** in the Template list, and then click **OK**.

    You have created a BPEL process.

    The `AQQueue2Queue` application, the `AQQueue2Queue` project, and the SOA composite appear in the design area.
12. Copy the `expense.xsd` file to the XSD folder in your project.

    This file is located in the `adapters-jms-108-aqjms-textmessageusingqueues` sample `artifacts/schemas` directory. You can obtain the `adapters-jms-108-aqjms-textmessageusingqueues` sample by acceding the Oracle SOA Sample Code site, and selecting the Adapters tab.

Creating an Inbound Adapter Service

Perform the following steps to create an adapter service to dequeue the message to a queue:

1. Drag and drop **JMS Adapter** from the Service Adapters list to the Exposed Services swim lane in the composite.xml page. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter **Inbound** in the **Service Name** field, and click **OK**. The JMS Provider page is displayed.
4. Select **Oracle Advanced Queueing** in the Oracle Enterprise Messaging Service (OEMS) box, and click **Next**. The Service Connection page is displayed.
5. Select the connection created in Creating an Application Server Connection.
6. Click **Next**. The Adapter Interface page is displayed.
7. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
8. Select **Consume Message**, and click **Next**. The Consume Operation Parameters page is displayed.
9. Click **Browse** and select **testInQueue** in the Destination field.
10. Click **Next**. The Messages page is displayed.
11. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
12. Select **Project Schema Files**, **expense.xsd**.
13. Click **Next**. The expenses.xsd schema file is displayed in the URL field in the Messages page.
14. Click **Next**. The Finish page is displayed.
15. Click **Finish**. You have configured a JMS inbound adapter service.

Creating an Outbound Adapter Service

Perform the following steps to create an adapter service that enqueues the request messages and dequeue the corresponding response messages (report) from a queue:

1. Drag and drop **JMS Adapter** from the Components window into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter **Outbound** in the **Service Name** field, and click **OK**. The JMS Provider page is displayed.
4. Select **Oracle Advanced Queueing** in the Oracle Enterprise Messaging Service (OEMS) box, and click **Next**. The **Service Connection** page is displayed.
5. Select the connection created in Creating an Application Server Connection, and click **Next**. The Adapter Interface page is displayed.
6. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
7. Select **Produce Message**, and click **Next**. The **Produce Operation Parameters** page is displayed.
8. Click **Browse** and select **testOutQueue** in the Destination field. The **Produce Operation Parameters** page is displayed.
9. Click **Next**. The **Messages** page is displayed.
10. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
11. Select **Project Schema Files**, **expense.xsd**.
12. Click **Next**. The expense.xsd schema file is displayed in the URL field in the Message dialog.
13. Click **Next**. The **Finish** page is displayed.
14. Click **Finish**. You have configured the JMS adapter service, and the composite.xml page is displayed.

Wiring Services and Activities

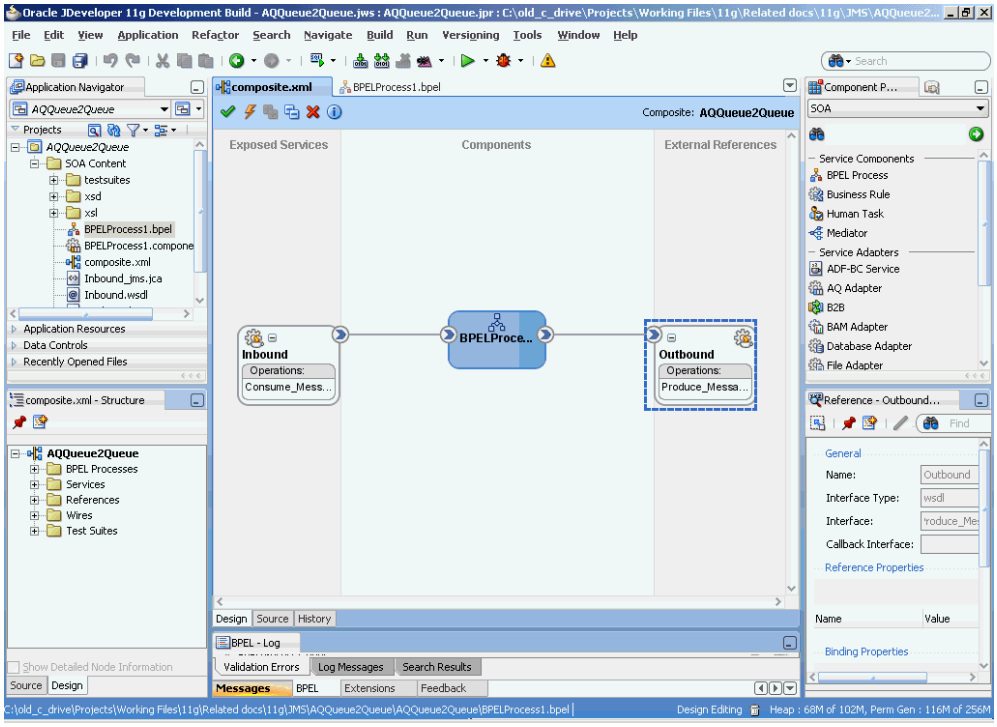
You must wire the three components that you have created: Inbound adapter service, BPEL process, and Outbound adapter reference.

Perform the following steps to wire the components:

1. Drag the small triangle in the inbound Oracle JMS Adapter component in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the outbound Oracle JMS Adapter in the External References area.

   The JDeveloper Composite.xml is displayed, as shown in Figure 8-25.

Figure 8-25 JDeveloper - Composite.xml



3. Click **File**, **Save All**.
4. Double-click the BPEL process. The BPELProcess1.bpel page is displayed.
5. Drag and drop the **Receive**, **Assign**, and **Invoke** activities, in the order mentioned, from the Components window to the Components area.
6. Double-click the **Receive** activity.

   The **Receive** dialog is displayed.
7. Click the **Browse Partner Links** icon at the end of the Partner Link field.

   The **Partner Link Chooser** dialog is displayed.
8. Select **Inbound**, and then click **OK**.

   The **Receive** dialog is displayed with the Partner Link field populated with the value *Outbound*.
9. Click the **Auto-Create Variable** icon that is displayed at the end of the Variable field. The **Create Variable** dialog is displayed.
10. Accept the defaults, and click **OK**.
11. Check the **Create Instance** box.
12. Double-click the **Invoke** activity to Outbound.

    The **Invoke** dialog is displayed.
13. Click the **Automatically Create Input Variable** icon that is displayed at the end of the Input Variable field.

14. Click the **Browse Partner Links** icon at the end of the Partner Link field.

    The Partner Link Chooser dialog is displayed.

15. Select **Outbound**, and then click **OK**.

    The Invoke dialog is displayed with the Partner Link field populated with the value Outbound.

16. Accept the defaults, and click **OK**.

17. Click **OK**.

18. Double-click the **Assign** activity.

    The Assign dialog is displayed.

19. Click the plus icon, and select **Copy Operation**.

    The Create Copy Operation dialog is displayed.

20. Select the variables, and click **OK**.

21. Click **OK** in the Assign dialog.

22. Click **File**, **Save All**.

Note:

When using Oracle JMS Adapter to dequeue from AQ JMS Topics with durable subscriptions, if you notice that the dequeue operation exhibits slow performance, then you can speed up the entire performance by using multiple inbound threads for each adapter service.

Oracle JMS Adapter allows multiple inbound threads if you specify an endpoint property `adapter.jms.receive.threads`.

However, this workaround is not applicable when using non-durable subscriptions because doing using the workaround in that scenario results in duplicate messages.

Note:

When resiliency is enabled and Oracle Event Delivery Network (EDN) subscriber is in quiescent mode, the EDN consumer threads enter into sleep mode. Based on the configuration, for example,`MaxStuckThreadTime = 600` (seconds), the sleeping EDN consumer threads are marked as STUCK when the configured `MaxStuckThreadTime` is elapsed. These STUCK logging statements in server log are temporary warnings, and have no functional impact. Once the quiescence mode is over and resiliency resumes incoming request processing, these STUCK warnings vanish, and normal processing is resumed.

SOA_Request_WM and SOA_EDN_WM also share the same max constraint. When the number of threads of EDN subscribers is larger than the max constraint, it impacts SOA_Request_WM performance. The work around is either reduce the number of threads of EDN subscribers or increase the max constraint defined by SOAIncomingRequests_maxThreads

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile by using JDeveloper, perform the following steps:

1. Create an application server connection by using the procedure described in Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application by using the procedure described in Deploying Oracle JCA Adapter Applications from .

Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed composite by using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed is displayed in the Application Navigator.
2. In the Last 5 Instances pane, there is an entry of a new instance. This is the instance that was triggered when you enqueued a message by using `AQQueue2Queue.java`.
3. Click one of the instances. The Flow Trace page is displayed.
4. Click the TextMessage component instance. The Audit page is displayed.
5. Click the **Flow-Debug** tab to debug the instance.

Accessing Queues and Topics Created in 11*g* from the OC4J 10.1.3.4 Server

This section describes the procedure for accessing queues and topics you created in Oracle Application Server 11*g* or 12.1.3 from OC4J 10.1.3.4.

To do this, you must configure Oracle BPEL PM JMS adapter with Oracle WebLogic Server.

The following are the steps to configure Oracle BPEL PM JMS adapter with Oracle WebLogic Server:

1. Create the `wlfullclient.jar` file using the following steps:

   a. Change to the `server/lib` directory, as shown in the following example:

   ```
   cd WL_HOME/server/lib
   ```

   b. Use the following command to create the `wlfullclient.jar` file in the `server/lib` directory:

   ```
   java -jar ../../../modules/com.bea.core.jarbuilder_X.X.X.X.jar
   ```

   where `X.X.X.X` is the version number of the jarbuilder module in the *`WL_HOME`*`/server/lib` directory. For example:

   ```
   java -jar ../../../modules/com.bea.core.jarbuilder_1.0.1.0.jar
   ```

2. Copy the `wlfullclient.jar` file to the 10.1.3.4. server at the following location:

   ```
   <ORACLEAS_HOME>/j2ee/<OC4J_INSTANCE>/connectors/
                JmsAdapter/JmsAdapter
   ```

3. Configure the connector factory setting in the `oc4j-ra.xml` file, as shown in the following example:

   ```
   <connector-factory location="eis/wlsjms/Queue"
   connector-name="Jms Adapter">
      <config-property name="connectionFactoryLocation"
   value="weblogic.jms.ConnectionFactory"/>
      <config-property name="factoryProperties"
   value="java.naming.factory.initial=weblogic.jndi.
   WLInitialContextFactory;java.naming.provider.url=t3:
                              //<WLS-SERVER-NAME>: <WLS-SERVER-PORT>;
   java.naming.security.principal=<USER>;
                  java.naming.security.credentials=
                           <PASSWORD>"/>
          <config-property name="acknowledgeMode"
                                   value="AUTO_ACKNOWLEDGE"/>
          <config-property name="isTopic" value="false"/>
          <config-property name="isTransacted" value="false"/>
          <config-property name="username" value=""/>
          <config-property name="password" value=""/>
      <connection-pooling use="none">
      </connection-pooling>
        <security-config use="none">
        </security-config>
   </connector-factory>
   ```

   Note:

   The `isTransacted` configuration property value must typically be set to `FALSE`. Currently, XA integration with WebLogic JMS is not supported unless the adapter is deployed on the Oracle WebLogic Server. Also note that `<WLS-SERVER-NAME>` must be replaced by the actual Weblogic Server name hosting the queues, and `<WLS-SERVER-PORT>` must be replaced by the actual port value for Weblogic Server hosting the queues.

4. Modify the `server.xml` file of the 10.1.3.4 server to include the `environment-naming-url-factory-enabled="true"` property, as shown in the following example:

```
<application-server
...
...
environment-naming-url-factory-enabled="true"
...
>
```

5. Restart the 10.1.3.4 server to ensure the changes occur.

Configuring the 11G Server or Later Server to Access Queues Present in 10.1.3.X OC4J

You can configure your 11G or later server to access queues present in 10.1.3.x OC4J with the following steps.

Copy Jar Files into the domains Folder of the Web Logic Server

Copy the following jar files under the `domains/<DOMAIN_NAME>/lib` folder of the WebLogic Server:

- `$J2EE_HOME/lib/jms.jar`
- `$J2EE_HOME/lib/jta.jar`
- `$J2EE_HOME/oc4jclient.jar`
- `$AS_HOME/opmn/lib/optic.jar`

Add Connector factory in the weblogic-ra.xml File

The next step is to add the Connector Factory in the `weblogic-ra.xml` file:

```
<connection-instance>
   <jndi-name>eis/oc4jjms/Queue</jndi-name>
    <connection-properties>
       <properties>
         <property>
          <name>ConnectionFactoryLocation</name>
          <value>jms/XAQueueConnectionFactory</value>
         </property>
         <property>
          <name>FactoryProperties</name>
            <value>java.naming.factory.initial=com.
evermind.server.rmi.RMIInitialContextFactory;
java.naming.provider.url= <PROVIDER_URL>;
java.naming.security.principal=oc4jadmin;
        java.naming.security.credentials=welcome1</value>
        </property>
          <property>
          <name>AcknowledgeMode</name>
          <value>AUTO_ACKNOWLEDGE</value>
       </property>
       <property>
         <name>IsTopic</name>
         <value>false</value>
       </property>
       <property>
         <name>IsTransacted</name>
         <value>false</value>
       </property>
       <property>
```

```
        <name>Username</name>
        <value>oc4jadmin</value>
      </property>
      </property>
      <property>
        <name>Password</name>
       <value>welcome1</value>
     </property>
    </properties>
   </connection-properties>
  </connection-instance>
```

where `<PROVIDER_URL>=opmn://localhost:6003` or `ormi://localhost:12401` to use against a specific node or, `opmn:ormi://localhost:6003:oc4j_soa` to use against the `oc4j_soa` instance.

Accessing Distributed Destinations (Queues and Topics) on the WebLogic Server JMS

A distributed destination is a set of destinations (queues, set of physical JMS queue members, or topics, set of physical JMS topic members) that are accessible as a single, logical destination to a client.

Note:

For more information on distributed destinations, and a definition of terms used in this context, visit the Using Distributed Destinations pages at http://download.oracle.com/docs/cd/E13222_01/wls/docs103//jms/dds.html

---

The JMS Adapter can process messages addressed to a distributed destination member after receiving available notification; it can process available, unavailable, and failure notifications related to a distributed destination member.To have the JMS Adapter process such messages when working with Distributed Topics, you must provide additional properties.

When you provide additional properties, you can separate multiple FactoryProperty values with a semicolon. See the following example.

```
<property>
  <name>FactoryProperties</name>
  <value>ClientID=SOAClient2;TopicMessageDistributionAll=true</value>
</property>
```

Also, in scenarios where the JMS adapter interacts with multiple WLS-managed servers in a cluster, you must specify all servers as part of the `FactoryProperties` property. These are in turn used to establish correct context for lookup of JMS artifacts; see the following example:

```
<property>
  <name>FactoryProperties</name>
  <value> java.naming.factory.initial=weblogic.jndi.
          WLInitialContextFactory;
          java.naming.provider.url=t3://<server1>:<port1>,
               <server2>:<port2>;
          java.naming.security.principal=
               <username>;java.naming.security.credentials
                     =<password>
  </value>
</property>
```

Replace the brackets <> with values applicable for your environment.

Providing JMS Adapter Access to Distributed Topics

You use three FactoryProperty parameter values to provide adapter access to distributed topics, to specifically enable the Client ID to be shared by multiple connections, to enable the sharing of Durable subscriptions among multiple subscribers, and to specify whether you want one copy of a message per application or per endpoint. The properties include:

- `ClientIDPolicy`

  Use the FactoryProperties parameter `ClientIDPolicy` property with a value of `UNRESTRICTED` to enable the Client ID to be shared by multiple connections. The default, if no value is specified, is `UNRESTRICTED`. The non-default value is `RESTRICTED`. The default is used in almost all cases, so typically you do not have to set it. See the following example:

```
</property>
   <name>FactoryProperties</name>
<value>ClientIDPolicy=UNRESTRICTED</value>

 </property>
```

- `SubscriptionSharingPolicy`

  Use the FactoryProperties parameter with a value of `SHARABLE` to enable the sharing of Durable Subscriptions among multiple subscribers.

  A value of SubscriptionSharingPolicy `EXCLUSIVE` means you cannot share Durable Subscriptions among multiple subscribers. If you do not specify a value, the default is `SHARABLE`; in most cases, you do not have to change the value.

```
 <property>
    <name>FactoryProperties</name>
    <value>SubscriptionSharingPolicy=
               SHARABLE</value>
 </property>
```

- `TopicMessageDistributionAll`

  See the section on Distributed Topics for more information on the `TopicMessageDistributionAll` FactoryProperties parameter. You can set it as in the following example:

```
<property>
    <name>FactoryProperties</name>
    <value>TopicMessageDistributionAll=true</value>
</property>
```

The JMS Adapter with Distributed Queues and Distributed Topics

Specific inbound and outbound queue and error handling behaviors apply to the JMS Adapter with WebLogic Server JMS Distributed Queues and Distributed Topics.

For inbound queues, the JMS Adapter creates an inbound poller thread and registers a notification listener with the WebLogic Server JMS on endpoint activation; it unregisters notification listener upon endpoint deactivation.

Note:

Internally, that consumer is pinned to a member of the Distributed Queue. You must deploy the adapter with a large number of threads so that all members of the distributed queues could be accounted for.

From the SOA 11.1.1.4.0 version forward, the JMS Adapter fully supports both Distributed Queues and Topics. Newer versions of the JMS Adapter rely on notifications from the WebLogic Server JMS to create and remove consumers for the Distributed Destination members.

Comparing JMS adapter behavior in SOA 11.1.1.3.0 and prior versions (where a consumer is created randomly for more than one member of Distributed Queue) with the new behavior in SOA 11.1.1.4.0 and later, there is no change, except that the JMS Adapter is now able to account for all members without relying on your starting the JMS Adapter with large numbers of poller threads at activation.

The JMS Adapter handles errors in the Distributed environment in the same fashion as such errors are handled in a non-Distributed environment: retriable exceptions lead to message retry; non-retriable exceptions lead to message rejection.

There is no change from the behavior of other Adapters to JMS adapter behavior when the Adapter produces a message to a Distributed Queue.

JMS messages for Distributed Destinations are produced by creating a MessageProducer for the Distributed Destination and not for a specific member.

Outbound errors are processed based on fault-policies previously defined for the outbound reference.

For inbound adapters with distributed topics, the JMS Adapter registers a notification listener with the WebLogic Server JMS on endpoint activation. The JMS Adapter creates an inbound poller thread for each available notification received from WebLogic Server JMS for a Distributed Topic member.

The inbound poller thread stops working and necessary cleanup is performed if an unavailable notification is received for the member for which the poller thread was created. The durable subscription is maintained in a similar fashion as in a non-Distributed topic scenario.

The Adapter unregisters the notification listener upon endpoint deactivation. Any message arriving at a Distributed Topic is processed based on the various settings used and the type of Distributed Destination in use: either one copy of a message per application, or one copy of a message per adapter endpoint.

The behaviors for each of these types of Distributed Destination follow.

One Copy of a Message Per Application (Default Behavior)

The default behavior for WebLogic Server Partitioned Distributed Topics when used with the JMS Adapter is to provide one copy of a message per application. Each message must be processed exactly once (that is, there is no duplicate processing.). In this scenario, where there is one copy of a message per application, the client id and subscription name are the same for every Distributed Destination and each adapter instance creates subscriptions on every member. The name is unique and immutable across server restarts.When using Partitioned Distributed Topics you must configure the JMS adapter to use unrestricted clientid and shared subscription policy. These two are the default settings for the JMS Adapter.When using Replicated Distributed

Topics, you must configure JMS adapter to use the unrestricted clientid and shared subscription policy, which are the default settings. In addition, you must specify the following message selector, `NOT JMS_WL_DDForwarded` when defining an activation spec.

To achieve better performance, you should use Partitioned Distributed Topics.

Refer to the following example, consisting of a snippet of a connection instance from the `weblogic-ra.xml` file for a local cluster:

```
<property>
    <name>FactoryProperties</name>
    <value>ClientID=SOAClient1;</value>
</property>
```

One Copy Of a Message Per Adapter Endpoint

The second type of scenario you can employ with Distributed Topics is to have one copy of message per adapter endpoint. In this case, either the client id or the subscription name is unique for each adapter instance. The unique part of the member name is immutable across server restarts.

When using Partitioned Distributed Topics you have to configure the JMS Adapter to use unrestricted clientid and shared subscription policy, which are the default settings.

At the same time, to achieve subscription name uniqueness, the JMS Adapter requires that the property `TopicMessageDistributionAll` (default value of false) is set to true. You can define this property by setting the `FactoryProperties` property of the connection instance in the `weblogic-ra.xml` file.

An example usage (a snippet of a connection instance from a `weblogic-ra.xml` file for a local cluster) is:

```
<name>FactoryProperties</name>
<value>ClientID=SOAClient2;
       TopicMessageDistributionAll=true</value>
</property>
```

To achieve better performance, you should use Partitioned Distributed Topics.

When using Replicated Distributed Topics, configure the JMS Adapter to use unrestricted clientid and shared subscription policy, which are the default settings.

At the same time, to achieve subscription name uniqueness, the JMS adapter requires that the property `TopicMessageDistributionAll` (default value of false) is set to true. You can define this property by setting the `FactoryProperties` property of the connection instance in `weblogic-ra.xml`. An example usage (snippet of connection instance from `weblogic-ra.xml` for a local cluster) is:

```
<name>FactoryProperties</name>
<value> ClientID=SOAClient2;
       TopicMessageDistributionAll=true</value>
</property>
```

In addition, you must specify the message selector, `NOT JMS_WL_DDForwarded` when defining an activation spec.

Specifying the Message Selector when Defining an Activation Spec

Specify a message selector when defining an activation spec. The message selector is required when you create one copy of message per adapter Endpoint.

To specify the selector, use the JMS Adapter Wizard when modeling a composite application that reads from Replicated Distributed Topic The metadata for the message selector you specify are captured in the `.jca` file.

The following is an example of a message selector defined in an activation spec.This message selector filters out the copy of the forwarded message when sending a message to a destination subscriber.

This message selector is only applicable for when using Replicated Distributed Topics.

```
<activation-spec className="oracle.tip.adapter.jms.inbound.JmsConsumeActivationSpec">
     <property name="DestinationName" value="jms/DemoInTopic"/>
     <property name="UseMessageListener" value="false"/>
     <property name="DurableSubscriber" value="dsub1"/>
     <property name="MessageSelector"
                     value="NOT JMS_WL_DDForwarded"/>
     <property name="PayloadType" value="TextMessage"/>
</activation-spec>
```

With Distributed Topics, retriable exceptions lead to message retry, while non-retriable exceptions lead to message rejection.

Available/Unavailable/Failure notification does not impact the working of the outbound adapter reference. The message is produced by creating a MessageProducer for the Distributed Destination and not for a specific member.

In the Distributed Topics environment, as elsewhere, an error is processed based on the fault policies defined for the outbound reference.

Compatibility and Migration

Remote Distributed Queue support is feasible back to WLS JMS version 9.0 using the DestinationAvailabilityListener API. A remote Distributed Topic cannot be supported if it is older than WebLogic 10.3.4, as “shared subscriptions", “unrestricted client ids", the “not forwarded" selector, and even “partitioned" Distributed Topics are not supported. You must instead directly reference a DT member JNDI name, and deal with the single subscriber per subscription limitation.

Configuring Oracle JMS Adapter with IBM WebSphere Default JMS Provider

This section describes how to configure Oracle JMS Adapter for IBM WebSphere 7.x JMS.

1. Copy the following files from under the `<WAS_INSTALL DIR>/fmwwas-nd/websphere/runtimes` directory to the `SOAInstall_DIR>/user_projects/domains/<DOMAIN_NAME>/`lib folder:

   - `com.ibm.jaxws.thinclient_7.0.0.jar`
   - `com.ibm.ws.admin.client_7.0.0.jar`
   - `com.ibm.ws.ejb.thinclient_7.0.0.jar`
   - `com.ibm.ws.jpa.thinclient_7.0.0.jar`
   - `com.ibm.ws.messagingClient.jar`
   - `com.ibm.ws.orb_7.0.0.jar`
   - `com.ibm.ws.sib.client.thin.jms_7.0.0.jar`

- com.ibm.ws.sib.client_ExpeditorDRE_7.0.0.jar
- com.ibm.ws.webservices.thinclient_7.0.0.jar
- ejb3exceptions.jar
- sibc.jmsra.rar
- sibc.nls.zip

2. Configure the connector factory by modifying the weblogic-ra.xml file in the soa/connectors/JmsAdapter.rar, as shown in the following example

Example - Configure the Connection Factory

```
<connection-instance>
<jndi-name>eis/wasspherejms/Queue</jndi-name>
   <connection-properties>
      <properties>
         <property>
           <name>ConnectionFactoryLocation</name>
           <value><QUEUE_CONECTION_FACTORY></value>
         </property>
         <property>
            <name>FactoryProperties</name>
            <value>java.naming.provider.
           url=iiop://<HOST_NAME>:<PORT>;
java.naming.factory.initial=com.ibm.
           websphere.naming.WsnInitialContextFactory;
java.naming.security.principal=<USERNAME>;
java.naming.security.credentials=<PASSWORD>;
ThirdPartyJMSProvider=true
            </value>
          </property>
          <property>
              <name>AcknowledgeMode</name>
              <value>AUTO_ACKNOWLEDGE</value>
          </property>
          <property>
             <name>IsTopic</name>
             <value>false</value>
          </property>
          <property>
             <name>IsTransacted</name>
             <value>true</value>
         </property>
         <property>
             <name>Username</name>
             <value><USERNAME></value>
         </property>
         <property>
            <name>Password</name>
            <value><PASSWORD></value>
        </property>
        </properties>
    </connection-properties>
</connection-instance>

<connection-instance>
<jndi-name>eis/wasspherejms/Topic</jndi-name>
   <connection-properties>
      <properties>
```

```
            <property>
              <name>ConnectionFactoryLocation</name>
              <value><TOPIC_CONECTION_FACTORY></value>
            </property>
            <property>
               <name>FactoryProperties</name>
               <value>java.naming.provider.url=
               iiop://<HOST_NAME>:<PORT>;
java.naming.factory.initial=com.ibm.websphere.
               naming.WsnInitialContextFactory;
java.naming.security.principal=<USERNAME>;
java.naming.security.credentials=<PASSWORD>;
ThirdPartyJMSProvider=true
               </value>
             </property>
             <property>
                 <name>AcknowledgeMode</name>
                 <value>AUTO_ACKNOWLEDGE</value>
             </property>
             <property>
                <name>IsTopic</name>
                <value>true</value>
             </property>
             <property>
                <name>IsTransacted</name>
                <value>true</value>
            </property>
            <property>
                <name>Username</name>
                <value><USERNAME></value>
            </property>
            <property>
               <name>Password</name>
               <value><PASSWORD></value>
           </property>
           </properties>
       </connection-properties>
   </connection-instance>
```

<QUEUE_CONECTION_FACTORY> and <TOPIC_CONECTION_FACTORY> refer to the JNDI name of the Queue and Topic connection factory, respectively created in WebSphere 7.0 for the default JMS provider.

Alternatively, you can configure a new connection factory by using the Oracle WebLogic Server Administration Console, and use the steps mentioned in Adding an Adapter Connection Factory.

Note:

The JMS Adapter can only be used in non-XA mode when interacting with WebSphere 7.x JMS.

Configuring Request-Reply in the JMS Adapter

The Request-Reply configuration feature enables you to perform the following:

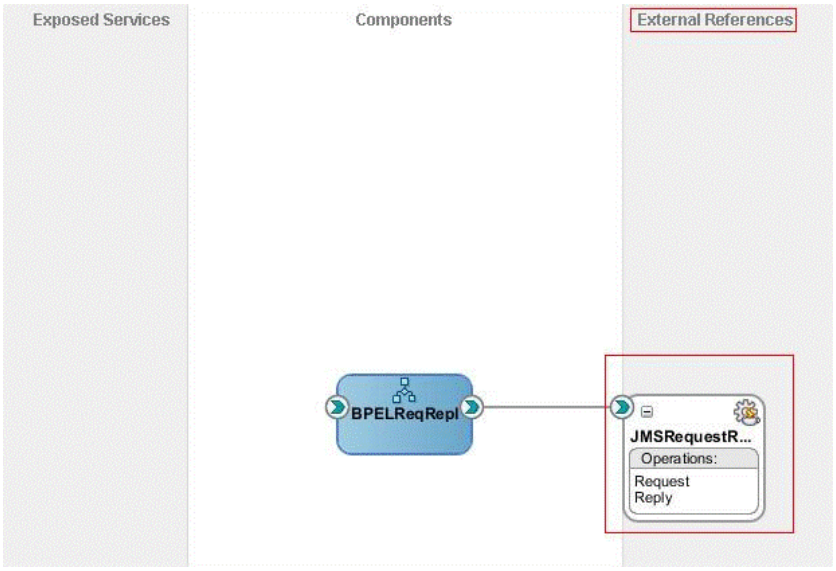
- Combine Request and Reply in a single configuration step. In the prior releases of the Oracle SOA Suite, you would require to configure two distinct adapters.

- Automatic correlation without your needing to configure BPEL correlation set. This works seamlessly in Mediator and BPMN as well.

To configure the JMS Adapter Request-Reply feature:

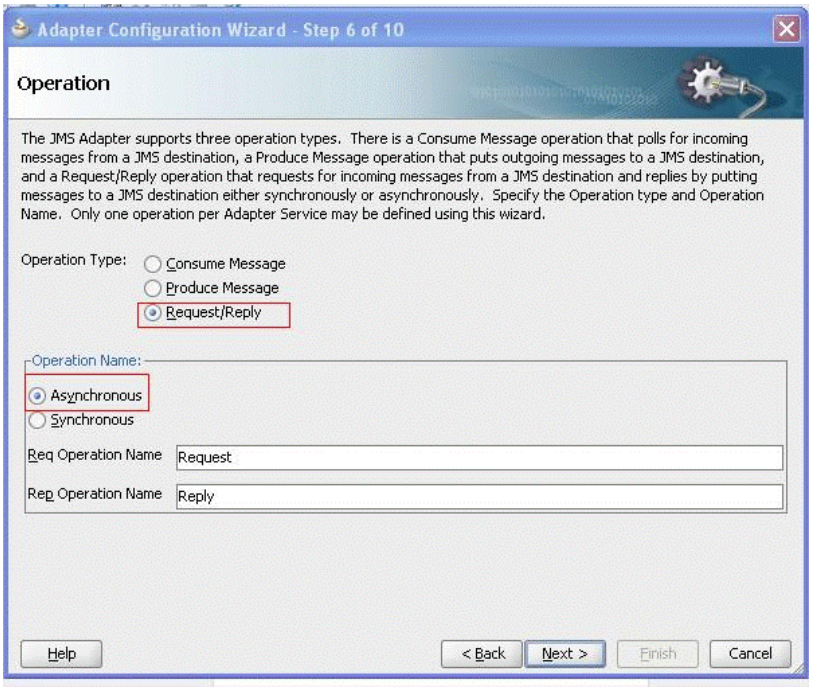
1. Drag and drop a JMS Adapter onto the External References swim lane in the JDeveloper composite editor.

Figure 8-26 Dragging and Dropping a JMS Adapter into External References Swimlane



2. Enter default values for the first few screens in the JMS Configuration Adapter wizard until you reach the screen where the JMS Configuration Adapter wizard prompts you to enter the operation name. Select Request-Reply as the “Operation Type" and Asynchronous as “Operation Name".

Figure 8-27 Operations Screen for Request/Reply



3. Select the Request and Reply queues in the following screens of the wizard. The message is enqueued in the Request queue and the reply is returned in the Reply queue.

Figure 8-28 The Request Operation Parameters Screen

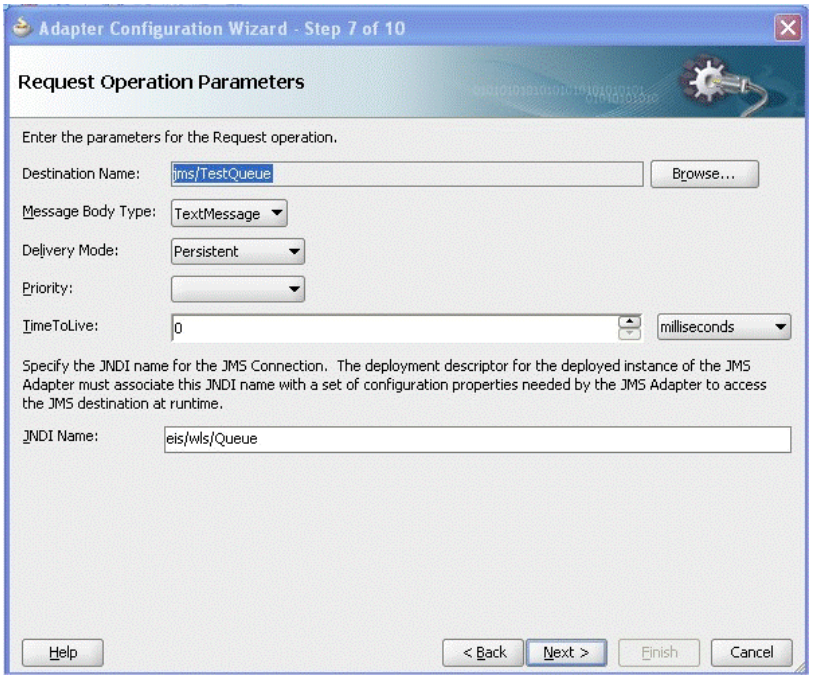
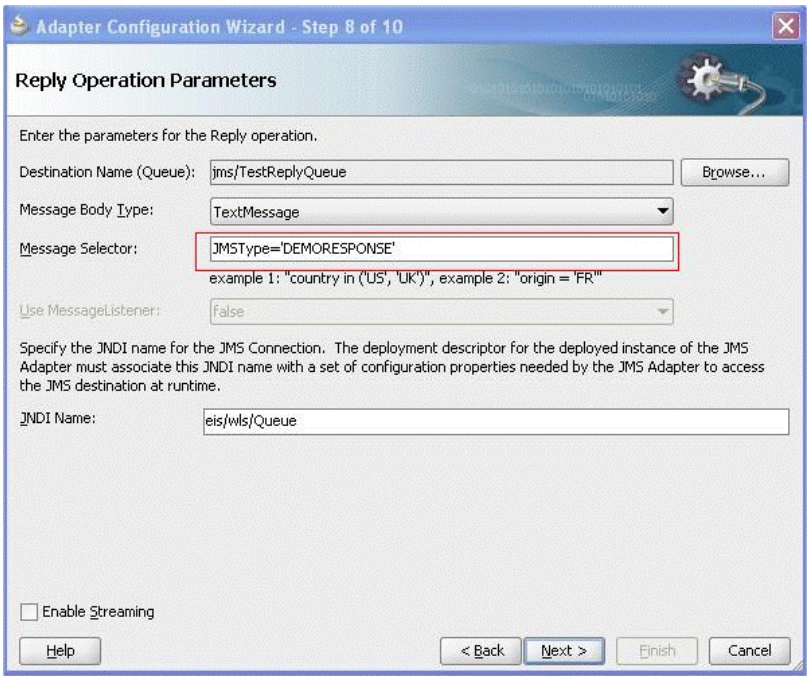


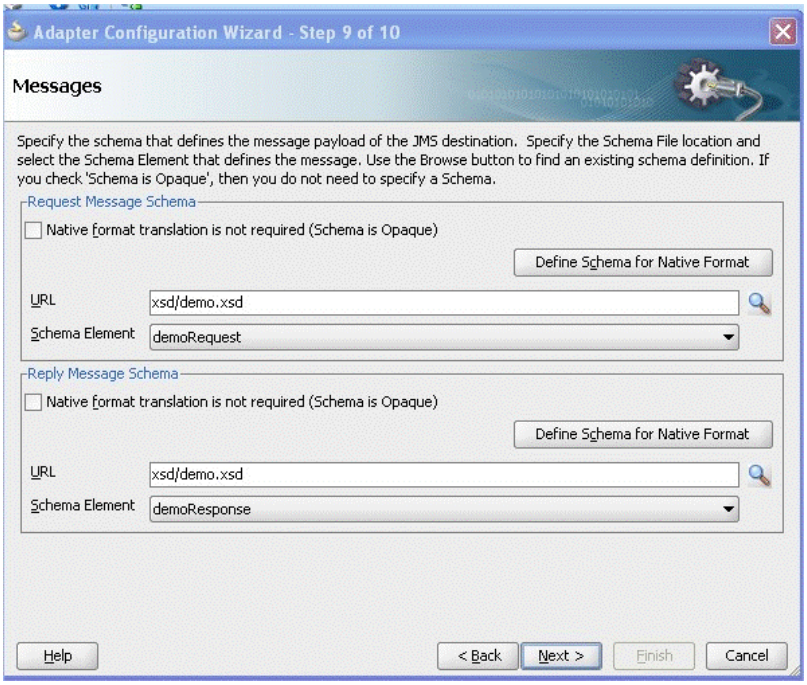
Figure 8-29 Reply Operation Parameters Screen



The reason we have used such a selector is that the back-end system that reads from the request queue and generates the response in the response queue actually generates more than one response and hence we must use a filter to exclude the unwanted responses.

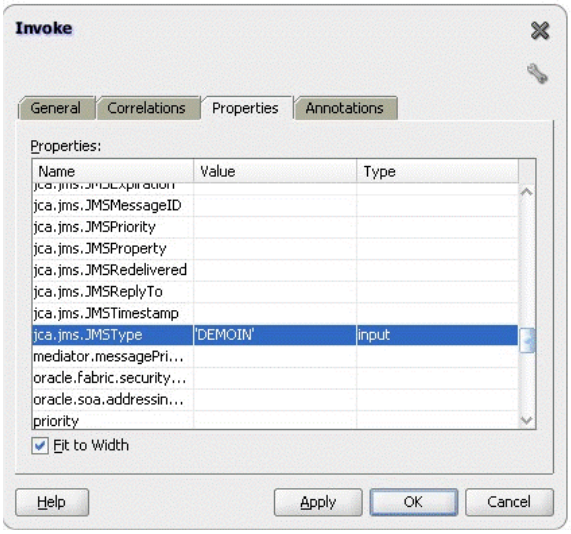
4. Select the message schema for and a response.

Figure 8-30 Selecting Message Schema for Request and for Response



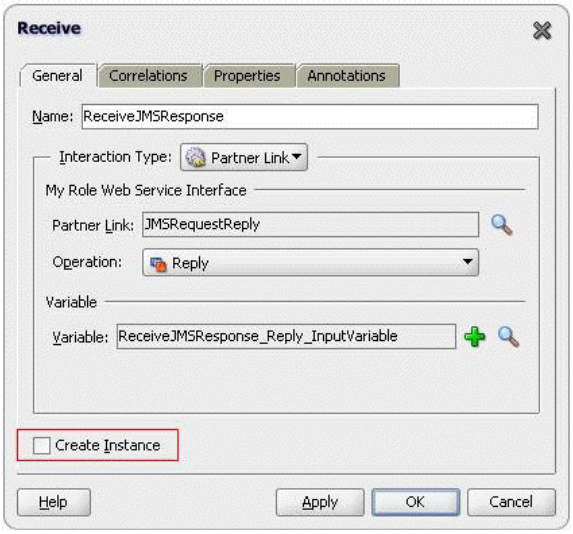
5. Add an <invoke> activity in BPEL corresponding to the JMS Adapter partner link. An additional header is set as the third-party application used requires this.

Figure 8-31 Invoke BPEL Properties Dialog Corresponding to the JMS Adapter Link



Add a <receive> activity just after the <invoke> activity, and select the Reply operation. Ensure that the **Create Instance** option is unchecked.

Figure 8-32 Receive Dialog for Reply Operation



Using the WLS JMS Unit-Of-Order with the JMS Adapter

You can use the SOA JMS Adapter to produce messages that create a specific unit-of-order for the messages.The Adapter InteractionSpec supports a property called `UnitOfOrder`, which you configured when modelling an adapter reference, through the **Produce Operations** page of the JMS Adapter Configuration Wizard. See the description of this page in the section, "Produce Message Procedure".

You use this `UnitOfOrder` property to set the unit-of-order value for the MessageProducer when producing a message. The new field is only visible if the JMS provider is WebLogic Server JMS.

The JMS Adapter enables you to override this unit-of-order used when producing a message to a WLS destination.

To perform this override, the JMS Adapter supports a Normalized message property called `jca.jms.WeblogicUnitOfOrder`. This value overrides the value specified using the property `UnitOfOrder` for the `JmsProduceInteractionSpec`.

You can modify the value of the `UnitOfOrder` spec property from the EM console. Any outbound interactions after that point use the new value you supply.

If you define neither the property `jca.jms.WeblogicUnitOfOrder` nor the `JmsProduceInteractionSpec` property `UnitOfOrder`, no unit-of-order value is set by JMS Adapter and the default unit-of-order, if specified administratively on the connection factory and destination, takes effect.

The JMS unit-of-order feature only works with the WebLogic Server. For non Web Logic Server destinations, the property `jca.jms.WeblogicUnitOfOrder` (if one exists) are ignored.

Getting a Unit of Order Property

You can obtain user-specified value from the interaction spec instance.

All JMS message properties are available as Normalized Message properties. You can get the unit-of-order property by looking up the Normalized message property `jca.jms.JMSProperty.JMS_BEA_UnitOfOrder` on the delivered message.

JMS Synchronous Consume

The Oracle JMS Adapter supports the synchronous consume (mid-process receive) interaction pattern.

You can use the Adapter Configuration Wizard to model a process that enables the Oracle JMS Adapter to be used in such a synchronous consume interaction pattern. In this case, the Oracle JMS Adapter dequeues a message from a specified queue while the process is executing (hence the name mid-process receive).

If the queue is empty, execution still continues. Underneath, the Oracle JMS Adapter uses a new interaction pattern `JmsReceiveNoWaitInteractionSpec`.

When modeling the JMS Synchronous Consume using the JMS Adapter Configuration Wizard, you must provide the queue name used for the dequeue. For this operation, the JMS Adapter supports all the other jca properties configured when the current Consume operation is configured.

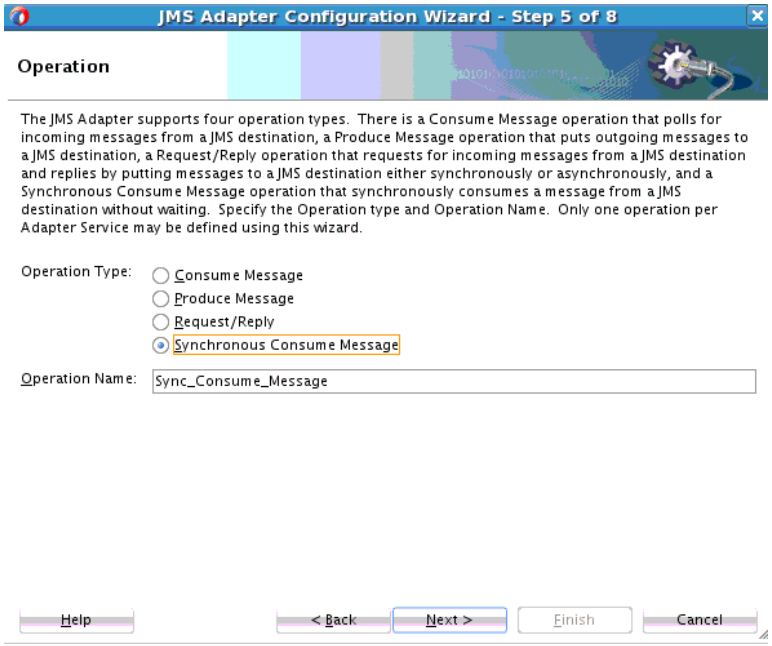
In a scenario using this operation, the BPEL Process continues executing the next step in the process if the JMS Adapter did not have any messages on the queue to dequeue.

Configuring JMS Synchronous Consume

Follow these steps to configure the JMS Synchronous Consume operation using the JMS Adapter Configuration Wizard.

1. On the JMS Adapter Configuration Wizard Screen, select **Synchronous Consume Message**.

Figure 8-33 JMS Adapter Configuration Wizard Operation Screen, with Synchronous Consume Message Selected



2. The JMS Adapter Configuration Wizard **Synchronous Consume Parameters Screen** appears, where you can enter parameters for the Synchronous Consume Screen. Enter a Message

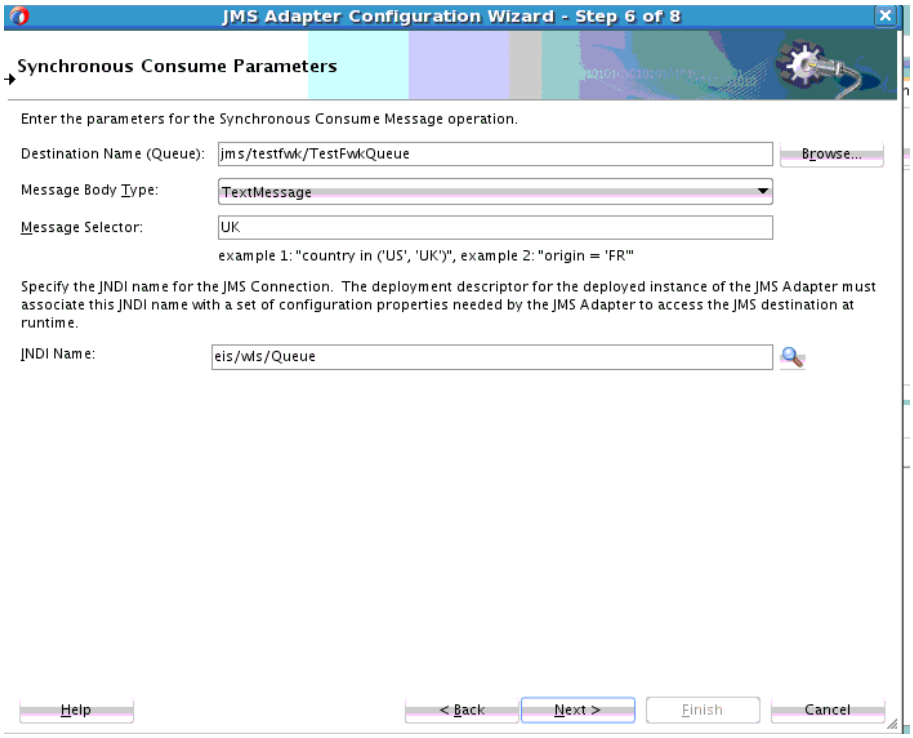
   - Enter a **Destination Name** or Select **Browse** to open the Destination Browser.
   - Select a Message Body Type from the Chooser. Choices are:
     - **TextMessage**
     - **BytesMessage**
     - **MapMessage**
   - Enter a **Message Selector**.

     This field is optional. It filters messages based on header and property information. The message selector rule is a Boolean expression. If the expression is `true`, then the message is consumed. If the expression is `false`, then the message is rejected.

     Some examples are supplied on the screen. For other examples, you can enter logic, such as:

     - JMSPriority > 3. Based on this, messages with a priority greater than 3 are consumed; all other messages are rejected.
     - JMSType = 'car' AND color = 'blue' AND weight > 2500
     - Country in ('UK', 'US', 'France')
   - Specify a JNDI name. This is also filled in when you select a Destination Name.

Figure 8-34 The JMS Adapter Configuration Synchronous Consume Screen



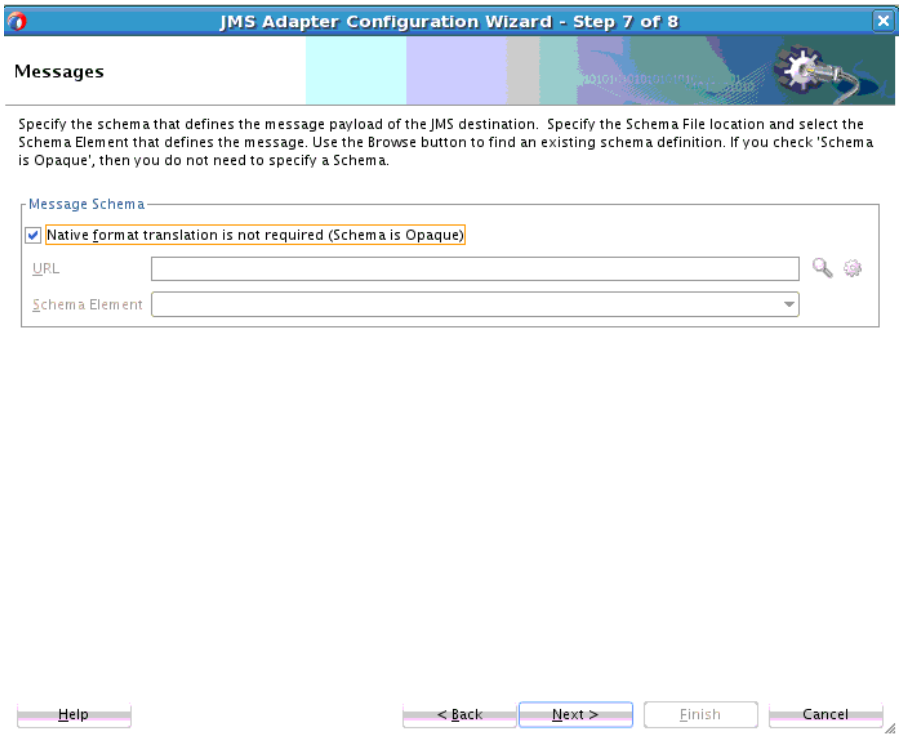
3. If you selected Browse to open the Destination Browser, on this Browser:

   - Choose a Destination Type by selecting the **Destination Type: All Types, Queues, or Topics**.
   - Supply a **Destination** or click Search to search from those Destinations that populate the browser. Click Show All to show expand the list of Destinations.
   - Select a **Destination**, and click **OK** to use that Selection in the **Synchronous Parameters** page.

Note:

When using the Oracle JMS Adapter in a synchronous pattern, you can use an XA ConnectionFactory for JMS or BPEL incoming events, and set the connector factory `isTransacted` property to true in `weblogic-ra.xml`. However, if you are using a web service, ensure that you use a non-XA ConnectionFactory, and set the connector factory `isTransacted` property to true in `weblogic-ra.xml`.

Figure 8-35 The JMS Adapter Configuration Wizard Select Destination Browser Screen



4. The JMS Adapter Configuration Wizard displays the JMS Adapter Messages Screen. On this screen, select **Native Format is Not Required (Schema is Opaque)** or browse to find a schema or open the Native Format Builder.

9

Oracle JCA Adapter for Database

This chapter describes the Oracle JCA Adapter for Database. In addition, it contains references to use cases for the Oracle Database Adapter and for stored procedures.

This chapter includes the following topics:

Introduction to the Oracle Database Adapter

The Oracle Database Adapter enables SOA Composite Applications to communicate with Oracle databases or third-party databases through JDBC.

This section includes the following topics:

Functional Overview

This section provides a functional overview of the Oracle Database Adapter. The Oracle Database Adapter enables Oracle SOA Suite and Oracle Fusion Middleware to communicate with database end points. These include Oracle database servers and any relational databases that follow the ANSI SQL standard and which provide JDBC drivers.

The principle of the tables and views in the Oracle Database Adapter is to expose to SOA tables and SQL as transparently and non-intrusively as possible. From an integration standpoint, tables and SQL are what relational database products have in common, so a generic solution focused on what is standard has the greatest reach. In exposing databases to SOA, it is also about combining the technologies of SQL and XML, the former an ideal language for querying information, the latter an ideal format for transporting and representing information. While stored procedure support is less

standard across databases, Oracle Database Adapter provides support for stored procedures as the guide describes.

The Oracle Database Adapter is a JCA 1.5 connector, which runs on the Oracle WebLogic Server. It relies on an underlying JDBC connector/driver to enact the database communication. In contrast to JDBC, it is non-programmatic. The interaction (series of `SELECT`, `UPDATE`, `INSERT`) is loosely modeled using the Adapter Configuration Wizard. The inputs/outputs are XML, most easily seen as input parameters and result sets converted to XML. These XML inputs and outputs allow the Oracle Database Adapter services to be plugged into Oracle Fusion Middleware.

To access an existing relational schema, you must create an application and an SOA project to use the Adapter Configuration Wizard to perform the following:

- Import a relational schema (one or more related tables) and map it as an XML schema (XSD)

  For more information, see Relational-to-XML Mapping.

- Abstract SQL operations such as `SELECT`, `INSERT`, and `UPDATE` as Web services

  For more information, see SQL Operations as Web Services.

- Have database events initiate an Oracle Fusion Middleware process.

Although Oracle Streams Advanced Queuing (Oracle AQ) is an Oracle Database feature, you use the separate, specialized Oracle JCA Adapter for AQ to integrate with Oracle AQ. For more information, see Overview .

For non-relational and legacy systems (with a few exceptions such as DB2 on AS/400), application and mainframe adapters are available. For more information about application and mainframe adapters, see:

- Legacy Adapters
- Packaged-Application Adapters
- Oracle E-Business Suite Adapter

For more information on the Oracle Database Adapter, see:

- Features
- Concepts
- Complete Walkthrough of the Database Adapter Configuration Wizard

Oracle Database Adapter Integration with Oracle BPEL PM

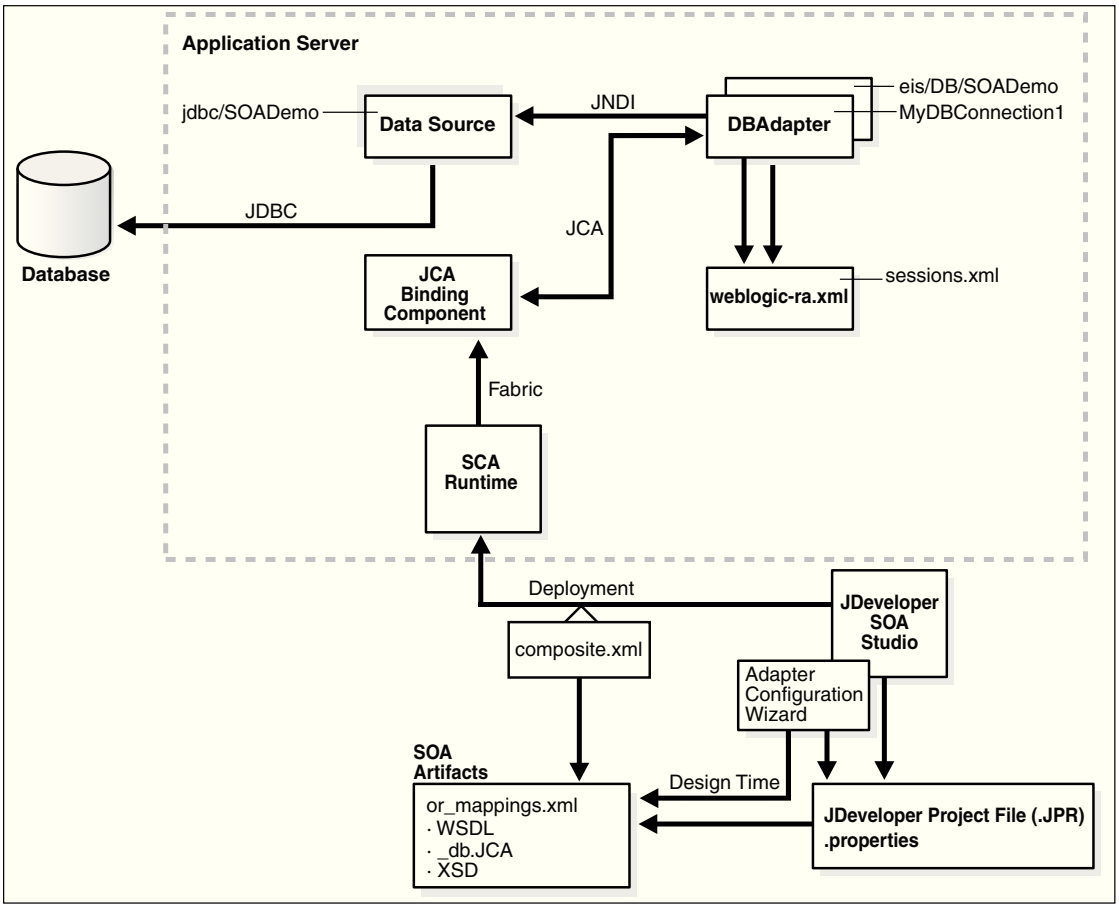
When the Oracle Database Adapter is used to poll for database events (usually an `INSERT` operation on an input table) and initiate a process, it is called an exposed service. In Oracle BPEL process it is a partner link tied to a Receive activity. The expression `inbound` (from database into SOA) is commonly used.

When the Oracle Database Adapter is used to invoke a one-time DML statement such as `INSERT` or `SELECT`, in a Mediator component or an SOA composite, it is called a service reference. In Oracle BPEL process, it is a partner link tied to an Invoke activity. The expression `outbound` (from SOA out to the database) is used.

Design Overview

This section provides an overview of the design of the Oracle Database Adapter. Figure 9-1 shows how the Oracle Database Adapter interacts with the various design-time and deployment artifacts.

Figure 9-1 How the Oracle Database Adapter Works



The Oracle Database Adapter is a JCA 1.5 connector, which is deployed to the application server during installation.

The Oracle Database Adapter consists of multiple instances; each instance represents a connection to a database end point. Different SOA processes may point to the same adapter instance (database), while different service endpoints in a SOA process may point to different adapter instances (databases).

Because each adapter instance points to a single database, there is a one-to-one correspondence from adapter instances to application server data sources. Out of the box there is a single Oracle Database Adapter instance named `eis/DB/SOADemo`, which points to the data source `jdbc/SOADataSource`.

The list of adapter instances is stored in a deployment descriptor file, `weblogic-ra.xml` on Oracle WebLogic Server. (It is inside of `DbAdapter.rar`, which contains also the Java class files in `DBAdapter.jar`). Configuring an Oracle Database Adapter instance is more about creating the underlying data source: getting the correct JDBC driver and connection URL.

For more information, see JDBC Driver and Database Connection Configuration.

However `weblogic-ra.xml` entries occasionally have more than simply the name of the underlying data source. These properties are detailed further under Database Adapter Deployment.

While at runtime you have Oracle Database Adapter instances, at design time you have the Adapter Configuration Wizard (link). You can run it once to generate a single adapter service end point, and then multiple times in edit mode to make incremental changes to each. It generates all the adapter related artifacts required when deploying a SOA composite as Table 9-1 lists.

Table 9-1 Adapter Configuration Wizard Generated SOA Composite Adapter Artifacts

File	Description
`<serviceName>.wsdl`	This is an abstract WSDL, which defines the service end point in terms of the name of the operations and the input and output XML elements.
`<serviceName>_table.xsd`	This contains the XML file schema for these input and output XML elements. Both these files form the interface to the rest of the SOA project.
`<serviceName>_or-mappings.xml`	This is an internal file. It is a TopLink specific file, which is used to describe the mapping between a relational schema and the XML schema. It is used at runtime.
`<serviceName>_db.jca`	This contains the internal implementation details of the abstract WSDL. It has two main sections, location and operations. Location is the JNDI name of an adapter instance, that is, `eis/DB/SOADemo`. Operations describe the action to take against that end point, such as `INSERT`, `UPDATE`, `SELECT`, and `POLL`. The contents of the `db.jca` file are wholly determined by choices made while running the Adapter Configuration Wizard.
`<serviceName>.properties`	This is also an internal file. It is created when tables are imported, and information about them is saved. It is used only at design time. At runtime, the location is used to look up the adapter instance which executes the service. Based on the properties in the `db.jca` file and the linked `or-mappings.xml` file, `<seviceName>.properties` file generates the correct SQL to execute, parses the input XML, and builds an output XML file matching the XSD file. To execute the SQL, it obtains a pooled SQL connection from the underlying data source.

Complete Walkthrough of the Database Adapter Configuration Wizard

This section describes the Adapter Configuration Wizard and how you can define an Oracle Database Adapter by using the Adapter Configuration Wizard.

This section describes the various Oracle Database Adapter concepts through a use case, which is, a complete walkthrough of the Adapter Configuration Wizard. In addition, this use case also describes how by using the Adapter Configuration Wizard, you can import tables from the database, specify relationships spanning multiple tables, generate corresponding XML schema definitions, and create services to expose the necessary SQL or database operations. These services are consumed to define

partner links that are used in the BPEL process. You use the Adapter Configuration Wizard to both create and edit adapter services.

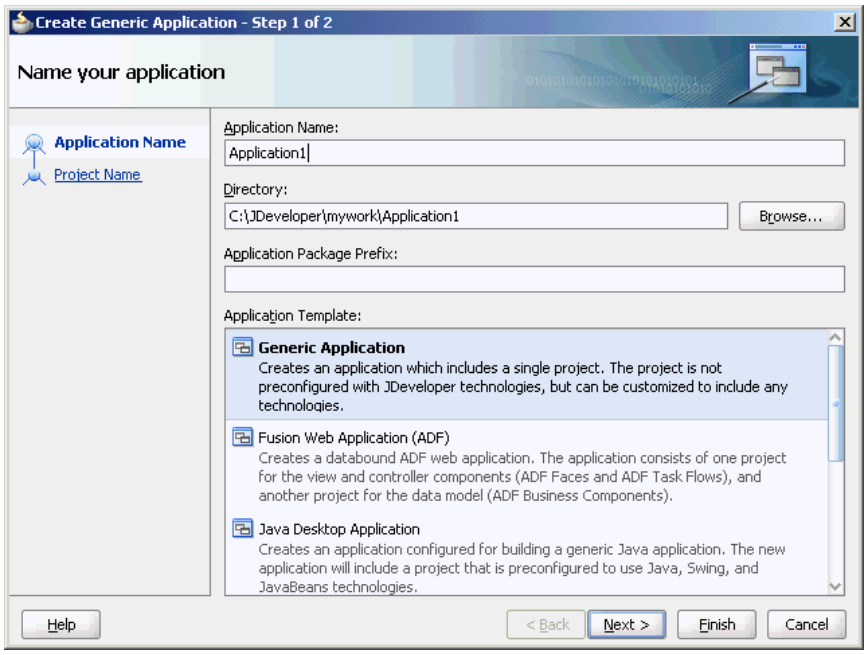
Creating an Application and an SOA Project

You must create an Oracle JDeveloper (JDeveloper) application to contain the SOA composite. Perform the following steps to create an application, and an SOA project:

1. Open JDeveloper.
2. In the Application Navigator, click **New Application**.

   The Create Generic Application - Name your application page is displayed, as shown in Figure 9-2.
3. Enter a name for the application in the **Application Name** field.
4. In the Application Template list, choose **Generic Application**.

Figure 9-2 The Create Generic Application - Name your application Page



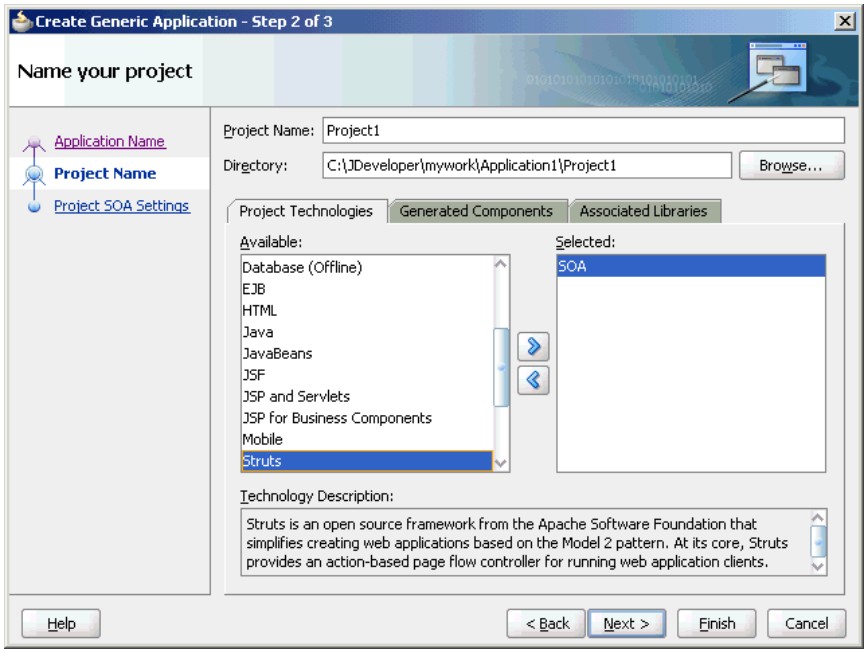
5. Click **Next**.

   The Create Generic Application - Name your project page is displayed, as shown in Figure 9-3.

6. In the **Project Name** field, enter a descriptive name.

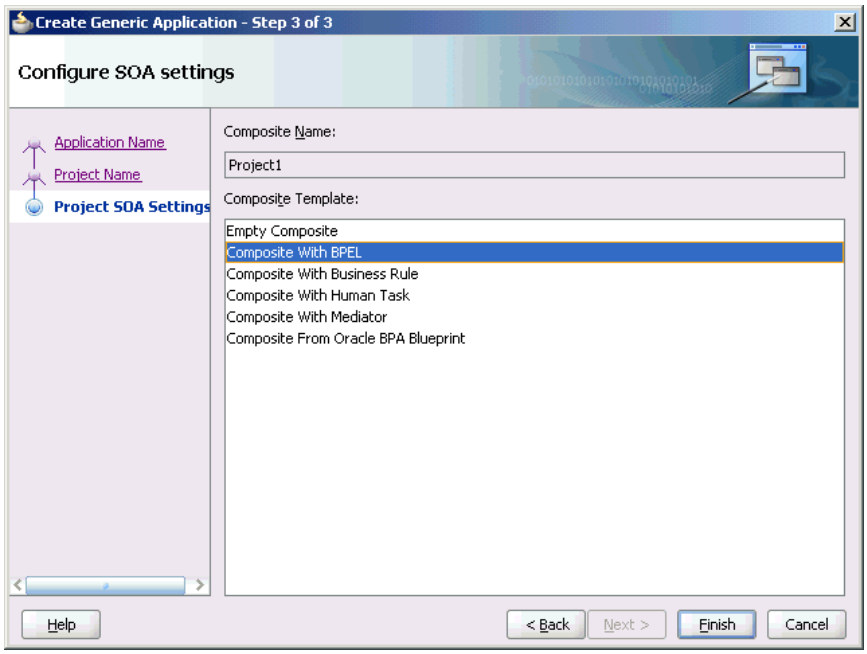
7. In the **Available** list in the **Project Technologies** tab, double-click **SOA** to move it to the Selected list.

Figure 9-3 The Create Generic Application - Name your Generic project Page



8. Click **Next**. The Create Generic Application - Configure SOA settings page is displayed, as shown in Figure 9-4.

Figure 9-4 The Create Generic Application - Configure SOA Settings Page

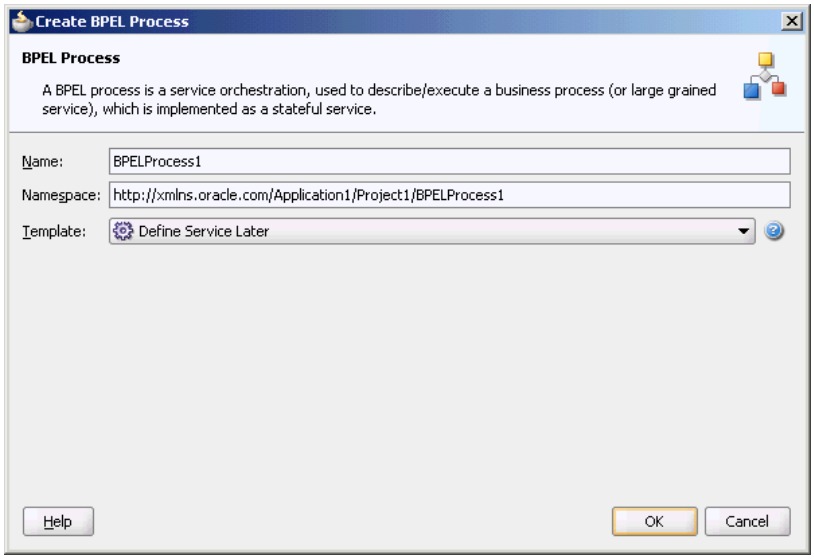


9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

   You have created a new application and an SOA project. This automatically creates an SOA composite.

   The Create BPEL Process page is displayed, as shown in Figure 9-5.

Figure 9-5 The Create BPEL Process Page



10. Enter a name for the BPEL process in the **Name** field.

11. Select **Define Service Later** in the Template list, and then click **OK**.

You have created a BPEL process.

Defining an Oracle Database Adapter

The next step is to define an Oracle Database Adapter service. Perform the following steps to create an Oracle Database Adapter service:

1. In the Component Palette, select **SOA**.
2. Drag and drop **Database Adapter** from the Service Adapters list to the Exposed components swim lane in the composite.xml page.

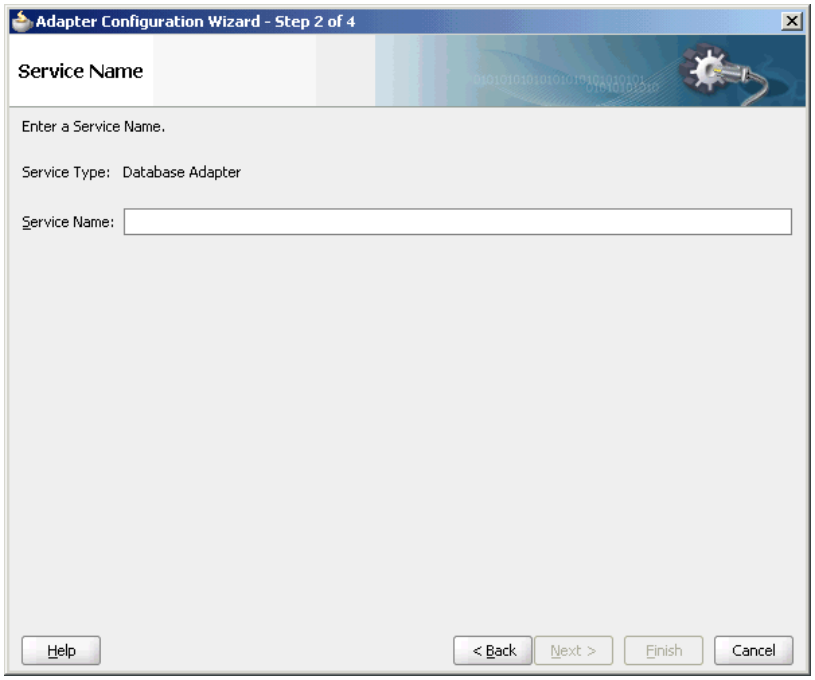
   The Adapter Configuration Wizard is displayed.

   Note:

   To create an Oracle Database Adapter service as part of a BPEL process, drag and drop a BPEL process from Service Components onto Components. Double-click it. Then, in the BPEL Component Palette, drag and drop Database Adapter from BPEL Services onto a Partner Links swim lanes.

3. Click **Next**. The Service Name page is displayed, as shown in Figure 9-6. Enter the following information:

   Figure 9-6 Specifying the Service Name



4. In the **Service Name** field, enter a service name, and then click **Next**. The Service Connection page is displayed.

See Connecting to a Database to continue using the Adapter Configuration Wizard.

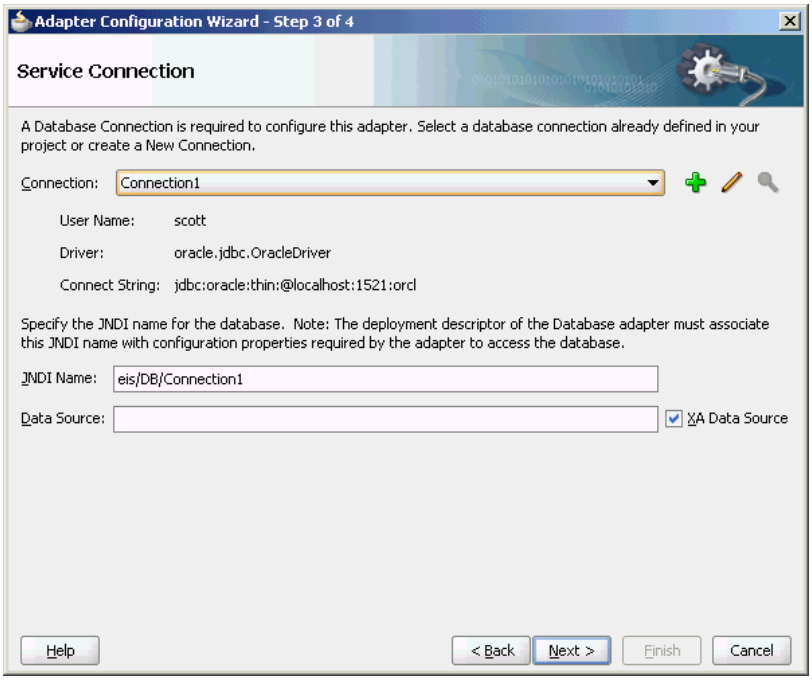
Connecting to a Database

Figure 9-7 shows where you select the database connection that you are using with the service. This is the database from which you import tables to configure the service. This is the database from which you import tables to configure the service. You can re-create it here in each new JDeveloper application you create.

You can provide a Java Naming and Directory Interface (JNDI) name to identify the database connection, as the default name that is provided is `eis/DB/<ConnectionNameInJDev>`.

For more information, see Database Adapter Deployment.

Figure 9-7 The Adapter Configuration Wizard: Service Connection Page



Note the following:

- In production environments, it is recommended that you add the JNDI entry to the adapter deployment descriptor (`weblogic-ra.xml`). This way, the Oracle Database Adapter is more performant by working in a managed mode.

  For information about creating a data source and an outbound connection pool, see Adding an Adapter Connection Factory.

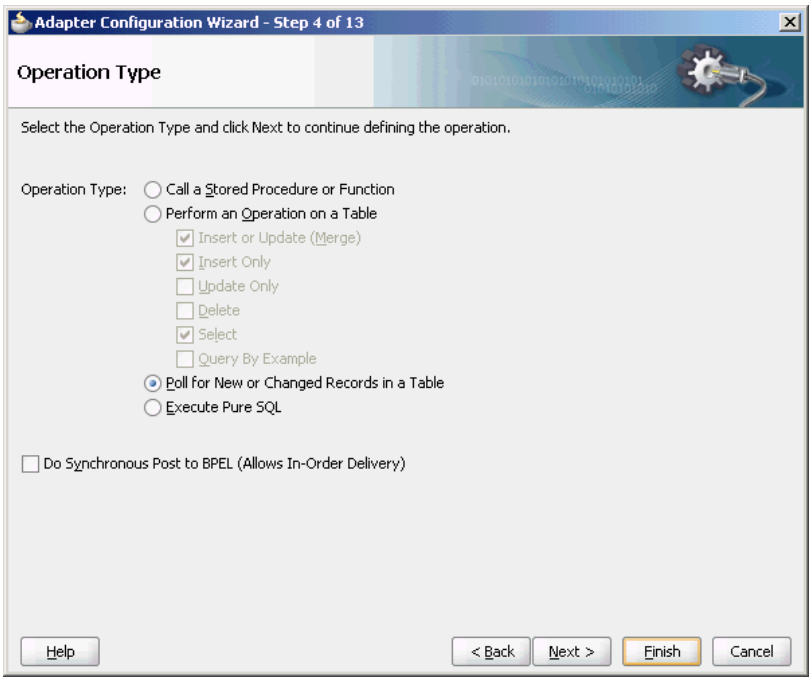
- When you click **Next**, a connection to the database is attempted. If a connection cannot be made, you cannot proceed to the next window, even if you are editing an existing partner link.

See Selecting the Operation Type to continue using the Adapter Configuration Wizard.

Selecting the Operation Type

Figure 9-8 shows where you indicate the type of operation you want to configure for this service.

Figure 9-8 The Adapter Configuration Wizard: Operation Type Page



The following operation types are available:

- **Call a Stored Procedure or Function**

  Select this option if you want the service to execute a stored procedure or function. For more information, see Stored Procedure and Function Support.

- **Perform an Operation on a Table**

  Select this option for outbound operations. You can select **Insert or Update**, **Insert Only**, **Update Only**, **Delete**, **Select**, or any combination of the six. These operations loosely translate to SQL `MERGE`, `INSERT`, `UPDATE`, `DELETE`, and `SELECT` operations.

  For more information, see DML Operations.

  Note:

  The operation `Update Only` sometimes performs inserts/deletes for child records. That is, an update to Master could involve a new or deleted detail. So if the input to update contains only one detail record, then the other detail records in the table are deleted.

- **Poll for New or Changed Records in a Table**

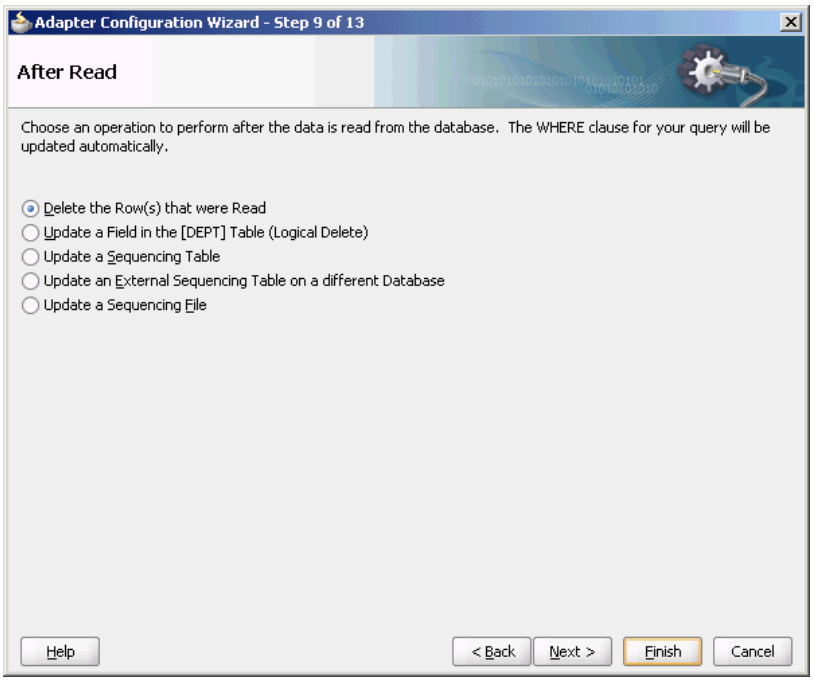
  Select this option for an inbound operation (that is, an operation that is associated with a Receive activity). This operation type polls a specified table and returns for processing any new rows that are added. You can also specify the polling frequency.

  For more information, see Polling Strategies.

  The following is a list of polling operations that you can perform after the data is read from the database, as shown in Figure 9-9:

- Delete the Row(s) that were Read
- Update a Field in the [Table_Name] Table (Logical Delete)
- Update a Sequencing Table
- Update an External Sequencing Table on a Different Database
- Control Table Strategy

Figure 9-9 Polling Operations



- **Execute Pure SQL**

  Useful when dealing with arbitrarily complex statements, aggregate queries (result is not row-based), and `XMLType` columns. See Pure SQL - XML Type Support to follow this usage of the Adapter Configuration Wizard.

  Note:

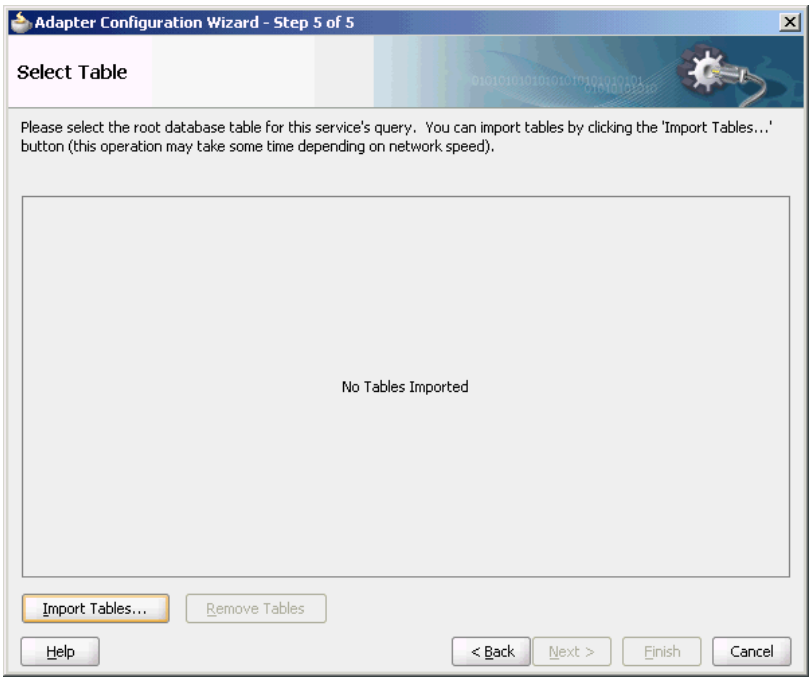
  Schema Bound XML tables are not supported.

Otherwise, see Selecting and Importing Tables to continue using the Adapter Configuration Wizard.

Selecting and Importing Tables

Figure 9-10 shows where you select the root database table for your operation. If you are using multiple related tables, then this is the highest-level table (or highest parent table) in the relationship tree.

Figure 9-10 The Adapter Configuration Wizard: Select Table



Selecting Import Tables launches a sub-wizard, which lets you search for and select multiple tables to import from the database. Removing a table removes (or undoes) any relationships on related tables that remain. If any underlying tables have changed when running this wizard in edit mode, you get a warning showing you what changes have occurred. To reconcile, import the tables again. If you click **Import Tables** and select multiple tables, then relationships between these tables are inferred based on the foreign key constraints. However if you launch **Import Tables** once for each table imported, then no relationships are inferred.

Note:

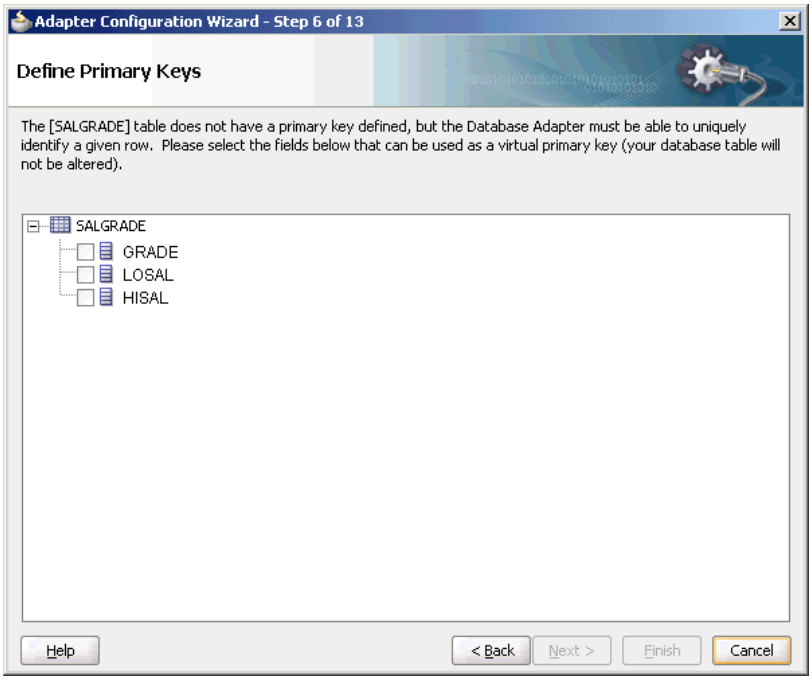
If you reimport a table, you lose any custom relationships you may have defined on that table and any custom `WHERE` clauses (if the table being imported was the root table).

See Defining Primary Keys to continue using the Adapter Configuration Wizard.

Defining Primary Keys

If any of the tables you have imported do not have primary keys defined on the database, you are prompted to provide a primary key for each one, as shown in Figure 9-11. You must specify a primary key for all imported tables. You can select multiple fields to specify a multipart primary key.

Figure 9-11 The Adapter Configuration Wizard: Define Primary Keys Page



The primary key that you specify here is recorded on the offline database table and is not persisted back to the database schema; the database schema is left untouched.

See Creating Relationships to continue using the Adapter Configuration Wizard.

Note:

Note that the Oracle Database Adapter only supports tables where there is a primary key defined. If primary key constraints have not been defined on a table explicitly, then you must provide one at design time while defining the Oracle Database Adapter by using the Adapter Configuration Wizard. If you do not provide a valid primary key, then the unique constraint is not guaranteed, and this could result in possible loss of messages at runtime. That is, rows with duplicate primary key values are likely to be lost. Also, you should ensure that you primary key is less than 100 bytes.

Note:

Oracle recommends that you use `varchar` instead of `char` for primary key columns, otherwise you must set the `weblogic-ra.xml` property `shouldTrimStrings` to `false`. The truncation of trailing spaces could cause the primary key to be read incorrectly, making it impossible to update read rows as processed.

Using ROWID as the Primary Key

Oracle Database Adapter ROWID support now allows you to choose the ROWID as the primary key for certain supported operations.

When you import a table with no primary key constraint, you are prompted to select the set of columns that can be used to uniquely identify a row (see Figure 9-12). For an

integration scenario this might require more knowledge about the target schema than is had, and can introduce errors. For example, picking a non-unique column here can lead to operations updating multiple unintended rows, or Selects returning the some rows twice and not others.

On an Oracle database you can choose to use the ROWID pseudo column as the primary key, which is less error prone. In some case, it is faster than a regular index as ROWID is the direct physical address.

A ROWID is a pseudo-column on Oracle tables. A pseudo-column is a column that cannot be updated directly by the user, is not a column created by the user, and does not appear in any column meta-data views or is returned from a select * query.

Use of ROWID has certain restrictions. In such cases, the option are grayed out on the user interface and unselectable.

ROWID can only be used for operations which either do not need to uniquely identify a row (Insert, Select) or which only need to identify rows it has already read in the same transaction (most polling operations). Because the physical address has no meaning external to the database and is not part of the row (and the value is stable within a transaction), it cannot be used with operations which must uniquely identify a row using the columns of the row itself (Merge, Delete, Select by primary key).

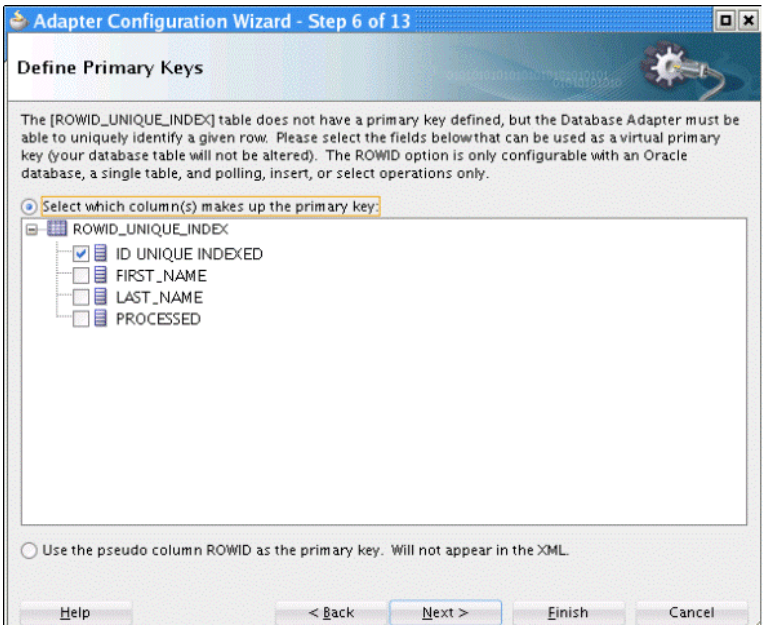
ROWID can only be used in conjunction with a single table, as relationships between tables require rows to have explicit primary keys.

ROWID support is not applicable to Pure SQL or Stored procedures.

Using Rowid on the Primary Key Page

When you import a table that does not have a primary key set, the wizard displays the primary key page. See Figure 9-12.

Figure 9-12 Defining Primary Key Using Rowid



From this page you can either:

- Select columns to create a primary key by clicking the radio button **Select which columns makes up the primary key**

- Select ROWID as the primary key by clicking the radio button **Use the pseudo column ROWID as the primary key**

Again, ROWID can only be used with inbound polling, insert, or select operations.

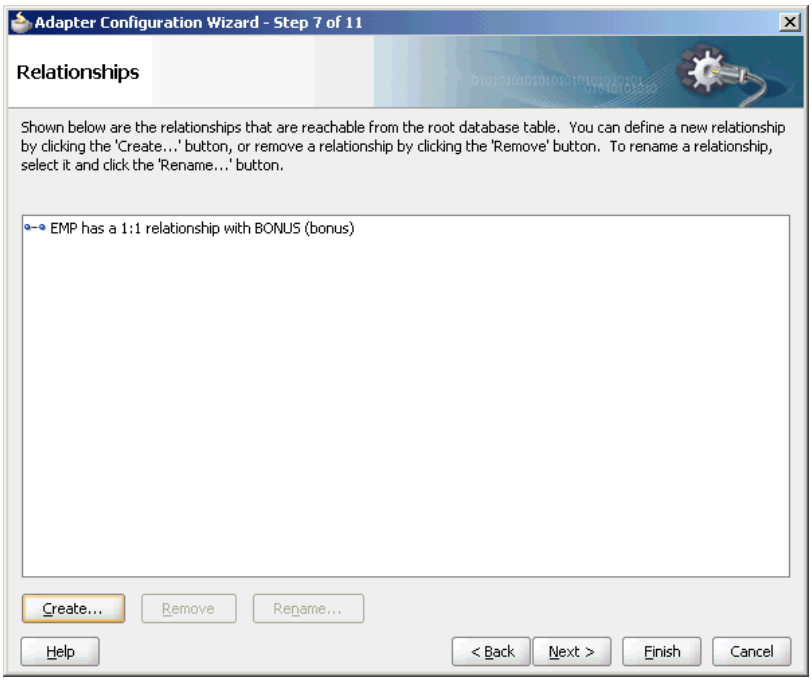
If you choose to select columns to make up the primary key, next to each column appear several possible hints, such as `NOT NULL`, `INDEXED`, `AUTO INCREMENT` and `UNIQUE INDEXED`.

In the last case, that column's checkbox is automatically selected by default, and you are asked to confirm.

Creating Relationships

Figure 9-13 shows the relationships defined on the root database table and any other related tables. You can click **Create Relationships…** to create a relationship between two tables, or click **Remove Relationship** to remove it. To rename a relationship, click **Rename Relationship**.

Figure 9-13 The Adapter Configuration Wizard: Relationships Page



Note the following regarding creating relationships:

- If foreign key constraints between tables exist on the database, then two relationships are created automatically when you import the tables, a one-to-one (1:1) from the source table (the table containing the foreign key constraints) to the target table, and a one-to-many (1:M) from the target table to the source table.
- As Figure 9-13 shows, you see only the relationships that are reachable from the root database table. If, after removing a relationship, other relationships are no longer reachable from the root table, then they are not shown in the Relationships window. Consider the following set of relationships:

  ```
  A --1:1--> B --1:1--> C --1:M--> D --1:1--> E --1:M--> F

  (1) (2) (3) (4) (5)
  ```

  If you remove relationship 3, then you see only:

```
A --1:1--> B
B --1:1--> C
```

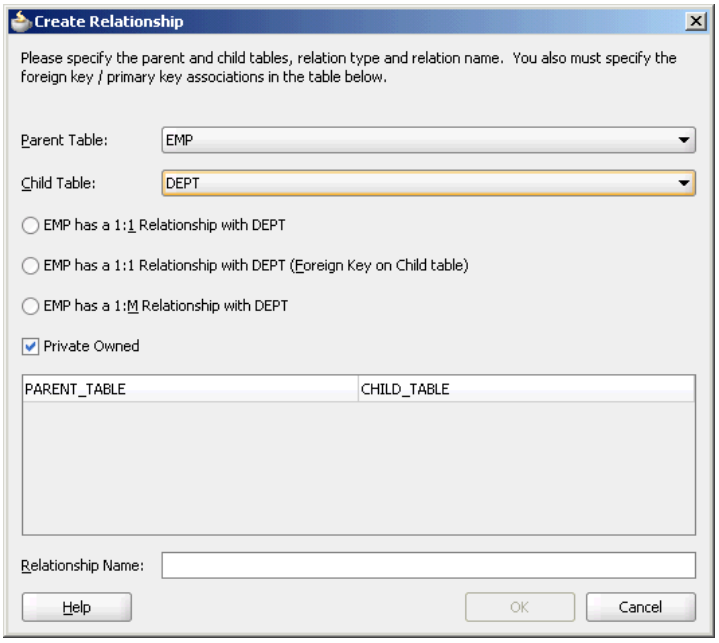
If you remove relationship 2, then you see only:

```
A --1:1--> B
```

If you remove relationship 1, you no longer see any relationships.

Figure 9-14 shows where you can create a relationship.

Figure 9-14 The Create Relationship Dialog



To create a relationship:

1. Select the parent and child tables.
2. Select the mapping type (one-to-many, one-to-one, or one-to-one with the foreign key on the child table).
3. Associate the foreign key fields to the primary key fields.
4. Optionally name the relationship (a default name is generated).

Note:

Only tables that are reachable from the root table can be selected as a parent.

What Happens When Relationships Are Created or Removed

When tables are initially imported into the Adapter Configuration Wizard, a TopLink direct-to-field mapping corresponding to each field in the database is created. Consider the schemas shown in Figure 9-15 and Figure 9-16:

Figure 9-15 EMPLOYEE Schema

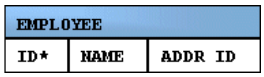


Figure 9-16 ADDRESS Schema

Immediately after importing these two tables, the following mappings in the `Employee` descriptor are created:

`Employee:`

- `id` (direct mapping to the `ID` field, for example, *151*)
- `name` (direct mapping to the `NAME` field, for example, *Stephen King*)
- `addrId` (direct mapping to the `ADDR_ID` field, for example, *345*)

When creating a relationship mapping, the direct-to-field mappings to the foreign key fields are removed and replaced with a single relationship (one-to-one, one-to-many) mapping. Therefore, after creating a one-to-one relationship between `Employee` and `Address` called `homeAddress`, the `Employee` descriptor appears, as shown in the following example:

`Employee:`

- `id`
- `name`
- `homeAddress` (one-to-one mapping to the `ADDRESS` table; this attribute now represents the entire `Address`s object.)

When a relationship is removed, the direct mappings for the foreign keys are restored.

Different Types of One-to-One Mappings

When relationships are auto created, the one-to-many relationship is from the table without the foreign key. However, you can declare this mapping, which is technically 1-many, as a 1-1. For that, choose 1-1 (foreign key on target).

When Foreign Keys Are Primary Keys

Not all tables imported are in the third normal form (3NF). In rare cases, you may have two or more tables which share the same primary key but no separate foreign key columns exist. It is recommended to create 1-1 (foreign key on target) relationships from the root table to all related tables. The reason is two fold. First, if you were to declare the primary key on the root as a foreign key (1-1, foreign key on source), then that mapping would be removed, so you would not see the primary key in the root record, only in the child record. Second, a foreign key can only point to a single table. Once you declare a column to be part of a foreign key, it is removed, so it cannot be used again in a new relationship. Creating a 1-1 (foreign key on source) on the root table not only makes the primary key column disappear but prevents you from joining the root table to the remaining tables.

Creating the Attribute Filter

Figure 9-17 shows the attribute filter that is created from the imported table definitions, including any relationships that you may have defined.

Figure 9-17 The Adapter Configuration Wizard: Attribute Filtering Page

If your object filter contains self-relationships (for example, the employee-to-employee manager relationship), then you see these as loops in the tree. These loops are not present in the XSD file. This is the descriptor object model, not the XSD file.

In this page, you select those columns that appear in the XML file, whether for input (`MERGE`, `INSERT`) or output (`SELECT`). Columns you are not interested in or which are to be read-only (should not be modified) can be deselected here.

See Defining a WHERE Clause to continue using the Adapter Configuration Wizard.

Defining a WHERE Clause

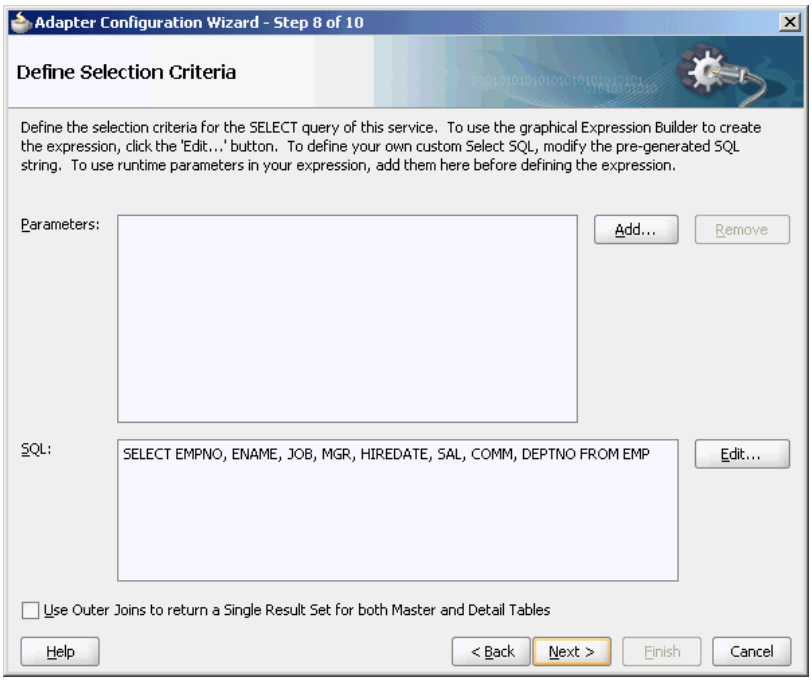
If your service contains a `SELECT` query (that is, inbound polling services, or outbound services that contain a `SELECT`), then you can customize the `WHERE` clause of the `SELECT` statement.

Note:

When using polling with `Sequencing Table/Update an External Sequencing Table`, ensure that the name of the table in the `SELECT` query matches the case of the data in the sequencing table.

Figure 9-18 shows where you define a `WHERE` clause for an outbound service.

Figure 9-18 The Adapter Configuration Wizard: Define Selection Criteria Page



Note:

The `WHERE` clause applies to `SELECT` operations only (that is, polling for new or changed records or performing a `SELECT` operation on a table). It does not apply to `INSERT`, `UPDATE`, and `DELETE` operations.

The most basic expression in a `WHERE` clause can be one of the following three cases, depending on what the right-hand side (RHS) is:

1. `EMP.ID = 123`

   In this case, the RHS is a literal value. This RHS is the **Literal** option shown in Figure 9-19.

2. `EMP.ADDR_ID = ADDR.ID`

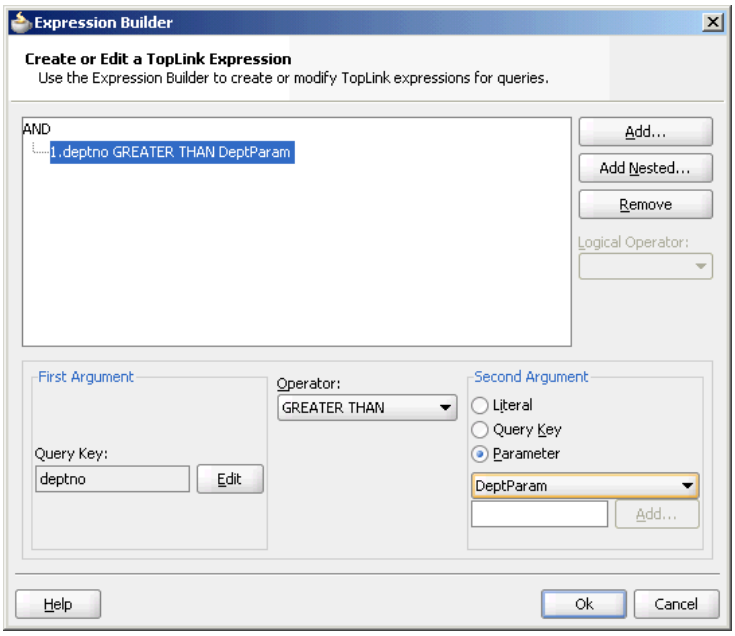
   In this case, the RHS is another database field. This RHS is the Query Key option shown in Figure 9-19.

3. `EMP.ID = ?`

   In this case, the RHS value must be specified at runtime. This is the Parameter option shown in Figure 9-19.

You can create the parameters that you require in the `WHERE` clause by clicking **Add** before you move on to build the `WHERE` clause. To build the `WHERE` clause, click **Edit...** to launch the Expression Builder, as shown in Figure 9-19.

Figure 9-19 Expression Builder



To model more complex `WHERE` clauses (sub selects and functions), and to add `ORDER BY` clauses, you can edit the SQL procedure manually and click **Next**. However, this creates maintenance overhead later on, due to hard-coded `SQL`, and you may lose platform independence.

You can also achieve `ORDER BY` clause by adding `shouldOrderRows` and `sequencingField` properties with the name of the column to the mapping.xml file.

```
  if ((this.sequencingField != null) && (shouldOrderRows()))
     {
        ReadAllQuery raPollingQuery = (ReadAllQuery)this.pollingQuery;
raPollingQuery.addOrdering(raPollingQuery.getExpressionBuilder().getField(this.sequen
cingField).ascending());
     }
```

You can change the columns listed in the `FROM` clause when the number of columns and the types of each remain unchanged. For more complex changes consider using the Execute Pure SQL option directly where you can type any `SQL`.

Return Single Result Set

You must select **Use Outer Joins to return a Single Result Set for both Master and Detail Tables** in the Define Selection Criteria page to use an advanced feature that influences how many total statements `TopLink` uses when querying against multiple related tables. The safest method is to use the default (1 per table), and this feature attempts 1 total, by outer joining all related tables into a single result set.

See Choosing an After-Read Strategy to continue using the Adapter Configuration Wizard.

Choosing an After-Read Strategy

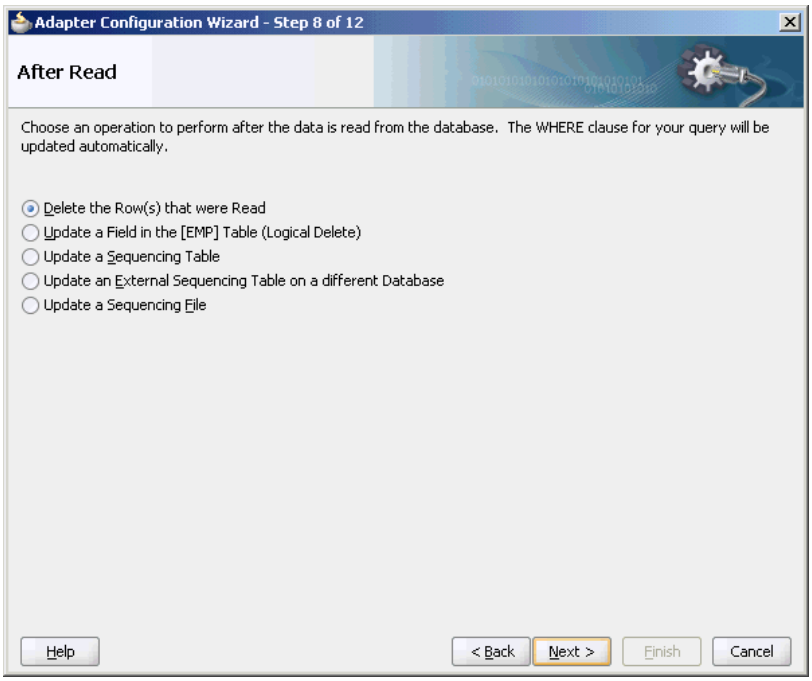
If you selected Perform an Operation on a Table, then you can skip ahead to the Specifying Advanced Options..

When configuring an inbound operation, you have the following options about what to do after a row or rows have been read:

- Delete the Rows That Were Read
- Update a Field in the Table (Logical Delete)
- Update a Sequencing Table
- Update an External Sequencing Table on a Different Database
- Update a Sequencing File

Figure 9-20 shows these options.

Figure 9-20 The Adapter Configuration Wizard: After Read Page



See Polling Strategies to continue using the Adapter Configuration Wizard.

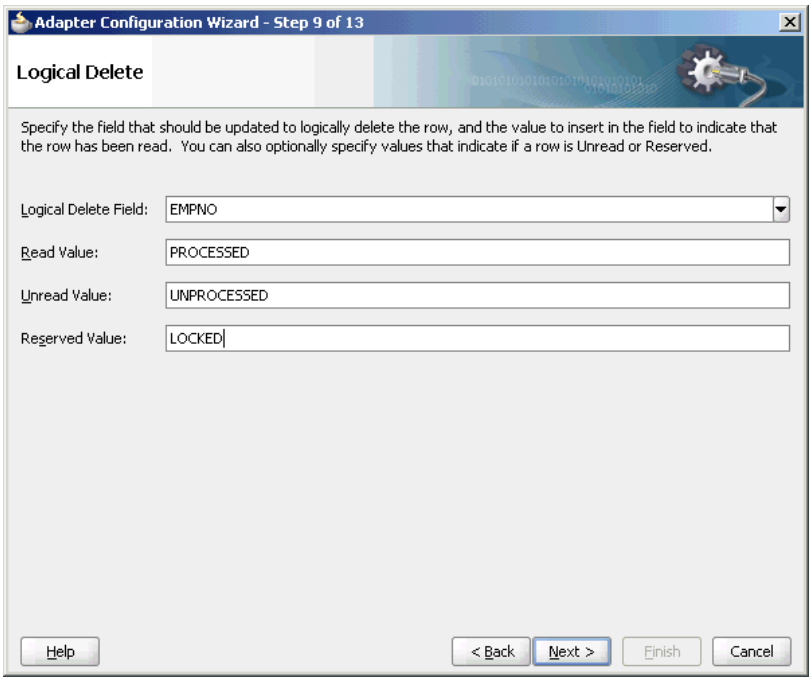
Delete the Rows That Were Read

With this option, the rows are deleted from the database after they have been read and processed by the adapter service.

Update a Field in the Table (Logical Delete)

With this option, you update a field in the root database table to indicate that the rows have been read. The `WHERE` clause of the query is updated automatically after you complete the configuration, as shown in Figure 9-21.

Figure 9-21 The Adapter Configuration Wizard: Logical Delete Page



When you use this approach, your database table appears, as shown in Figure 9-22.

Figure 9-22 Updating Fields in a Table

EMPLOYEE				
ID*	NAME	SALARY	...	STATUS
150	Dean Koontz	55000	...	PROCESSED
151	Stephen King	75000	...	
152	Patricia Cornwell	58000	...	UNPROCESSED
153	John Grisham	67500	...	PROCESSED
154	Michael Crichton	61250	...	LOCKED

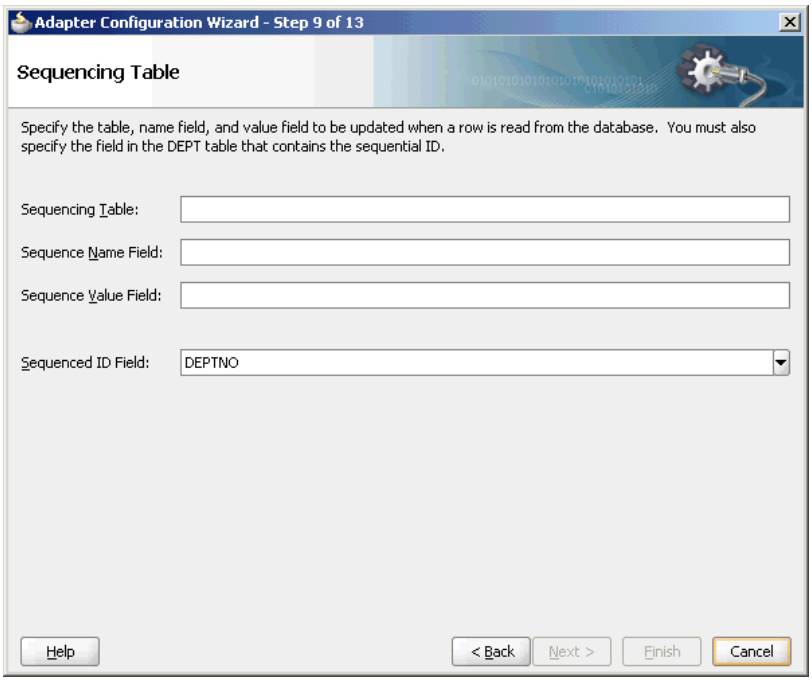
Note the following:

- Rows 150 and 153 have been previously read and processed.
- At the next polling event, row 152 is read and processed because it contains `UNPROCESSED` in the Status column. Because an explicit `Unread Value` was provided, row 151 is not read.
- Row 154 has been flagged as `LOCKED` and is not read. You can use this reserved value if your table is used by other processes.

Update a Sequencing Table

With this option, you are keeping track of the last-read rows in a separate sequence table. Figure 9-23 shows the information you provide. The `WHERE` clause of your query is updated automatically after you complete the configuration.

Figure 9-23 The Adapter Configuration Wizard: Sequencing Table Page



When you use these settings, your sequence table appears, as shown in Figure 9-24.

Figure 9-24 Updating a Sequence Table

MY_SEQUENCE	
SEQ_NAME	SEQ_VALUE
EMPLOYEE	154

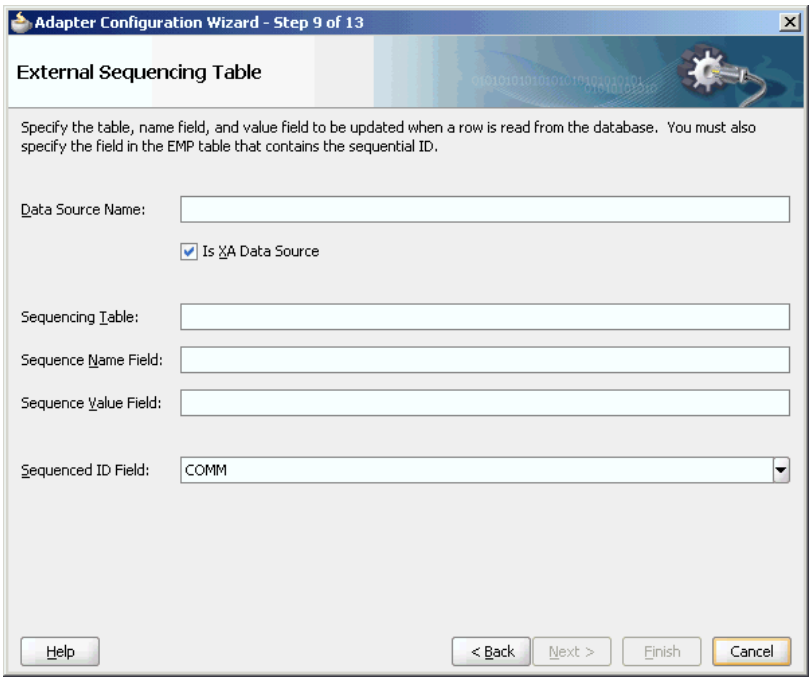
Whenever a row is read, this table is updated with the ID that was just read. Then, when the next polling event occurs, it searches for rows that have an ID greater than the last-read ID (154).

Typical columns used are `event_id`, `transaction_id`, `scn` (system change number), `id`, or `last_updated`. These columns typically have (monotonically) increasing values, populated from a sequence number or `sysdate`.

Update an External Sequencing Table on a Different Database

Choose this operation to employ the sequencing table: last updated strategy. Figure 9-25 shows the Adapter Configuration Wizard - External Sequencing Table page in which you specify the details required to perform this operation.

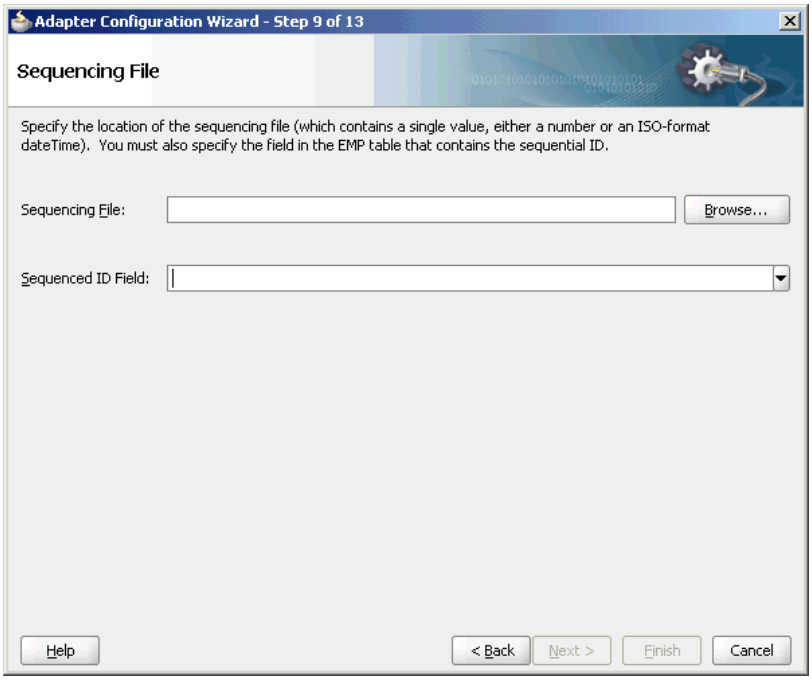
Figure 9-25 The Adapter Configuration Wizard - External Sequencing Table page



Update a Sequencing File

Use this option to update a sequencing file. Figure 9-26 shows the Adapter Configuration Wizard - Update a Sequencing File page where you specify the details for performing this operation.

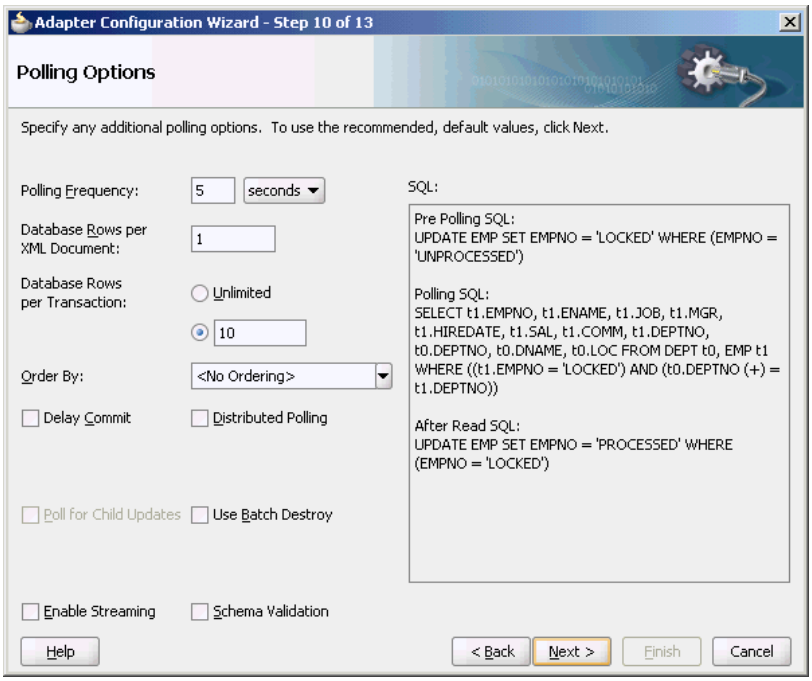
Figure 9-26 Adapter Configuration Wizard - Update a Sequencing File Page



Specifying Polling Options

You can specify additional polling options, if any, in this page. Figure 9-27 shows the Adapter Configuration Wizard - Polling Options page.

Figure 9-27 Specifying Polling Options



In this page, you specify details about how to poll the database table for new rows or events.

From the Polling Frequency list, select how frequently to poll for new records or events.

In the **Database Rows per XML Document** field, specify the number of rows per XML document when sending events to Oracle BPEL PM or Mediator. This is the batch setting between the database adapter and its consumer: Oracle BPEL PM or Mediator.

In the **Database Rows per Transaction** field, select **Unlimited** or enter a value to indicate the number of table rows to process during a single transaction.

When polling the database for events, you can order the returned rows by the selected column by using the **Order By** list. The best practice is to choose **<No Ordering>**, as message ordering regardless is not guaranteed without extra configuration.

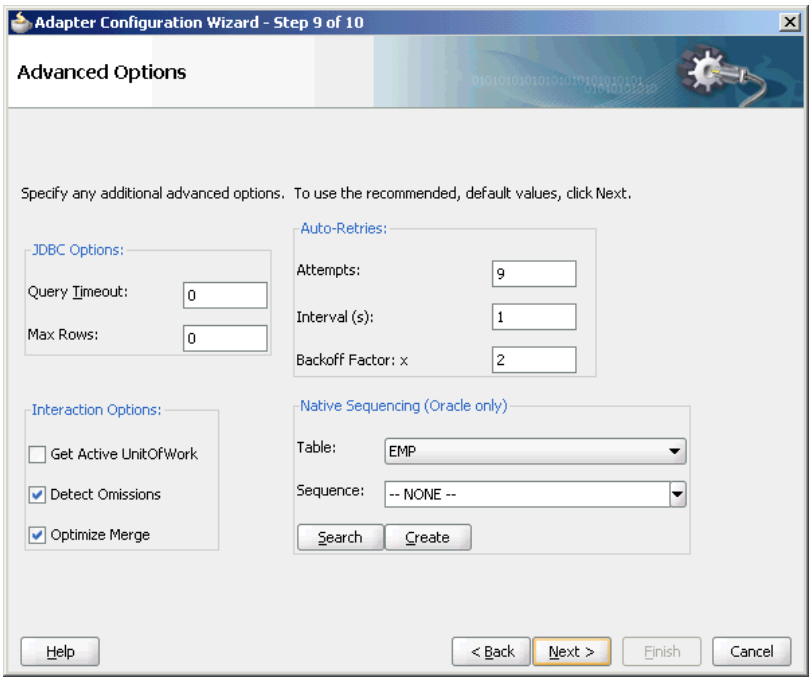
In the SQL field, if the SQL syntax is incorrect, then a message is displayed in red.

For more information about specifying polling options, click **Help** in the Polling Options page or press F1.

Specifying Advanced Options

You can specify advanced options, if any. Figure 9-28 shows the Adapter Configuration Wizard - Advanced Options page. In this page, you can specify advanced JDBC and DBAdapter options, configure retries, and configure native sequencing.

Figure 9-28 Specifying Advanced Options



You must specify JDBC options in the **JDBC Options** section. Set low-level JDBC options on calls to the database. The operation you selected determines which options may appear here.

In the **Auto-Retries** section, specify the value for auto-retry incase of time out. In case of a connection related fault, the Invoke activity can be automatically retried a limited number of times. You can specify the following values in the fields in this section:

- To retry indefinitely, type `unlimited` in the **Attempts** field.
- **Interval** is the delay between retries.
- **Backoff Factor: x** allows you to wait for increasing periods of time between retries. 9 attempts with a starting interval of 1 and a back off of 2 leads to retries after 1, 2, 4, 8, 16, 32, 64, 128, and 256 (2^8) seconds.

In the **Interaction Options**, specify the interaction options, as follows:

- **GetActiveUnitOfWork**. Set `GetActiveUnitOfWork` to true when making multiple incremental changes to the same record across multiple invokes in the same SOA instance.`GetActiveUnitOfWork` ensures that all Database Adapter invokes which participate by setting the option to true, and which are within the same JTA transaction, and to the same eis/DB/ instance, use the same EclipseLink session for all operations. For EclipseLink-based operations (excluding stored procedures and pure SQL) all writes are deferred until JTA before Completion.

  This means if you insert/merge the same object in two invokes, it is written once. Since EclipseLink-based writes are deferred, a select all may not conform to previous invokes which did writes. Selects by primary key conform, however. As writes happen inside the JTA callbacks, there is no way to handle exceptions which occur at that time and BPEL's global transaction will unexpectedly fail.`GetActiveUnitOfWork` is frequently used to guarantee that operations on two invokes used the same physical SQL connection, since a connection is pinned to the EclipseLink session for the duration of a transaction. However, most

application server data sources provide the same guarantee however. WebLogic also has a similar Pinned-To-Thread property and GridLink has XA affinity, which ensures that all writes in an XA transaction happen on the same node in a RAC cluster. Setting this does not resolve lock contention between different SOA instances.

If you have multiple operations on related data (such as parent-child,) try to have them occur in the same SOA instance or even invoke. Nor will this ensure connection reuse if the two invokes are across transaction boundaries. Make sure in BPEL if the second DbAdapter invoke is within a sub-process, that BPEL property `bpel.config.transaction` is set to `required` on the callee composite level.

- **Detect Omissions** allows the `MERGE` and `INSERT` operations to ignore empty or missing XML elements in the input payload. For a `MERGE` operation, this prevents valid but unspecified values from being overwritten with NULL. For `INSERT` operations, they are omitted from the `INSERT` statement, allowing default values to take effect.
- **Optimize Merge** should always be set to true, as it is a general enhancement to `MERGE` performance (using an in query for the primary key existence check).

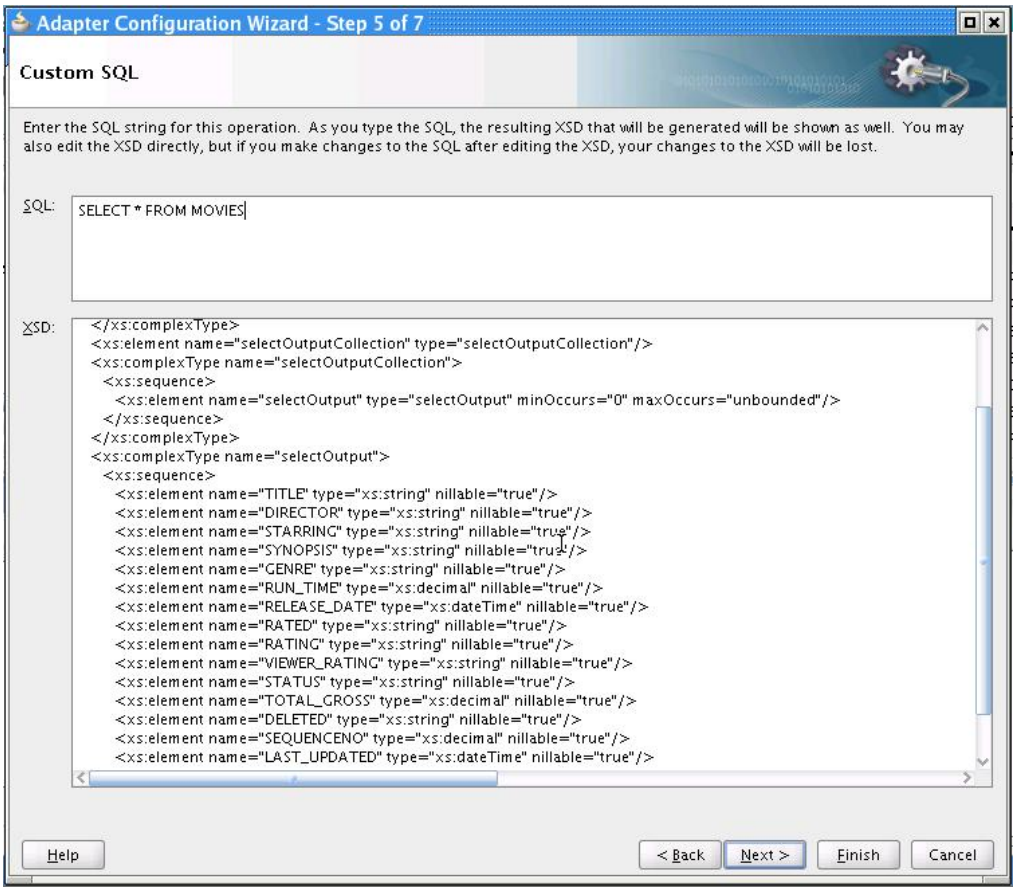
Native Sequencing (Oracle only) allows you to specify that the primary key are assigned from a sequence on any insert. Click **Search** and then select a sequence from the **Sequence** list, or type the name and click **Create**.

For more information about specifying advanced options, click **Help** in the Advanced Options page or press F1.

Entering the SQL String for the Pure SQL Operation

You can enter a SQL string for performing the Execute Pure SQL operation in the Custom SQL page. Figure 9-29 shows the Adapter Configuration Wizard - Custom SQL page.

Figure 9-29 Entering a SQL String



In the **SQL** field, enter a custom SQL string. An XSD schema of your SQL input is automatically created in the **XSD** field.

The XSD field displays the XSD schema of the custom SQL string you entered. You can directly edit the resulting XSD. However, if you make subsequent changes to the SQL string, then your XSD changes are lost.

For more information about entering a SQL string, click **Help** in the Custom SQL page or press F1.

Oracle Database Adapter Features

This section discusses the Oracle Database Adapter features.

It includes the following topics:

- Transaction Support
- Pure SQL - XML Type Support
- Row Set Support Using a Strongly or Weakly Typed XSD
- Proxy Authentication Support
- Streaming Large Payload
- Schema Validation

- High Availability
- Scalability
- Performance Tuning
- detectOmissions Feature
- OutputCompletedXml Feature
- QueryTimeout for Inbound and Outbound Transactions
- Doing Synchronous Post to BPEL (Allow In-Order Delivery)

Transaction Support

The Oracle Database Adapter enables transaction support, which, along with the inherent data processing, ensures that each modification has a clearly defined outcome, resulting in either success or failure, thus preventing potential corruption of data, executes independently from other changes, and, once completed, leaves underlying data in the same state until another transaction takes place.

There are two types of transaction support, XA Transaction support and Local Transaction support. XA transaction support allows a transaction to be managed by a transaction manager external to a resource adapter, whereas, a local transaction support allows an application server to manage resources that are local to the resource adapter.

To ensure two Oracle Database Adapter invokes commit or rollback as a unit, you must perform the following:

- Both Oracle Database Adapter invokes must be configured to participate in global transactions.
- Both Oracle Database Adapter invokes must participate in the same global transaction.
- The failure of either invoke must cause the global transaction to roll back.

Note:

You must use a non-XA driver with the `SOALocalTxDataSource` parameter. Switching to an XA driver breaks product functionality.

Configuring Oracle Database Adapter for Global Transaction Participation

In the deployment descriptor (`weblogic-ra.xml` file), you must set the `xADataSourceName` parameter. Additionally, the referenced DataSource must be configured for transaction participation by creating a data source in Oracle WebLogic Server Console.

You must create a data source and choose a XA data sources from the list.

Note:

True Database XA is only certified on Oracle 10.2.0.4 or 11.1.0.7. For earlier versions, you are safer picking a non-XA data source implementation and selecting **Emulated Two-phase commit** on the next page.

For information about the recommended setting for non-XA and XA data sources used by Oracle JCA Adapters, see Recommended Setting for Data Sources Used by Oracle JCA Adapters.

You cannot edit the `data-sources.xml` file in the Oracle WebLogic Server. You must create a data source by using the Oracle WebLogic Server Administration Console, as mentioned in Creating a Data Source.

Both Invokes in Same Global Transaction

Once both the Oracle Database Adapter invokes participate in global transactions, to commit or rollback as a unit, they must be participating in the same global transaction. In BPEL, this requires the understanding of where the transaction boundaries are, at what points does a checkpoint have to write to the dehydration store, commit the current global transaction, and start a new one.

The transaction boundaries in a BPEL process occur either before a Receive activity or Wait activity, or before an `onMessage` or Pick activity. This can also occur when invoking a synchronous child BPEL process, unless the `bpel.config.transaction` property is set on the partnerlink, as shown in the following code sample.

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

Otherwise, the parent process is broken into two transactions and the child process runs in its own transaction.

Failure Must Cause Rollback

Finally, even if both Oracle Database Adapter invokes participate in the same global transaction, the failure of either invoke may not cause the global transaction to rollback.

The only cases where a failure can actually cause a global rollback are:

- A Oracle Database Adapter operation that inserts/updates multiple tables as part of one invoke fails after having succeeded in some writes but not others. In this case, the Oracle Database Adapter marks the global transaction as rollback only, because the invoke operation was not atomic and a commit could cause data corruption.
- The invoke retries multiple times in a database down scenario, until the global transaction times out and is rolled back.
- An explicit `bpelx:rollback` fault is thrown from within the BPEL process.

Using the Same Sessions for Both Invokes

You must set the `GetActiveUnitOfWork` JCA parameter to true to enable using the same sessions or connections for both the Oracle Database Adapter invokes.

`GetActiveUnitOfWork` is an advanced JCA property you can set on any `DBInteractionSpec`. It causes the invoke to register itself with the two-phase

commit callbacks, and all writes to the database are performed as part of the two-phase commit. By setting this property on any failure, the transaction is automatically rolled back, as there is no way to *handle* a fault at this late stage. Similarly, the same underlying TopLink session is used for both invokes, meaning if you merge the same object twice, it is inserted/updated once. All merge invokes that set `GetActiveUnitOfWork` as true are cumulative.

Transaction/XA Support

To make two Oracle Database Adapter invokes commit or roll back as a unit requires the following: both Oracle Database Adapter invokes must be configured to participate in global transactions, both invokes must participate in the same global transaction, and the failure of either invoke must cause the global transaction to rollback.

Configuring an Oracle Database Adapter for Global Transaction Participation

In the deployment descriptor (weblogic-ra.xml), you must set `xADataSourceName`. The matching data source entry must be configured for global transaction participation.

True XA: Two-Phase (XA) Versus One-Phase (Emulated) Commit

XA is a two-phase commit protocol, which is more robust than a one-phase commit or emulated protocol. The difference is that with a one-phase protocol, you may very rarely still see message loss or other rollback/commit inconsistency, on the order of one per one thousand generally.

Oracle RAC Configuration

For more information about Oracle RAC configuration, see the *Real Application Clusters* in *Oracle Database High Availability Overview Guide*.

True XA Configuration with Third-Party Drivers

When configuring true XA for third-party drivers (that is, Microsoft SQL Server 2008, IBM DB2), see if the driver jars contain a class that implements `javax.sql.XADataSource`.

For data direct drivers, the naming happens to be `com.oracle.ias.jdbcx.db2.DB2DataSource`, or `com.oracle.ias.jdbcx.sqlserver.SQLServerDataSource`.

Failure Must Cause Rollback

Finally, even if both invokes participate in the same global transaction, the failure of either invoke may not cause the global transaction to roll back.

The only cases where a failure can actually cause a global roll back are:

- An Oracle Database Adapter operation that inserts/updates multiple tables as part of one invoke fails after having succeeded in some writes but not others. In this case, the adapter marks the global transaction rollback only, as the invoke operation was not atomic and a commit could cause data corruption.
- The invoke retries multiple times in a database down scenario, until the global transaction times out and is rolled back.
- An explicit `bpelx:rollback` fault is thrown from within the BPEL process. `GetActiveUnitOfWork="true"` in WSDL.

Pure SQL - XML Type Support

Pure SQL Adapter is an option in the Oracle Database Adapter Wizard that allows you to type the SQL string directly and have an XSD/Web service generated automatically. The database tables are introspected dynamically in the Adapter Configuration Wizard to test the SQL and populate the XSD file better (that is, with valid return types.)

The Pure SQL support allows the Oracle Database Adapter to deal with tables/views as entities and for dealing directly with SQL. You can use Pure SQL:

- for simple data projection style report queries
- in cases where the result set is not table oriented, such as select count(*)
- to perform an update or delete all
- when working with `XMLType` columns and `xquery`
- when using complex SQL, which are not modeled in the Adapter Configuration Wizard expression builder

You can use the Pure SQL Adapter with Oracle `XMLTypes`. It is a natural fit for inserting XML into `XMLType` tables and columns, and retrieving XML using `xquery` selects. Pure SQL is a natural fit for the Oracle Database Adapter that provides a relational-xml mapping that parallels XML DB(XDB) support. So, when using XDB the adapter should be as lightweight and transparent as possible, to let you focus on XDB and `XQuery`.

If your data is in XML (unstructured/semi-structured) format, and you have no relational schema at all that you can map your data to, then you could use XDB. The conventional Oracle Database Adapter allows you to import an existing relational schema as an XML schema to be used with Web services. XDBs XML shredding algorithm can generate a relational schema from an existing XML schema for persistent storage.

Note:

Use of schema bound `XMLTypes` requires the `oci` driver, which is not certified in the 11*g* release. Therefore, you must use non-schema bound `XMLTypes` at runtime, though you can use schema bound `XMLTypes` at design time to import a representative XSD.

For more information, see:

- Selecting the Operation Type

Row Set Support Using a Strongly or Weakly Typed XSD

Currently a `REF CURSOR` by nature can support any arbitrary result set, so the XSD generated at design time allows this and looks like the XSD as shown in the following example.

Note:

Oracle Database stored procedures return result sets that are referred to as `RefCursors`, whereas third-party databases result sets that are returned are referred to as `RowSets`.

Example - Weakly Typed XSD

```
<refCursorOutputParam>
    <Row>
        <Column name="DEPTNO" sqltype="NUMBER">20</Column>
        ...
    </Row>
</refCursorOutputParam>
```

However the XML output from this is hard to use. It is very difficult to write an Xpath expression or XSL based on a weakly typed XSD and column names as attribute values instead of element names.

Although a row set can represent any result set, it is possible to assume for some procedures that it has the same structure each time, and hence can be described with a strongly typed XSD. A strongly typed XSD is almost a necessity to transform the result set to another XSD later on. A strongly typed XSD looks like the XSD that is shown in the following example.

Example - Strongly Typed XSD

```
<refCursorOutputParam>
    <dept>
        <deptno>20</deptno>
        ...
    </dept>
</refCursorOutputParam>
```

You can use the Adapter Configuration Wizard to create a strongly typed XSD for a row set returned by a stored procedure or function `REF CURSOR` variable. An Oracle Database function is a special stored procedure that always has one out variable, and can be inlined - for example, inside select statements - and so traditionally does not do updates.

Using this feature, you can select a stored procedure (or stored function), enter its arguments, and perform a test execution to retrieve an actual row set. The Adapter Configuration Wizard then introspects the returned row set and generates a strongly typed XSD. You can enter arguments easily through the wizard. For example, you can enter numbers and strings directly, dates as literals (2009/11/11), and you can even enter structs such as `MYOBJ('a', 'b')`.

Note:

Functions are not supported for IBM DB2 UDB. Only SQL stored procedures are supported.

The Adapter Configuration Wizard row set support using a strongly typed XSD and has the following restrictions:

- Oracle Database PL/SQL `record` or `boolean` types are not supported.

- Oracle Database PL/SQL `varray` is not supported.
- Oracle Database PL/SQL `%rowtype` is not supported.
- Oracle Database PL/SQL `table` types are not supported.
- Oracle Database PL/SQL procedures with `IN` only `REF CURSOR` parameters are not supported.

  For an Oracle Database PL/SQL procedure with `REF CURSOR` as an `IN/OUT` parameter, the Adapter Configuration Wizard ignores the `IN` and generates the strongly typed XSD based on the `OUT` parameter.
- Referencing an element in the XSD using `ref` is not supported.
- SQL Server 2008 table valued functions and CLR functions are not supported.

The Oracle Database Adapter supports strongly typed XSD for the following third-party databases:

- Microsoft SQL Server 2005
- Microsoft SQL Server 2008
- IBM DB2 UDB 9.7

The Oracle Database Adapter does not support strongly typed XSD for the following third-party databases:

- IBM DB2 AS/400
- MySQL
- Informix Dynamic Server
- Sybase 15.0.2

For more information, see:

- Stored Procedure and Function Support
- Row Set Support Using a Strongly Typed XSD
- Row Set Support Using a Weakly Typed XSD

Proxy Authentication Support

You can connect to your Oracle data store by using Proxy Authentication. On a per-invoke basis, you can set a combination of the following new header properties:

- `jca.db.ProxyUserName`: to use the `OracleConnection.PROXYTYPE_USER_PASSWORD` proxy type, set this property to the proxy user name as a `java.lang.String`.
- `jca.db.ProxyPassword`: to use the `OracleConnection.PROXYTYPE_USER_PASSWORD` proxy type, set this property to the proxy user password as a `java.lang.String`.
- `jca.db.ProxyCertificate`: to use the `OracleConnection.PROXYTYPE_CERTIFICATE` proxy type, set this property to a `base64Binary` encoded `byte[]` array containing a valid certificate.

This is a more encrypted way of passing the credentials of the user, who is to be proxied, to the database. The certificate contains the distinguished name encoded in it. One way of generating the certificate is by creating a wallet and then decoding the wallet to get the certificate. The wallet can be created using `runutl mkwallet`. It is then necessary to authenticate using the generated certificate.

- `jca.db.ProxyDistinguishedName`: to use the `OracleConnection.PROXYTYPE_DISTINGUISHED_NAME` proxy type, set this property to the proxy distinguished name as a `java.lang.String`.

  This is a global name in lieu of the password of the user being proxied for.

- `jca.db.ProxyRoles`: regardless of what proxy type you use, you can optionally set this property to define the roles associated with the proxy user as a `String[]` array where each `java.lang.String` corresponds to a role name.

- `jca.db.ProxyIsThickDriver`: if you are using the OCI driver, set this property to a value of `true` to accommodate differences in the JDBC-level API between the thick and thin drivers.

To run the invoke, a proxy connection is obtained from the data source.

For more information, see, *Proxy Authentication* in *Oracle Database JDBC Developer's Guide* .

Streaming Large Payload

To enable support to stream payload, you must select the Enable Streaming check box while specifying polling options, as shown in Figure 9-27. When you enable this feature, the payload is streamed to a database instead of getting manipulated in SOA runtime as in a memory DOM. You use this feature while handling large payloads. When you select the Enable Streaming check box, a corresponding Boolean property `StreamPayload` is appended to the ActivationSpec properties defined in the respective `.jca` file.

Schema Validation

The `SchemaValidation` [false/true] property is a new activation specification property that has been added, and this can be configured in a `.jca` file. When set to true, all XML files produced by the polling Oracle Database Adapter (for Receive activities) is validated against the XSD file. On failure, the XML record is rejected but still marked as processed by the Oracle Database Adapter.

Databases provide structured storage and the XSD file is generated by the Oracle Database Adapter Wizard itself. However, if you edit the auto generated XSD and add your own restrictions, you may want to start validation. For instance, if you import a VARCHAR(50) field, the auto-generated XSD has the max-length 50 restriction. However, if your BPEL process for some reason can only handle values of fixed length 22, it may want to validate the XML file.

High Availability

The Oracle Database Adapter supports high availability in an active-active setup. In an active-active setup, distributed polling techniques can be used for inbound Database Adapters to ensure that the same data is not retrieved more than once. For more information, see Distributed Polling First Best Practice: SELECT FOR UPDATE (SKIP LOCKED). Similar to other adapters, an Oracle Database Adapter can also be configured for singleton behavior within an active-passive setup. This allows a high

performance multithreaded inbound Oracle Database Adapter instance running in an active-passive setup, to follow a fan out pattern and invoke multiple composite instances across a cluster. The Oracle Database Adapter also supports the high availability feature when there is a database failure or restart. The DB adapter picks up again without any message loss.

Scalability

The following sections describe best practice for multiple Oracle Database Adapter process instances deployed to multiple Oracle BPEL PM or Mediator nodes, including:

- Distributed Polling First Best Practice: SELECT FOR UPDATE (SKIP LOCKED)
- Distributed Polling Second Best Practice: Tuning on a Single Node First

Distributed Polling First Best Practice: SELECT FOR UPDATE (SKIP LOCKED)

The first best practice for multiple Oracle Database Adapter process instances deployed to multiple Oracle BPEL PM or Mediator nodes is to use the Adapter Configuration Wizard to set both the **Distributed Polling** check box in the Adapter Configuration Wizard and to set `MaxTransactionSize`.

Increase concurrency by setting the `adapter _db.JCA` property `NumberOfThreads`.

On an Oracle database, doing this automatically uses the syntax `SELECT FOR UPDATE SKIP LOCKED`. With a Microsoft SQLServer database the syntax is `WITH (UPDLOCK,READPAST)`

With `SELECT FOR UPDATE SKIP LOCKED`, concurrent threads each try to select and lock the available rows, but the locks are only obtained on fetch. If an about to be fetched row is locked, the next unlocked row are locked and fetched instead. If many threads all execute the same polling query at the same time, they should all relatively quickly obtain a disjoint subset of unprocessed rows.

SKIP LOCKED in Depth

What is unique about skip locks is that each thread 'skips' the locks that other threads have already obtained, enabling each thread to get its own set of locked rows. In other locking strategies, the Select either encounters the first lock and waits or fails immediately. This is what allows the Database Adapter to scale.

That is a broad description of how SKIP LOCKED works, but it is also important to consider how SKIP LOCK works in depth and with respect to the settings of the following properties and the interactions and dependencies on them:

- NumberOfThreads- The number of concurrent threads.
- PollingInterval-The interval in seconds at which the Oracle Database Adapter executes the polling statement against the Oracle database.
- MaxTransactionSize- The number rows to be fetched fromDB to the Database Adapter in one transaction.
- MaxRaiseSize-The maximum number rows sent from the Database Adapter to BPEL as one message.
- RowsPerPollingInterval- A limit on the number of records which can be processed in one polling interval.

When a Database Adapter `.jca` file is deployed, the polling threads are created, based on the `NumberOfThreads` property. These threads poll independently of each other, starting a transaction and issuing a `SELECT FOR UPDATE SKIP LOCK`.

A database cursor is then returned for all available rows, and if there are no records matching the `SELECT`, the thread releases the transaction and sleeps for the duration specified by the `PollingInterval` property.

Once records appear in the database that match the polling `SELECT` statement, each thread wakes up after sleeping, starts a transaction, and issues the `SELECT FOR UPDATE SKIP LOCK`.

At this point, a database cursor is returned but this time there are rows that match the `SELECT` criteria. Each thread now issues a FETCH for a number of rows defined by the `MaxTransactionSize` property.

It is the `FETCH` that locks the rows in the database (`SKIP LOCKED`) preventing any other `FETCH` from retrieving those rows. This enables each polling thread to concentrate only on `SELECT`, `FETCH`, and process without concern for duplicate processing.

Now that each thread has its own set of rows to work with, the threads loop over the fetched rows and group them based on the `MaxRaiseSize` property.

Each grouping is delivered to the configured destination and once all rows have been delivered successfully, the transaction is committed.

Each thread then compares how many rows have been delivered to the value specified by the `RowsPerPollingInterval` property. If the rows delivered are equal to or greater than the RowsPerPollingInterval property value, the thread sleeps. If the number of rows delivered is less than RowsPerPollingInterval, the thread repeats the whole process over again.

Calculating `PollingInterval` and `MaxTransactionSize` with respect to Skip Locking is discussed in Configuring PollingInterval_ MaxTransactionSize_ and ActivationInstances in Depth

On a Non-Oracle Database

On a non-Oracle database, `SELECT FOR UPDATE` safely ensures that the same row cannot be processed multiple times, however you might obtain less scalability than you would otherwise. You should consider either using additionally a partition field or the second best practice, which is essentially multi-threading on a single node with fan-out (see Distributed Polling Second Best Practice: Tuning on a Single Node First).

Note:

A distributed approach ensures that multiple activation instances do not process the same rows.

When configuring this best practice, also consider the following:

- Configuring PollingInterval_ MaxTransactionSize_ and ActivationInstances in Depth
- Partition Field
- activationInstances

- Indexing and Null Values
- Disabling Skip Locking
- MarkReservedValue and Skip Locking
- SequencingPollingStrategy (Last Read or Last Updated)

Configuring PollingInterval, MaxTransactionSize, and ActivationInstances in Depth

In a distributed scenario, each polling instance tries to balance the load by not attempting to process all unprocessed rows by itself. Thus, at a time, an instance only fetches at most `MaxTransactionSize` rows.

When using skip locking, if full `MaxTransactionSize` rows are fetched, the next `MaxTransactionSize` rows can be immediately fetched continuously.

This is because concurrent threads do no block each other when using skip locking, so there is no danger of one instance fetching all the rows.

However, with skip locking disabled, all threads tries to lock the same rows, and only one succeeds. Consequently, once this thread has processed `MaxTransactionSize` rows, it pauses until the next polling interval, to allow other threads to also lock and process rows.

Hence, the maximum throughput with distributed polling enabled but using SkipLocking disabled is:

```
NumberOfThreads x MaxTransactionSize/PollingInterval
```

Note:

Although you might want to increase `MaxTransactionSize`, if you increase it to a value that is too high, you might start to see transaction timeouts. Table 9-2 lists safe values for `MaxTransactionSize`.

For load balancing purposes, it is dangerous to set the `MaxTransactionSize` too low in a distributed environment with skip locking disabled (where MaxTransactionSize becomes a speed limit). It is best to set the `MaxTransactionSize` close to the per CPU throughput of the entire business process. This way, load balancing occurs only when you require it.

Table 9-2 MaxTransactionSize and MaxRaiseSize Values

MaxTransactionSize	MaxRaiseSize	Description
10	1	When using sequential routing. For 10 rows you have 10 individual instances and 10 XML records passing through SOA.
100	-	When using parallel routing.
>= 100	`MaxTransactionSize`	When using the adapter to stream rows through as fast as possible.

For load balancing purposes, it is dangerous to set the `MaxTransactionSize` too low in a distributed environment (where it becomes a speed limit). It is best to set the

MaxTransactionSize close to the per CPU throughput of the entire business process. This way, load balancing occurs only when you require it.

If distributed polling is not set, the Adapter tries to process all unprocessed rows in a single polling interval.

Partition Field

In a distributed scenario there are polling instances on multiple servers; however, per server there can be multiple threads configured. You can configure these activation instances to cooperate somewhat by processing separate rows, possibly improving scaling.

To so, simply add the property PartitionField to your db.jca file:

```
<property name="PartitionField" value="ID"/>
```

If you set activationInstances to 2, then activation instances 1 and 2 (or 0 and 1) would respectively execute:

```
SELECT ... WHERE ... AND MOD (ID, 2) = 0 FOR UPDATE SKIP LOCKED
```

and

```
SELECT ... WHERE ... AND MOD (ID, 2) = 1 FOR UPDATE SKIP LOCKED
```

Activation instance 0 still conflicts with other activation instances with this ID on other servers, but at least it does not conflict with other activation instances with ID 1.

Ensure that the partition field is numeric and that applying mod evenly distribute the rows (that is, in this case make sure all the IDs are not either even or odd).

On Oracle Database, you can set the partition field to be rowid by setting db.jca file property PartitionField as follows:

```
 <property name="PartitionField" value="rowid"/>
```

Then the SQL is in fact converted to:

```
SELECT ... WHERE ... AND MOD (dbms_rowid.rowid_row_number(rowid), 2) = [0/1] FOR
UPDATE SKIP LOCKED
```

Because Oracle Database skip locking provides scalability, setting a partition field is not recommended. There is a cost of increased database CPU usage with more complex SQL.

activationInstances

T

The adapter framework level property activationInstances (configured in the composite.xml) is interchangeable with NumberOfThreads for distributed scenarios.

Setting activationInstances to 5 and NumberOfThreads to 5 is equal to setting one to 25 and the other to 1. As the extra work instances are created outside of the Database Adapter, they do not cooperate in any way. Hence, in a multi-threaded single node scenario, always configure NumberOfThreads only. Without database level concurrency control through enabling distributed polling, duplicates are read.

Note:

In a distributed cluster scenario configuring `NumberOfThreads` or `activationInstances` has the same effect. For a non distributed scenario, you must use `NumberOfThreads`. Hence it is safe to always use `NumberOfThreads` and disregard `activationInstances`.

For more information, see Singleton (Active/Passive) Inbound Endpoint Lifecycle Support Within Adapters.

Indexing and Null Values

Try to index (and/or add explicit constraints on the database for) the primary and all foreign keys to joined tables. If you are using Logical delet polling, try to index the status column. Try to configure a non-null `MarkUnreadValue` and `MarkReadValue`.

For optimal performance all operations (excluding INSERT) on the outbound Database Adapter, you should create an index in the database on the column that is selected as the primary key for the Database Adapter.

If you have no indexes at all and prefer to have none, you can proceed with the single node multi-threaded approach (see Distributed Polling Second Best Practice: Tuning on a Single Node First). That way the polling query is executed once, which might be a full table scan, but multiple threads help to exhaust the entire result set until all rows are processed. With a distributed approach, all work must be done while the rows are exclusively locked, which means locked in a timed transaction. In a distributed scenario there are many repeated selects, which can harm performance if each one is doing a full table scan.

Note:

Performance is very slow if `MarkUnreadValue` is configured as null.

Disabling Skip Locking

Skip locking has been available on Oracle Database since Oracle 8 but it is documented in Oracle 11. You rarely come across an incompatible feature and have to disable it. If that does occur, you can set the Oracle Database Adapter connector property `usesSkipLocking` to `false` in the `ra.xml` file you deploy with your application as shown in the example below.

Example - Configuring usersSkipLocking in ra.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd" version="1.5">
...
  <resourceadapter>
        <outbound-resourceadapter>
            <connection-definition>
...
            <config-property>
              <config-property-name>usesSkipLocking</config-property-name>
              <config-property-type>java.lang.Boolean</config-property-type>
                    <config-property-value>false</config-property-value>
```

```
            </config-property>
...
            </connection-definition>
...
        </outbound-resourceadapter>
  </resourceadapter>
</connector>
```

For more information on how to configure connector-level properties, see:

- *Configuring the ra.xml File* in the *Developing Resource Adapters for Oracle WebLogic Server*

  .

- *Packaging and Deploying Resource Adapters* in the *Developing Resource Adapters for Oracle WebLogic Server*.

Note:

The Database Adapter Configuration Wizard and Database Adapter runtime might add a `AND ROWNUM <= ?` clause to the SQL statements under the one of the following conditions:

- `usesSkipLocking` is set to "false"
- `MarkReservedValue` is used
- `ReturnSingleResultSet` is set to "true"

The added `AND ROWNUM <= ?` limits the number of rowsreturned from the SQL query. However, because of the `AND ROWNUM <= ?` clause, the returned rows might not be in the order in which the rows are inserted. To prevent the `AND ROWNUM <= ?` clause from being added to the query, you can add a property to the adapter `db.jca` file:

```
<property name="UseRowNumClause" value="false"/>
```

Without the `AND ROWNUM <= ?` clause, the returned rows will be in the order in which the rows are inserted.

MarkReservedValue and Skip Locking

If you are using Logical Delete polling and you set `MarkReservedValue`, skip locking is not used.

Formerly, the best practice for multiple Oracle Database Adapter process instances deployed to multiple Oracle BPEL Process Manager or Oracle Mediator nodes was essentially using `LogicalDeletePollingStrategy` or `DeletePollingStrategy` with a unique `MarkReservedValue` on each polling node, and setting `MaxTransactionSize`.

However with the introduction of skip locking, that approach has now been superseded. If you were using this approach previously, you can simply remove (in `db.jca`) or clear (Logical Delete Page of wizard) the `MarkReservedValue`, and you automatically get skip locking.

The benefits of using skip locking over a reserved value include:

- Skip locking scales better in a cluster and under load.
- All work is in one transaction (as opposed to update/reserve, then commit, then select in a new transaction), so the risk of a non-recoverable situation in an HA environments is minimized.
- No unique `MarkReservedValue` must be specified. For this to work you had to configure a complex variable like `R${weblogic.Name-2}-${IP-2}-${instance}`.

SequencingPollingStrategy (Last Read or Last Updated)

This distributed approach works with Delete or Logical Delete based polling strategies.

The work of the sequencing polling based strategies cannot be distributed as records are initially processed in order.

For example, the second row cannot be marked as processed ahead of the first (setting last read ID to 2 means not just that 2 has been processed but 1 also).

However, as the sequencing polling strategies are non-intrusive, requiring no post updates or deletes to the source tables, they are extremely fast.

Use sequencing polling strategies with a single node and with fan-out on a cluster. It is still safe to use in a cluster; however, the select for update is instead applied on accessing the last read ID in the helper table.

Distributed Polling Second Best Practice: Tuning on a Single Node First

The next best practice for multiple Oracle Database Adapter process instances deployed to multiple Oracle BPEL PM or Mediator nodes is to tune on a single node first.

For an Oracle Database Adapter intensive process, such as a database-database integration, performance can be improved by a factor 10 or 100 just by tuning on a single Java Virtual Machine (JVM), scaling |NumberOfThreads|, and setting high values for `MaxTransactionSize` and `MaxRaiseSize`.

As Distributed Polling First Best Practice: SELECT FOR UPDATE (SKIP LOCKED) describes, there may be times where it is best to improve performance on a single node, and then optionally do fan-out to multiple nodes in a cluster. Relying on concurrency control features of the database such as locking can be great, but these are often designed more for preserving data integrity than for high performance scalability.

Cases where it may be best to do polling on a single node in the cluster include using the non-intrusive Sequencing Polling strategy, polling large un-indexed tables, or using a non-Oracle back-end database that does not provide high concurrency locks like skip locks.

Note:

Wth the Oracle Database Adapter with polling operation in a clustered environment, you must use the option of distributed polling by selecting the **Distributed Polling** check box in the Adapter Configuration Wizard.

You can also refer to Singleton (Active/Passive) Inbound Endpoint Lifecycle Support Within Adapters.

Performance Tuning

The Oracle Database Adapter is preconfigured with many performance optimizations. You can, however, make some changes to reduce the number of round trips to the database by implementing performance tuning.

For information about performance tuning, see: Database Adapter Performance and Tuning

`detectOmissions` Feature

The following are the features of the `detectOmission` feature:

Available Since

Release 10.1.3

Configurable

Yes

Default Value

Design Time: `true`, unless explicitly set to `false`

Use Case

Users can pass incomplete or partial XML to a merge, update, or insert, and see that every column they left unspecified in XML is set to null in the database.

This features enables the `DBAdapter` merge, insert, or update to differentiate between `null` value and the absence of a value (omission) in XML documents. On a case by case basis, it determines which information in XML is meaningful and which is not. In this way, XML is seen as a partial representation of a database row, different from a complete representation. The following table lists examples for null values, and values that can be omitted.

Table 9-3 Examples for Null Values

Element Type	Omission	Null

Column	`<director></director>`
	`<director />`
	`<!-- director>…</director -->`
		`<director xsi:nil="true" />`

1-1	<!-- dept> … </dept -->	`<dept xsi:nil="true" />`
1-M	`<!-- empCollection>…` `</empCollection -->`	`</empCollection>` `</empCollection>` (empty)

Note:

The 1-1 representation `<dept />` denotes an empty department object and should not be used. For 1-M, `<empCollection />` actually means a collection of 0 elements and is considered a meaningful value.For columns, `<director></director>` is not considered an omission in cases where it represents an empty string.

A value considered omitted is omitted from `UPDATE` or `INSERT` SQL. For an update operation, existing (meaningful) values on the database are not overwritten. For an insert operation, the default value on the database is used, as no explicit value is provided in the SQL string.

A `DBAdapter receive` is not able to produce `XML` with omissions, and makes use of `xsi:nil="true"`. If you are unable to produce input XML with `xsi:nil="true"`, or are concerned about the difference between `<director />` and `<director></director>`, then it is best to set `DetectOmissions="false"` in the `JCA` file.

When you are expecting an update, you can improve performance, by omitting `1-1` and `1-M` relationships. The `merge` operation can skip considering the detail records completely.

Alternatively, you can map only those columns that you are interested in, and create separate mappings for different invokes. If two updates are to update two different sets of columns, create two separate `partnernlinks`.

Performance

By default, `XML` is not used as an input to the Oracle Database Adapter containing omissions. Until an `XML` with omissions is detected, there is no performance overhead.

Once omissions are detected, a `TopLink descriptor event listener` is added. This `event listener` has some overhead, and every `modifyRow` about to become a `SQLUpdate` or `SQLInsert` must be iterated over, to check for omissions. Hence, every column value sent to the database is checked. If the input `XML` has mostly omissions, then the cost overhead should be more than compensated by sending fewer values to the database.

Incompatible Interactions

`DirectSQL="true"` and `DetectOmissions="true"` - `DetectOmissions` takes precedence. The following are some examples for incompatible interactions:

- `DetectOmissionsMerge`
- `IgnoreNullsMerge`
- `OptimizeMerge`

Note:

For migrated old BPEL project, you must re-run the Database Adapter Wizard to regenerate the JCA file. When re-run the Database Adapter Wizard, the `DetectOmissions` and `OptimizeMerge` options appear in the JCA file with default values as `DetectOmissions="false"` and `OptimizeMerge="false"`.

See the following for more information:

You can also access the forums from Oracle Technology Network at

- The Oracle BPEL Process Manager forum at

  http://forums.oracle.com/forums/forum.jspa?forumID=212

- The TopLink forum at

  http://forums.oracle.com/forums/forum.jspa?forumID=48

  This site contains over 2,000 topics, such as implementing native sequencing, optimistic locking, and JTA-managed connection pools with TopLink

  http://www.oracle.com/technology

`OutputCompletedXml` Feature

`OutputCompletedXml` is a feature of the outbound `insert` activity. The following are some of the features of the `OutputCompletedXml` feature:

Available Since

Release 10.1.2.0.2

Configurable

`OutputCompletedXml` appears in the JCA file only when default is `true`.

Default Value

OutputCompletedXml is `true` when `TopLink` sequencing is configured to assign primary keys on insert from a database sequence; otherwise OutputCompletedXml is `false`.

Issue

You can have primary keys auto-assigned on `insert` from a database sequence. However, the usefulness of this feature is diminished, because `insert/merge` have no output message, so there is no way to tell which primary keys were assigned.

Note:

After configuring sequencing (link), run the Adapter Configuration Wizard again so that the `insert/merge` `WSDL` operations can be regenerated with an output message, and `WSDL` property `OutputCompletedXml="true"`.

Performance

An `output` `XML` is provided only when the `output` `XML` is significantly different, so if `TopLink` sequencing is not used, this feature is disabled and there is no performance hit. Further, this feature can be explicitly disabled. Likewise, the original input `XML` is updated and returned; a completely new `XML` is not built. Additionally, only a shallow update of the `XML` is performed; if primary keys were assigned to detail records, then these are not reflected in the output `XML`.

Incompatible Interactions

When `DirectSQL="true"` and `OutputCompletedXml` are present, `OutputCompletedXml` takes precedence.

QueryTimeout for Inbound and Outbound Transactions

You can configure `QueryTimeout` from the Adapter Configuration Wizard **Advanced Options** page. This feature exposes the `java.sql.Statement` level property of the same name. Essentially, `QueryTimeout` enables you to configure a timeout on the call.

Doing Synchronous Post to BPEL (Allow In-Order Delivery)

In this feature, the entire invocation is in a single thread and global transaction. By default, initiation is asynchronous and the BPEL process is invoked in a separate global transaction. With Oracle Mediator, it is generally a synchronous invoke so this is only specific to an Oracle BPEL process.

To enable this feature, click the **Do Synchronous Post to BPEL (Allow In-Order Delivery)** option in the Adapter Configuration Wizard **Operation** page.

Oracle Database Adapter Concepts

This section includes the following topics related to Oracle Database Adapter Concepts:

- Relational-to-XML Mapping
- SQL Operations as Web Services

Relational-to-XML Mapping

This section includes the following topics related to Relational-to-XML mapping:

- Relational Types to XML Schema Types
- Mapping Any Relational Schema to Any XML Schema
- Querying over Multiple Tables

For a flat table or schema, the relational-to-XML mapping is easy to see. Each row in the table becomes a complex XML element. The value for each column becomes a text node in the XML element. Both column values and text elements are primitive types.

Table 9-4 shows the structure of the `MOVIES` table. This table is used in the use cases described in this chapter. See Use Cases for more information.

Table 9-4 MOVIES Table Description

Name	Null?	Type
`TITLE`	NOT NULL	`VARCHAR2(50)`
`DIRECTOR`	--	`VARCHAR2(20)`
`STARRING`	--	`VARCHAR2(100)`
`SYNOPSIS`	--	`VARCHAR2(255)`
`GENRE`	--	`VARCHAR2(70)`
`RUN_TIME`	--	`NUMBER`

Table 9-4 (Cont.) MOVIES Table Description

Name	Null?	Type
RELEASE_DATE	--	DATE
RATED	--	VARCHAR2(6)
RATING	--	VARCHAR2(4)
VIEWER_RATING	--	VARCHAR2(5)
STATUS	--	VARCHAR2(11)
TOTAL_GROSS	--	NUMBER
DELETED	--	VARCHAR2(5)
SEQUENCENO	--	NUMBER
LAST_UPDATED	--	DATE

The corresponding XML schema definition (XSD) is shown in the example below.

Example - XSD for Movies Collection

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<xs:schema targetNamespace="http://xmlns.oracle.com/
                pcbpel/adapter/db/top
/ReadS1" xmlns="http://xmlns.oracle.com/pcbpel/
adapter/db/top/ReadS1" elementFormDefault=
"qualified" attributeFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="MoviesCollection"
              type="MoviesCollection"/>
  <xs:complexType name="MoviesCollection">
     <xs:sequence>
        <xs:element name="Movies" type="Movies"
                            minOccurs="0"
maxOccurs="unbounded"/>
     </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Movies">
     <xs:sequence>
        <xs:element name="title">
           <xs:simpleType>
              <xs:restriction base="xs:string">
                 <xs:maxLength value="50"/>
              </xs:restriction>
           </xs:simpleType>
        </xs:element>
        <xs:element name="director" minOccurs="0"
                            nillable="true">
           <xs:simpleType>
              <xs:restriction base="xs:string">
                 <xs:maxLength value="20"/>
              </xs:restriction>
           </xs:simpleType>
        </xs:element>
        <xs:element name="starring" minOccurs="0"
                              nillable="true">
```

```
                  <xs:simpleType>
                     <xs:restriction base="xs:string">
                        <xs:maxLength value="100"/>
                     </xs:restriction>
                  </xs:simpleType>
               </xs:element>
               <xs:element name="synopsis" minOccurs="0"
                                    nillable="true">
                  <xs:simpleType>
                     <xs:restriction base="xs:string">
                        <xs:maxLength value="255"/>
                     </xs:restriction>
                  </xs:simpleType>
               </xs:element>
               <xs:element name="genre" minOccurs="0"
                                       nillable="true">
                  <xs:simpleType>
                     <xs:restriction base="xs:string">
                        <xs:maxLength value="70"/>
                     </xs:restriction>
                  </xs:simpleType>
               </xs:element>
               <xs:element name="runTime" type="xs:decimal"
                                    minOccurs="0"
                                       nillable="true"/>
               <xs:element name="releaseDate" type="xs:dateTime"
                                       minOccurs="0"
      nillable="true"/>
               <xs:element name="rated" minOccurs="0"
                                        nillable="true">
                  <xs:simpleType>
                     <xs:restriction base="xs:string">
                     </xs:restriction>
                  </xs:simpleType>
               </xs:element>
               <xs:element name="rating" minOccurs="0"
                                        nillable="true">
                  <xs:simpleType>
                     <xs:restriction base="xs:string">
                        <xs:maxLength value="4"/>
                     </xs:restriction>
                  </xs:simpleType>
               </xs:element>
               <xs:element name="viewerRating" minOccurs="0"
                                        nillable="true">
                  <xs:simpleType>
                     <xs:restriction base="xs:string">
                        <xs:maxLength value="5"/>
                     </xs:restriction>
                  </xs:simpleType>
               </xs:element>
               <xs:element name="status" minOccurs="0"
                                         nillable="true">
                  <xs:simpleType>
                     <xs:restriction base="xs:string">
                        <xs:maxLength value="11"/>
                     </xs:restriction>
                  </xs:simpleType>
               </xs:element>
               <xs:element name="totalGross"
                              type="xs:decimal" minOccurs="0"
```

```
                              nillable="true"/>
       <xs:element name="deleted" minOccurs="0"
                         nillable="true">
          <xs:simpleType>
             <xs:restriction base="xs:string">
                <xs:maxLength value="5"/>
             </xs:restriction>
          </xs:simpleType>
       </xs:element>
       <xs:element name="sequenceno" type="xs:decimal"
                   minOccurs="0"
 nillable="true"/>
       <xs:element name="lastUpdated" type="xs:dateTime"
                   minOccurs="0"
 nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

As the preceding code example shows, `MOVIES` is not just a single `CLOB` or `XMLTYPE` column containing the entire XML string. Rather, it is an XML `complexType` comprising elements, each of which corresponds to a column in the `MOVIES` table. For flat tables, the relational-to-XML mapping is straightforward.

Table 9-5 and Table 9-6 show the structure of the `EMP` and `DEPT` tables, respectively. These tables are used in the `MasterDetail` use case. See Use Cases for more information.

Table 9-5 EMP Table Description

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME	--	VARCHAR2(10)
JOB	--	VARCHAR2(9)
MGR	--	NUMBER(4)
HIREDATE	--	DATE
SAL	--	NUMBER(7,2)
COMM	--	NUMBER(7,2)
DEPTNO	--	NUMBER(2)

Table 9-6 DEPT Table Description

Name	Null?	Type
DEPTNO	NOT NULL	NUMBER(2)
DNAME	--	VARCHAR2(14)
LOC	--	VARCHAR2(13)

As the preceding table definitions show, and as is typical of a normalized relational schema, an employee's department number is not stored in the EMP table. Instead,

one of the columns of EMP (DEPTNO) is a foreign key, which equals the primary key (DEPTNO) in DEPT.

However, the XML file equivalent has no similar notion of primary keys and foreign keys. Consequently, in the resulting XML file, the same data is represented in a hierarchy, thereby preserving the relationships by capturing the detail record embedded inside the master.

An XML element can contain elements that are either a primitive type (`string`, `decimal`), or a complex type, that is, another XML element. Therefore, an employee element can contain a department element.

The corresponding XML shows how the relationship is materialized, or shown inline. DEPTNO is removed from EMP, and instead you see the DEPT itself.

```
<EmpCollection>
  <Emp>
    <comm xsi:nil = "true" ></comm>
    <empno >7369.0</empno>
    <ename >SMITH</ename>
    <hiredate >1980-12-17T00:00:00.000-08:00</hiredate>
    <job >CLERK</job>
    <mgr >7902.0</mgr
    <sal >800.0</sal>
    <dept>
      <deptno >20.0</deptno>
      <dname >RESEARCH</dname>
      <loc >DALLAS</loc>
    </dept>
  </Emp>
    ...
</EmpCollection>
```

Materializing the relationship makes XML human readable and enables the data to be sent as one packet of information. No cycles are allowed in the XML file; therefore, an element cannot contain itself. This is handled automatically by the Oracle Database Adapter.

However, you might see duplication (that is, the same XML detail record appearing more than once under different master records). For example, if a query returned two employees, both of whom work in the same department, then, in the returned XML, you can see the same DEPT record inline in both the EMP records.

Therefore, when you import tables and map them as XML, you must avoid excessive duplication, although the Oracle Database Adapter does not print an element inside itself. The Oracle Database Adapter prints the following:

```
<Emp>
  <name>Bob</name>
  <spouse>
    <name>June</name>
  </spouse
</Emp>
```

But not:

```
<Emp>
  <name>Bob</name>
  <spouse>
    <name>June</name>
    <spouse>
      <name>Bob</name>
```

```
      <spouse>
        ...
      </spouse>
    </spouse>
  </spouse>
</Emp>
```

To avoid duplication, you can do the following:

- Import fewer tables. If you import only `EMP`, then `DEPT` does not appear.
- Remove the relationship between `EMP` and `DEPT` in the Adapter Configuration Wizard. This removes the relationship, but the foreign key column is put back.

In both these cases, the corresponding XML is as follows:

```
<EmpCollection>
  <Emp>
    <comm xsi:nil = "true" ></comm>
    <empno >7369.0</empno>
    <ename >SMITH</ename>
    <hiredate >1980-12-17T00:00:00.000-08:00</hiredate>
    <job >CLERK</job>
    <mgr >7902.0</mgr>
    <sal >800.0</sal>
    <deptno >20.0</deptno>
  </Emp>
   ...
</EmpCollection>
```

Either preceding solution is feasible only if returning foreign key suffices, as distinct from getting back the complete detail record in its entirety.

Relational Types to XML Schema Types

Table 9-7 shows how database data types are converted to XML primitive types when you import tables from a database.

Table 9-7 Mapping Database Data Types to XML Primitive Types

Database Type	XML Type (Prefixed with xs:)
VARCHAR, VARCHAR2, CHAR, NCHAR, NVARCHAR, NVARCHAR2, MEMO, TEXT, CHARACTER, CHARACTER VARYING, UNICHAR, UNIVARCHAR, SYSNAME, NATIONAL CHARACTER, NATIONAL CHAR, NATIONAL CHAR VARYING, NCHAR VARYING, LONG, CLOB, NCLOB, LONGTEXT, LONGVARCHAR, NTEXT	string
BLOB, BINARY, IMAGE, LONGVARBINARY, LONG RAW, VARBINARY, GRAPHIC, VARGRAPHIC, DBCLOB, BIT VARYING	base64Binary
BIT, NUMBER(1) DEFAULT 0, SMALLINT DEFAULT 0, SMALLINT DEFAULT 0	boolean
TINYINT, BYTE	byte

Table 9-7 (Cont.) Mapping Database Data Types to XML Primitive Types

Database Type	XML Type (Prefixed with xs:)
SHORT, SMALLINT	short
INT, SERIAL	int
INTEGER, BIGINT	integer
NUMBER, NUMERIC, DECIMAL, MONEY, SMALLMONEY, UNIQUEIDENTIFIER	decimal
FLOAT FLOAT16, FLOAT(16), FLOAT32, FLOAT(32), DOUBLE, DOUBLE PRECIS, REAL	double
TIME, DATE, DATETIME, TIMESTAMP, TIMESTAMP(6), SMALLDATETIME, TIMESTAMPTZ, TIMESTAMPLTZ, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE	dateTime

Essentially, NUMBER goes to DECIMAL, the most versatile XML data type for numbers, VARCHAR2 and CLOB to string, BLOB to base64Binary (to meet the plain-text requirement), and date types to dateTime.

Any type not mentioned in this discussion defaults to java.lang.String and xs:string.

Time Stamp support is basic, because only the xs:dateTime format is supported. The BFILE type is specifically not supported.

Note:

The user-defined Object, Struct and VARRAY, and REF types are supported in releases 11g and 12.

Because XML is plain text, BLOB and byte values are base 64/MIME encoded so that they can be passed as character data.

Mapping Any Relational Schema to Any XML Schema

The Oracle Database Adapter supports mapping any relational schema on any relational database to an XML schema, although not any XML schema of your choice, because the Adapter Configuration Wizard generates the XML schema with no explicit user control over the layout of the elements.

You can control how you map the schema in both the Adapter Configuration Wizard and later in TopLink Workbench. By pairing the Oracle Database Adapter with a transformation step, you can map any relational schema to any XML schema.

Querying over Multiple Tables

When executing a SQL select statement against multiple related tables there are the following three methods to build the SQL. These ways relate to how to pull in the detail records when the query is against the master record:

- Using Relationship Queries (TopLink Default)
- Twisting the Original Select (TopLink Batch-Attribute Reading)
- Returning a Single Result Set (TopLink Joined-Attribute Reading)
- Comparison of the Methods Used for Querying over Multiple Tables

The following sections contain an outline of these three methods and their comparison. When selecting rows from a single table there are no issues different from selecting from multiple tables.

Using Relationship Queries (TopLink Default)

Having selected a `Master` row, `TopLink` can always query separately to get all the details belonging to that `Master` table. These hidden queries (relationship queries) are cached in the `TopLink` metadata and must be prepared only once.

Consider the SQL statement in following sample scenario:

```
SELECT DIRECTOR, ..., VIEWER_RATING
       FROM MOVIES
WHERE RATING = 'A';
```

For each master, the SQL statement is as follows:

```
SELECT CRITIC, ..., TITLE
       FROM MOVIE_REVIEWS
WHERE (TITLE = ?)
```

This enables you to bring in all the data with 1 + n query executions, where n is the number of master rows returned by the first query.

This approach is safe but slow, as a large number of round trips to the database are required to pull in all the data.

For configuring using the relationship Queries (TopLink default) approach, you must edit `or_mappings.xml` outside of JDeveloper. In addition, change the batch-reading elements value to false.

Twisting the Original Select (TopLink Batch-Attribute Reading)

This is a default feature that allows `TopLink` to alter the original SQL `select` statement to read all the details in a second `select` statement, as shown in the following example:

```
SELECT DIRECTOR, ..., VIEWER_RATING
FROM MOVIES
WHERE RATING = 'A'
SELECT DISTINCT t0.CRITIC, ..., t0.TITLE
FROM MOVIE_REVIEWS t0, MOVIES t1
WHERE ((t1.RATING = 'A') AND (t0.TITLE = t1.TITLE))
```

By considering the original `select` statement in pulling in the details, a total of two (1 + 1 = 2) query executions must be performed.

Advantages

Batch attribute reading has the following advantages:

- All data read in two round trips to database

- The is a default feature in the 10.1.2.0.2 release

Disadvantages

Batch attribute reading has the following disadvantages:

- When using `maxTransactionSize` (on polling receive) or `maxRows` (on invoke select) to limit the number of rows loaded into memory at a time, these settings do not easily carry over to the batch attribute query. It is easier to work with a cursored result when there is only a single result set. (Multiple cursors can be used with difficulty, if the original query has an order by clause).
- `TopLink` can alter a SQL statement, only when it is in a format it can understand. If you use the hybrid SQL approach and set custom SQL for the root `select`, then `TopLink` cannot interpret that SQL to build the batch `select`.
- The `DISTINCT` clause is used on the batch query, to avoid returning the same detail twice if two masters happen to both point to it. The `DISTINCT` clause cannot be used when returning LOBs in the resultset.

Configuration

Configuration is on a per `1-1` or `1-M` mapping basis. By default, all such mappings since the 10.1.2.0.2 release have this property set. To configure, edit `or_mappings.xml` outside JDeveloper and edit the <batch-reading> elements to true (default) or false.

Returning a Single Result Set (TopLink Joined-Attribute Reading)

The detail tables are outer-joined to the original SQL `select` statement, returning both master and detail in a single result set, as shown in the following example:

```
SELECT DISTINCT t1.DIRECTOR, ..., t1.VIEWER_RATING, t0.CRITIC, ..., t0.TITLE
FROM MOVIE_REVIEWS t0, MOVIES t1
WHERE ((t1.RATING = 'A') AND (t0.TITLE (+) = t1.TITLE))
```

This requires one query execution in total.

Advantages

The advantages include the following:

- In case of using `maxTransactionSize` while polling, the benefits of dealing with a single cursor can be great.
- When following the hybrid SQL route and entering custom SQL statements, you only have to deal with a single SQL statement, whereas `TopLink` normally uses a series of additional hidden SQL statements to bring in related rows.
- `read` consistency: Enables you to read all related rows at the same time, and not at different instances in time for the different tables.
- Performance can be ideal as only a single round trip to the database is required, whereas batch attribute reading requires one for each table queried.

Disadvantages

There are some drawbacks, however, namely the cost of returning duplicate data. For example, consider that you read the `Master` and `Detail` tables; `Master` has 100 columns in each row, and `Detail` has 2 columns in each row. Each row in the table, `Master` also, typically has 100 related `Detail` rows.

With one query in each table, the result sets for the preceding example appears, as shown in the following example:

```
Master
Column1 column2 ….. column100

Master1 ...

Detail

Detail
Column1 column2
Detail1   ...
Detail2
...
Detail100 ...
```

In this example, 300 column values are returned as shown:

```
(columns in master + columns in detail x details per master) =
(          100            +           2               x
100        )  =  300
```

With one query for all tables, the result set appears, as shown in the following example:

```
Master                                          Detail

Column1 Column2 ... Column100                   Column1  Column2
Master1    ...                                  Detail1     ...
Master1    ...                                  Detail2     ...
Master1    ...                                  Detail100   ...
```

When there is one query for all tables, 10,200 column values are returned in a single result set, versus 300 in two result sets, as shown here:

```
((columns in master + columns in detail) x details per master) =
((         100     +           2     ) x       100         ) =  10,200
```

This can have a serious drain on network traffic and computation because 97 percent of the data returned is duplicate data. Also, if the master had two related tables `detail1` and `detail2` and there were 100 each in each master, then the number of column values returned would be over 10 million per master row.

In general, you can use the following simple formula to estimate the relative cost of returning all rows in a single result set:

$$\text{bloat} = \frac{\begin{array}{l}\text{(Master columns + Detail1 columns + Detail2 columns + ...) x}\\ \text{Detail1s per Master x}\\ \text{Detail2s per Master x ...}\end{array}}{\begin{array}{l}\text{(Master columns + Detail1 columns X Detail1s per Master +}\\ \text{Detail2 columns X Detail2s per Master + ...)}\end{array}}$$

For `1-1` relationships, this value is always 1, and if in the same example each master had two columns only and the details had 100 columns instead, and each master had only 3 or 4 details each, then the bloat would be

$$\text{bloat} = \frac{(2 + 100) \times 4}{(2 + 100 \times 4)} = \frac{408}{402} \sim= 1$$

Another disadvantage is that this setting could distort the meaning of the `maxRows` setting on an outbound select.

Configuration

To configure, select **Use Outer Joins to return a Single Result Set for both Master and Detail Tables** on the Adapter Configuration Wizard - Define Selection Criteria page.

Note:

When you create a SQL query such as the following by using the TopLink Expression Builder, the result may not be as expected:

```
SELECT DISTINCT t1.TABLE1_ID, t1.COLUMN_A FROM TABLE2 t0,
TABLE1 t1 WHERE ((t0.STATUS = 1) AND (t0.TABLE1_ID = t1.
TABLE1_ID))
```

The expected result for this query is that only rows with Table 1's and their owned Table 2's with status = 1 be returned.

However, what this query actually translates to is "table 1's, where any of its table 2's have status = 1," resulting in the return of table 1's that match the selection criteria, and `ALL` of the table 2's they own, including those with other statuses, whether or not their statuses =1. The `DISTINCT` keyword ensures the table 1's are not repeated and the join happens across table 2.

The misunderstanding happens in the way Toplink works. Through the Expression Builder, you can only specify a selection criteria for Table 1 and have no control over the Table 2's they own, this part is automatically done.

However, you can get the expected result by using either of the following two approaches:

1.) Query directly for table 2 using the selection criteria of status = 1, that is, do not go through table 1 and get the table 2's they own.

2.) Use direct (custom `SQL`), as shown in the following example:

```
SELECT TABLE1.TABLE1_ID, TABLE1.COLUMN_A, TABLE2.STATUS
FROM TABLE2, TABLE1 WHERE TABLE2.STATUS=1 AND TABLE1.
TABLE1_ID = TABLE2.TABLE1_ID
```

Comparison of the Methods Used for Querying over Multiple Tables

Superficially, returning a single result set looks best (1 query), followed by batch attribute reading (altering the `select` statement: 2 queries), and finally by default relationship reading (n + 1 queries). However, there are several pitfalls to both of the more advanced options.

Altering User-Defined SQL

If you specify `custom/hybrid` SQL, the `TopLink` cannot alter that SQL string to build the details `select`. For this reason, you must avoid using `hybrid` SQL and build `selects` using the wizards' `visual expression builder` as often as possible.

Show Me the SQL

The additional queries executed by TopLink in both, the default and the batch attribute reading cases can be somewhat of a mystery to users. For this reason, the raw SQL shown to users in the Adapter Configuration Wizard assumes returning a single result set, to make things clearer and also to improve manageability.

Returning Too Many Rows At Once

Databases can store vast quantities of information, and a common pitfall of select statements which return too much information at once. On a Database Adapter receive, a maxTransactionSize property can be set to limit the number of rows which are read from a cursored result set and processed in memory at a time. A similar max-rows setting exists for one-time invoke select statements. However, this setting is very risky.

SQL Operations as Web Services

After mapping a relational schema as XML, you must also map basic SQL operations as Web services. Some operations translate directly to the SQL equivalent, while others are more complex.

This section includes the following topics:

- DML Operations
- Polling Strategies

DML Operations

Data manipulation language (DML) operations align with basic SQL INSERT, UPDATE, and SELECT operations. SQL INSERT, UPDATE, DELETE, and SELECT are all mapped to Web service operations of the same name. The MERGE is either an INSERT or UPDATE, based on the results of an existence check. A distinction is made between the data manipulation operations—called outbound writes—and the SELECT operations—called outbound reads. The connection between the Web service and the SQL for merge (the default for outbound write) and queryByExample are not as obvious as for basic SQL INSERT, UPDATE, and SELECT.

This section includes the following topics:

- Merge
- querybyExample

Merge

Merge first reads the corresponding records in the database, calculates any changes, and then performs a minimal update. INSERT, UPDATE, and MERGE make the most sense when you are thinking about a single row and a single table. However, your XML can contain complex types and map to multiple rows on multiple tables. Imagine a DEPT with many EMPS, each with an ADDRESS. In this case, you must calculate which of possibly many rows have changed and which to insert, update, or delete. If a specific row did not change or only one field changed, then the DML calls is minimal.

querybyExample

Unlike the SELECT operation, queryByExample does not require a selection criteria to be specified at design time. Instead, for each invoke, a selection criteria is inferred from an exemplary input XML record.

For instance, if the output `xmlRecord` is an employee record, and the input is a sample `xmlRecord` with `lastName = "Smith"`, then on execution, all employees with a last name of Smith are returned.

A subset of `queryByExample` is to query by primary key, which can be implemented by passing in sample XML records where only the primary key attributes are set.

Use `queryByExample` when you do not want to create a query using the visual query builder and want the flexibility of allowing the input record to share the same XML schema as the output records.

The `queryByExample` operation is slightly less performant because a new `SELECT` must be prepared for each execution. This is because the attributes that are set in the example XML record can vary each time, and therefore the selection criteria vary.

Input `xmlRecord`:

```
<Employee>
      <id/>
      <lastName>Smith</lastName>
</Employee>
```

Output `xmlRecord`:

```
<EmployeeCollection>
      <Employee>
         <id>5</id>
         <lastName>Smith</lastName>
         ....
      </Employee>
      <Employee>
         <id>456</id>
         <lastName>Smith</lastName>
         ....
      </Employee>
</EmployeeCollection>
```

Restriction:

- When using `queryByExample` the query is dynamically generated each time the database adapter is invoked as each time the query might be different unlike the regular select query, where the query is cached.
- The `queryByExample` feature does not allow the user to set a limit on the maximum number of rows to return the results. This results in the return of a large number of rows. This in turn increases the memory requirement to process the rows.

Polling Strategies

The inbound receive enables you to listen to and detect events and changes in the database, which in turn can be the initiators of a business process. This is not a one-time action, but rather an activation. A polling thread is started, which polls a database table for new rows or events.

Whenever a new row is inserted into the `MOVIES` table, the polling operation raises it to the SCA runtime. The strategy is to poll every record once. The initial `SELECT` has to be repeated over time, to receive the rows that exist at the start and all new rows as

they are inserted over time. However, a new row once read is not likely to be deleted, and therefore can possibly be read repeatedly with each polling.

The various ways to poll for events, called polling strategies, also known as after-read strategies or publish strategies, range from simple and intrusive to sophisticated and non-intrusive. Each strategy employs a different solution for the problem of what to do after reading a row or event so it is not picked up again in the next polling interval. The simplest (and most intrusive) solution is to delete the row so that you do not query it again.

This section discusses the following polling operations that you can perform after the data is read from the database. This section also discusses the strategies and factors to help you determine which strategy to employ for a particular situation:

- Delete the Row(s) that were Read
- Update a Field in the [Table_Name] Table (Logical Delete)
- Update a Sequencing Table
- Update an External Sequencing Table on a Different Database
- Control Table Strategy
- Update a Sequencing File

Delete the Row(s) that were Read

Choose this operation to employ the physical delete polling strategy. This operation polls the database table for records and deletes them after processing. Use this strategy to capture events related to `INSERT` operations and cannot capture database events related to `DELETE` and `UPDATE` operations on the parent table. This strategy cannot be used to poll child table events. This strategy allows multiple adapter instances to go against the same source table. There is zero data replication.

Preconditions: You must have deletion privileges on the parent and associated child tables to use the delete polling strategy. Table 9-8 describes the requirements for using the delete polling strategy.

Table 9-8 Delete Polling Strategy Preconditions

Requirements Met	Conflicts With
Poll for inserts	No delete on source
Shallow delete	No updates on source
Cascading delete	Poll for updates
Minimal SQL	Poll for deletes
Zero data replication	Poll for child updates
Default	--
Allows raw SQL	--
Concurrent polling	--

Note:

In *Shallow delete* and *Cascading delete*, the delete operation can be configured to delete the top-level row, to cascade all, or to cascade on a case-by-case basis.

Concurrent polling can be configured for both delete and logical delete polling strategies.

Configuration: You can configure the delete polling strategy to delete the top-level row, to cascade all, or to cascade on a case-by-case basis. This strategy enables deleting only the parent rows and not the child rows, cascaded deletes, and optional cascaded deletes, determined on a case-by-case basis. You can configure the polling interval for performing an event publish at design time.

Delete Cascade Policy: The optional advanced configuration is to specify the cascade policy of the `DELETE` operation. For instance, after polling for an employee with an address and many phone numbers, the phone numbers are deleted because they are privately owned (default for one-to-many), but not the address (default for one-to-one). This can be altered by configuring `or_mappings.xml`, as in the following example:

```
<database-mapping>
   <attribute-name>orders</attribute-name>
   <reference-class>taxonomy.Order</reference-class>
   <is-private-owned>true</is-private-owned>
```

You can also configure the activation itself to delete only the top level (master row) or to delete everything.

Note:

A PrivateOwned annotation is used to specify a relationship is privately owned. A privately owned relationship means the target object is a dependent part of the source object and is not referenced by any other object and cannot exist on its own. Private ownership causes operations to be cascaded across the relationship, including these: deletion, insertion, refreshing, locking (when locking is cascaded).

PrivateOwned also ensures that private objects removed from collections are deleted and object added are inserted.

A receive operation appears in an inbound JCA as follows:

Example - Receive Operation Inbound JCA

```
<connection-factory location="eis/DB/Connection1"
            UIConnectionName="Connection1" adapterRef=""/>
  <endpoint-activation portType="dd_ptt" operation="receive">
    <activation-spec className="oracle.tip.adapter.db.
                     DBActivationSpec">
      <property name="DescriptorName" value="dd.Emp"/>
      <property name="QueryName" value="ddSelect"/>
      <property name="MappingsMetaDataURL" value="dd-or-mappings.xml"/>
      <property name="PollingStrategy"
                        value="LogicalDeletePollingStrategy"/>
      <property name="MarkReadColumn" value="STATUS"/>
      <property name="MarkReadValue" value="PROCESSED"/>
```

```
      <property name="MarkReservedValue" value="RESERVED-1"/>
      <property name="MarkUnreadValue" value="UNPROCESSED"/>
      <property name="PollingInterval" value="5"/>
      <property name="MaxRaiseSize" value="1"/>
      <property name="MaxTransactionSize" value="10"/>
      <property name="ReturnSingleResultSet" value="false"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

Update a Field in the [Table_Name] Table (Logical Delete)

Choose this operation to employ the logical delete polling strategy. This strategy involves updating a special field on each row processed and updating the `WHERE` clause at runtime to filter out processed rows.

It mimics logical delete, wherein applications rows are rarely deleted but instead a status column `isDeleted` is set to true. The status column and the read value must be provided, but the modified `WHERE` clause and the post-read update are handled automatically by the Oracle Database Adapter.

Preconditions: You must have the logical delete privilege or a one-time alter schema (add column) privilege on the source table. Table 9-9 describes the requirements for using the logical delete polling strategy.

Table 9-9 Logical Delete Polling Strategy Preconditions

Requirements Met	Conflicts With
Poll for inserts	No updates on source
No delete on source	Poll for deletes
Minimal SQL	--
Zero data replication	--
Minimal configuration	--
Allows raw SQL	--
Poll for updates	--
Poll for child updates	--
Concurrent polling	--

Note:

The requirements of the following are met, as follows:

- **Poll for updates**: By adding a trigger
- **Poll for child updates**: By adding a trigger
- **Concurrent polling**: By specifying additional mark unread and reserved values.

Configuration: The logical delete polling strategy requires minimal configuration. You must specify the mark read column and the value that indicates a processed record.

A receive operation appears in an inbound WSDL as follows:

```
<operation name="receive">
   <jca:operation
      ActivationSpec="oracle.tip.adapter.db.DBActivationSpec"
         …
      PollingStrategyName="LogicalDeletePollingStrategy"
      MarkReadField="STATUS"
      MarkReadValue="PROCESSED"
```

Given the configuration for logical delete, the Oracle Database Adapter appends the following WHERE clause to every polling query:

```
AND (STATUS IS NULL) OR (STATUS <> 'PROCESSED')
```

Database Configuration: A status column on the table being polled must exist. If it does not exist already, you can add one to an existing table.

Support for Polling for Updates: Given that rows are not deleted with each read, it is possible to repetitively read a row multiple times. You must add a trigger to reset the mark read field whenever a record is changed, as follows:

```
create trigger Employee_modified
before update on Employee
for each row
begin
   :new.STATUS := 'MODIFIED';
end;
```

Support for Concurrent Access Polling: Just as a single instance should never process an event more than once, the same applies to a collection of instances. Therefore, before processing a record, an instance must reserve that record with a unique value. Again, the status column can be used:

```
<operation name="receive">
   <jca:operation
      ActivationSpec="oracle.tip.adapter.db.DBActivationSpec"
         …
      PollingStrategyName="LogicalDeletePollingStrategy"
      MarkReadField="STATUS"
      MarkUnreadValue="UNPROCESSED"
      MarkReservedValue="RESERVED${IP-2}-${weblogic.Name-1}-${instance}"
      MarkReadValue="PROCESSED"
```

The polling query instead appears, as shown in the following example:

```
Update EMPLOYE set STATUS = 'RESERVED65-1-1' where (CRITERIA) AND (STATUS =
'UNPROCESSED');

Select … from EMPLOYEE where (CRITERIA) AND (STATUS = 'RESERVED65-1-1');
```

The after-read UPDATE is faster because it can update all:

```
Update EMPLOYEE set STATUS = 'PROCESSED' where (CRITERIA) AND (STATUS =
'RESERVED65-1-1');
```

Update a Sequencing Table

Choose this operation to employ the sequencing table: last-read Id strategy. This polling strategy involves using a helper table to remember a sequence value. The source table is not modified; instead, rows that have been read in a separate helper table are recorded. A sequence value of `1000`, for example, means that every record with a sequence less than that value have been processed. Because many tables have some counter field that is always increasing and maintained by triggers or the application, this strategy can often be used for noninvasive polling. No field on the processed row must be modified by the Oracle Database Adapter.

Native sequencing with a pre-allocation size of `1` can ensure that rows are inserted with primary keys that are always increasing over time.

This strategy is also called a nondestructive delete because no updates are made to the source rows, and you can use a sequencing strategy such as the `sequence` field to order the rows in a sequence for processing.

When the rows are ordered in a line, the Oracle Database Adapter knows which rows are processed and which are not with a single unit of information.

Preconditions: You must have a sequencing table or create table privilege on the source schema. The source table has a column that is monotonically increasing with every `INSERT` (an Oracle native sequenced primary key) or `UPDATE` (the last-modified timestamp). Table 9-10 describes the requirements for using the sequencing polling strategy.

Table 9-10 Sequencing Polling Strategy Preconditions

Requirements Met	Conflicts With
Poll for inserts	Poll for deletes
Poll for updates	Allows raw SQL
No delete on source	Concurrent polling
No updates on source	--
One extra SQL select	--
Zero data replication	--
Moderate configuration	--
Poll for child updates	--

Configuration: A separate helper table must be defined. On the source table, you must specify which column is ever increasing.

Example - Specifying Sequencing

```
<adapter-config name="ReadS" adapter="Database Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
    <connection-factory location="eis/DB/DBConnection1"
UIConnectionName="DBConnection1" adapterRef=""/>
  <endpoint-activation portType="ReadS_ptt" operation="receive">
    <activation-spec className="oracle.tip.adapter.db.DBActivationSpec">
      <property name="DescriptorName" value="ReadS.PerfMasterIn"/>
      <property name="QueryName" value="ReadSSelect"/>
      <property name="MappingsMetaDataURL" value="ReadS-or-mappings.xml"/>
      <property name="PollingStrategy" value="SequencingPollingStrategy"/>
      <property name="SequencingTable" value="PC_SEQUENCING"/>
```

```
      <property name="SequencingColumn" value="PK"/>
      <property name="SequencingTableKeyColumn" value="TABLE_NAME"/>
      <property name="SequencingTableValueColumn" value="LAST_READ_ID"/>
      <property name="SequencingTableKey" value="PERF_MASTER_IN"/>
      <property name="PollingInterval" value="60"/>
      <property name="MaxRaiseSize" value="1"/>
      <property name="MaxTransactionSize" value="10"/>
      <property name="ReturnSingleResultSet" value="false"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

The sequencing field type can be excluded if it is actually a number.

Database Configuration: A sequencing table must be configured once for a given database. Multiple processes can share the same table. Given the `ActivationSpec` specified in the preceding example, the `CREATE TABLE` command looks as follows:

```
CREATE TABLE SEQUENCING_HELPER
(
TABLE_NAME VARCHAR2(32) NOT NULL,
LAST_READ_DATE DATE
)
;
```

Polling for Updates: In the preceding example, the polling is for new objects or updates, because every time an object is changed, the modified time is updated.

A sample trigger to set the modified time on every `insert` or `update` is as follows:

```
create trigger Employee_modified
before insert or update on Employee
for each row
begin
  :new.modified_date := sysdate;
end;
```

Using a Sequence Number: A sequence number can be used for either insert or update polling. Native sequencing returns monotonically increasing primary keys, when an increment by 1 is used. You can also use the sequence number of a materialized view log.

Update an External Sequencing Table on a Different Database

Choose this operation to employ the sequencing table: last updated strategy. This polling strategy involves using a helper table to remember a `last_updated` value. A `last_updated` value of `2005-01-01 12:45:01 000`, for example, means that every record last updated at that time or earlier have been processed.

Because many tables have rows with a `last_updated` or `creation_time` column maintained by triggers or the application, this strategy can often be used for noninvasive polling. Fields on the processed row never require modification by the Oracle Database Adapter.

This strategy is also called a nondestructive delete because no updates are made to the source rows, and you can use a sequencing strategy such as the `last_updated` field to order the rows in a sequence for processing. When the rows are ordered in a line, the Oracle Database Adapter knows which rows are processed and which are not with a single unit of information.

See Update a Sequencing Table for information about preconditions and configuration.

Update a Sequencing File

This strategy works similar to Update an External Sequencing Table on a Different Database, the only difference is that the control information is stored in a file instead of a table.

Control Table Strategy

Choose this operation to employ the control table polling strategy. This polling strategy involves using a control table to store the primary key of every row that has yet to be processed. With a natural join between the control table and the source table (by primary key), polling against the control table is practically the same as polling against the source table directly. However, an extra layer of indirection allows the following:

- Destructive polling strategies such as the delete polling strategy can be applied to rows in the control table alone while shielding any rows in the source table.
- Only rows that are meant to be processed have their primary key appear in the control table. You can use information that is not in the rows themselves to control which rows to process (a good `WHERE` clause may not be enough).
- The entire row is not copied to a control table, and any structure under the source table, such as detail rows, can also be raised without copying.

Streams and materialized view logs make excellent control tables.

Preconditions: You must have the create/alter triggers privilege on the source table. Table 9-11 describes the requirements for using the control table polling strategy.

Table 9-11 Control Table Polling Strategy Preconditions

Requirements Met	Conflicts With
Poll for inserts	Advanced configuration: the native XML from the database has control header, and triggers are required.
Poll for updates	--
Poll for deletes	--
Poll for child updates	Minimal data replication (primary keys are stored in control table)
No delete on source	--
No updates on source	--
No extra SQL selects	--
Concurrent polling	--
Allows raw SQL	--
Auditing	--

Using triggers, whenever a row is modified, an entry is added to a control table, containing the name of the master table, and the primary keys. At design time, the control table is defined to be the root table, with a one-to-one mapping to the master table, based on the matching primary keys. The control table can contain extra control information, such as a time stamp, and operation type (`INSERT`, `UPDATE`, and so on).

The delete polling strategy is useful with this setup. It is important to keep the control table small, and if the option `shouldDeleteDetailRows="false"` is used, then only the control rows are deleted, giving you a nondestructive delete (the `DELETE` is not cascaded to the real tables).

It is possible to reuse the same control table for multiple master tables. In TopLink, you can map the same table to multiple descriptors by mapping the control table as one abstract class with multiple children. Each child has a unique one-to-one mapping to a different master table. The advantage of this approach is that you can specify for each child a class indicator field and value so that you do not require an explicit `WHERE` clause for each polling query.

The following are sample triggers for polling for changes both to a department table and any of its child employee rows:

```
CREATE OR REPLACE TRIGGER EVENT_ON_DEPT
   AFTER INSERT OR UPDATE ON DEPARTMENT
   REFERENCING NEW AS newRow
   FOR EACH ROW
   DECLARE X NUMBER;
BEGIN
   SELECT COUNT(*) INTO X FROM DEPT_CONTROL WHERE (DEPTNO = :newRow.DEPTNO);
   IF X = 0 then
   insert into DEPT_CONTROL values (:newRow. DEPTNO);
   END IF;
END;
CREATE OR REPLACE TRIGGER EVENT_ON_EMPLOYEE
   AFTER INSERT OR UPDATE ON EMPLOYEE
   REFERENCING OLD AS oldRow NEW AS newRow
   FOR EACH ROW
   DECLARE X NUMBER;
BEGIN
   SELECT COUNT(*) INTO X FROM DEPT_CONTROL WHERE (DEPTNO = :newRow.DEPTNO);
   IF X = 0 then
   INSERT INTO DEPT_CONTROL VALUES (:newRow.DEPTNO);
   END IF;
   IF (:oldRow.DEPTNO <> :newRow.DEPTNO) THEN
      SELECT COUNT(*) INTO X FROM DEPT_CONTROL WHERE (DEPTNO = :oldRow.DEPTNO);
      IF (X = 0) THEN
         INSERT INTO DEPT_CONTROL VALUES (:oldRow.DEPTNO);
      END IF;
   END IF;
END;
```

Database Adapter Deployment

The Oracle Database Adapter comes deployed to the application server by the install. It contains a single adapter instance entry `eis/DB/SOADemo`, which points to the data source `jdbc/SOADataSource`. The connection information to the database is inside the data source definition.

When deploying a SOA project that uses the OracleAS Adapter for Databases, you might have to add a new adapter instance and restart the application server first. This could be because you want to point to a database other than the one referred in `jdbc/SOADataSource`, or because you chose a name for the adapter instance that does not yet exist. For instance, if you create a connection in JDeveloper named `Connection1`, then by default the DB Adapter service points to `eis/DB/Connection1`, as shown in Figure 9-7.

You can also check which adapter instance the service is pointing to by looking at the `db.jca` file, as shown in the following code snippet:

```
<connection-factory location="eis/DB/Connection1" UIConnectionName="Connection1"
adapterRef="" />
```

In the preceding example, the location is the JNDI name of the adapter instance at runtime, and `UIConnectionName` is the name of the connection used in JDeveloper.

You can create a Database Adapter instance through the Oracle WebLogic Server Administration Console, as mentioned in Adding an Adapter Connection Factory or by directly editing the `weblogic-ra.xml` file. Following these steps are screenshots that show how to create an adapter instance through the Oracle WebLogic Administration Console. The following are the steps to edit `weblogic-ra.xml`:

1. Search `fmwhome/` for `DbAdapter.rar`.
2. Unzip the file.
3. Edit `META-INF/weblogic-ra.xml` (and possibly `ra.xml`.)
4. Jar the file again.
5. Restart the application server.

The following example is a sample adapter instance in `weblogic-ra.xml`.

Example - Sample Adapter Instance in weblogic-ra.xml

```
<connection-instance>
  <jndi-name>eis/DB/SOADemo</jndi-name>
    <connection-properties>
       <properties>
          <property>
            <name>xADataSourceName</name>
            <value>jdbc/SOADataSource</value>
              </property>
              <property>
                <name>dataSourceName</name>
                <value> </value>
              </property>
              <property>
                <name>platformClassName</name>
                <value>Oracle10Platform</value>
              </property>
           </properties>
        </connection-properties>
</connection-instance>
```

The four mandatory properties are: `jndi-name`, `xADataSourceName`, `dataSourceName`, and `platformClassName`. The `jndi-name` property must match the location attribute in the `db.jca` file, and is the name of the adapter instance.

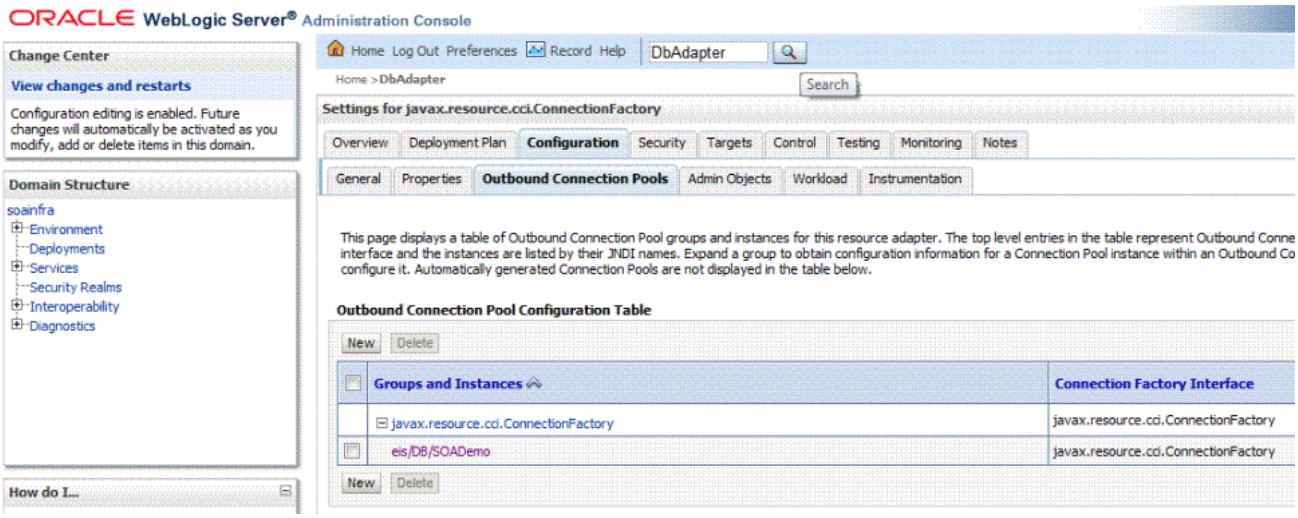
The `xADataSourceName` property is the name of the underlying data source (which has the connection information).

The `platformClassName` indicates which SQL to generate. For information about `PlatformClassName`, see Table 9-13.

The following screenshots show how to edit Database Adapter properties using the Oracle WebLogic Administration Console

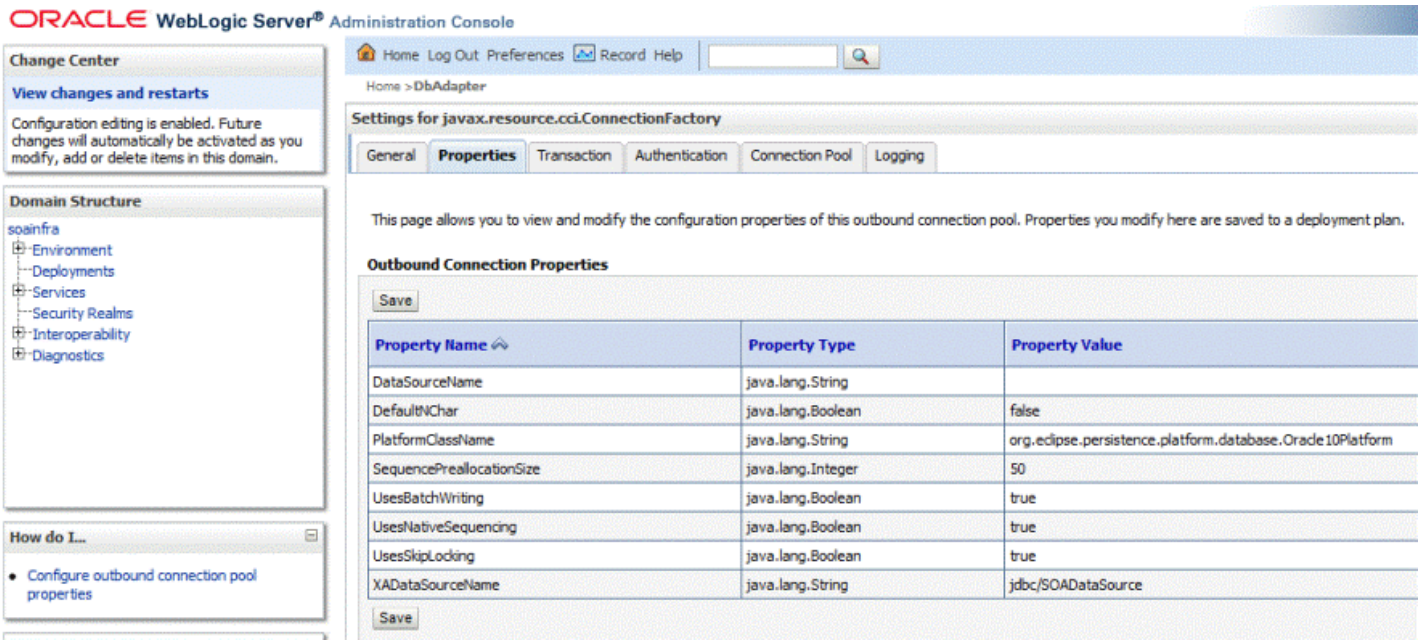
The first screenshot shows navigation to the Outbound Connection Pools within the WebLogic Administration Console. This is the actual Database Adapter Configuration, from where you can go to the subsequent page to edit the Database Adapter properties.

Figure 9-30 The Outbound Connection Pools Tab of the WebLogic Console



The second screenshot shows editing properties from the WebLogic Console that you edit accordingly and as required. Name, Type and Value are displayed on a per-property basis.

Figure 9-31 Database Adapter Properties in the Oracle WebLogic Administration Console



Most Common Mistakes

The following are the two most common mistakes with deployment:

- Not creating an adapter instance entry that matches the location attribute in your `db.jca` file (or not creating one at all.)

- Setting the location attribute in the `db.jca` file to the name of the data source directly.
- Not changing `platformClassName` when connecting to a database other than Oracle.

For the latter, there is a level of indirection in that you give the name of the adapter instance (`eis/DB/...`), which itself points to the data source pool (`jdbc/...`). It is a common mistake to miss this indirection and give the name `jdbc/...` directly in the location attribute.

Data Source Configuration

For the relevant Data Source configuration for your application server, see JDBC Driver and Database Connection Configuration. When configuring an Oracle data source, ensure that you use the `thin XA` option.

Additional Adapter Instance Properties

This section briefly describes additional properties in the Database Adapter instance beyond `xADataSourceName`, `dataSourceName`, and `platformClassName`.

When adding a property to `weblogic-ra.xml`, you must ensure that the property is also declared in `ra.xml` (also in `DbAdapter.rar`). For example, the following is a code snippet of the `ra.xml` file for the property `xADataSourceName` in `weblogic-ra.xml`:

```
<config-property>
<config-property-name>xADataSourceName </config-property-name>
<config-property-type>java.lang.String</config-property-type>
<config-property-value></config-property-value>
</config-property>
```

For information about the Oracle Database Adapter instance properties, see Oracle Database Adapter Properties.

Apart from the properties mentioned there, you can also add the properties listed inTable 9-12:

Table 9-12 Oracle Database Adapter Properties Listed in Toplink DatabaseLogin Object

Property Name	Type
`usesNativeSequencing`	Boolean
`usesSkipLocking`	Boolean
`usesStringBinding`	Boolean
`usesByteArrayBinding`	Boolean
`usesStreamsForBinding`	Boolean
`eventListenerClass`	String
`logTopLinkAll`	Boolean
`maxBatchWritingSize`	Integer
`nonRetriableSQLErrorCodes`	String

Table 9-12 (Cont.) Oracle Database Adapter Properties Listed in Toplink DatabaseLogin Object

Property Name	Type
`shouldOptimizeDataConversion`	Boolean
`shouldTrimStrings`	Boolean
`driverClassName`	String
`sequencePreallocationSize`	Integer
`tableQualifier`	String
`usesBatchWriting`	Boolean

The preceding properties appear in the `oracle.toplink.sessions.DatabaseLogin` object.

See `TopLink` API reference information on `DBConnectionFactory` Javadoc and `DatabaseLogin` Javadoc at `http://download.oracle.com/docs/cd/B10464_02/web.904/b10491/index.html`.

Deployment with Third-Party Databases

Table 9-13 lists databases and their advanced properties, which are database platform variables.

Set the `platformClassName` name to a listed variable. Setting `platformClassName` is mandatory if you are using an advanced database.features that are not uniform across databases, such as native sequencing or stored procedures.

As an example, to execute a stored procedure on DB2 versus SQL Server, the DbAdapter must generate and send different SQL. Use this example for use with the SQLServer Platform:

```
execute <procedure> @<arg1>=? ...
```

when using the DB2 Platform:

```
call <procedure>(?, ...)
```

The `platformClassName` setting indicates which SQL to generate. Since most databases offer non-uniform features (that is, variants on the ANSI SQL 92 language specification), it is safest to configure `platformClassName` accurately.The default value is `Oracle10Platform`, and should be changed to the appropriate variable if you are connecting to a different database vendor.

Note:

Providing the qualified class name with package is not necessary if it starts with org.eclipse.persistence.platform.database

Table 9-13 Database Platform Names

Database	PlatformClassName
Oracle10+ (including 11*g*)	`org.eclipse.persistence.platform.database.Oracle10Platform`
Oracle9+ (optional)	`org.eclipse.persistence.platform.database.Oracle9Platform`
DB2	`org.eclipse.persistence.platform.database.DB2Platform`
DB2 on AS400e	`oracle.tip.adapter.db.toplinkext.DB2AS400Platform`
Informix	`org.eclipse.persistence.platform.database.InformixPlatform`
SQLServer	`org.eclipse.persistence.platform.database.SQLServerPlatform`
MySQL	`org.eclipse.persistence.platform.database.MySQLPlatform`
Derby	`org.eclipse.persistence.platform.database.DerbyPlatform`
DB2 for z/OS	`org.eclipse.persistence.platform.database.DB2MainframePlatform`
Any other database	`org.eclipse.persistence.platform.database.DatabasePlatform`

JDBC Driver and Database Connection Configuration

In this release, Oracle JCA Adapters are certified against the following third-party databases using Oracle WebLogic Server Type 4 JDBC drivers:

- Microsoft SQL Server 2005 and 2008 (all SP levels included)
- Sybase 15
- Informix 11.5
- DB2 9.7 and later FixPaks
- MySQL 5.x+

Note:

Only major databases and versions are certified. Working with other databases should be feasible when they provide a working JDBC driver, and you rely on core ANSI SQL relational features, such as Create, Read, Update, and Delete (CRUD) operations on tables and views. Issues tend to be more prevalent due to the fact that not all JDBC drivers implement database metadata introspection the same way. However, it should be possible to import matching tables on a certified database and then point to the uncertified database at runtime. The information provided in this section for uncertified databases is meant as a guide only.

For more information, see the following topics:

- Creating a Database Connection Using a Native or Bundled JDBC Driver

Creating a Database Connection Using a Native or Bundled Oracle WebLogic Server JDBC Driver

To create a database connection when using a native or bundled Oracle WebLogic Server JDBC driver:

1. Install the appropriate JDBC driver JAR files and set the class path.

   For more information, see Location of JDBC Driver JAR Files and Setting the Class Path.

2. In the **File** menu, click **New**.

   The New Gallery page is displayed.

3. In the **All Technologies** tab, under **General** categories, select **Connections**.

   A list of the different connections that you can make is displayed in the Items pane on the right side of the New Gallery page.

4. Select **Database Connection**, and then click **OK**.

   The Create Database Connection page is displayed.

5. For **Create Connection In**, select **IDE Connections**.

6. Enter a name for this connection in the Connection Name field.

   For example, `SQLServer`.

7. Select the appropriate driver from the **Connection Type** menu.

8. Enter your credentials (such as user name, password, and role, if applicable).

9. Enter your connection information.

   For example, `jdbc:sqlserver://HOST-NAME:PORT;databaseName=DATABASE-NAME`

   For more information, see:

   - Table 9-14
   - Sample entries in the deployment descriptor file (`weblogic-ra.xml`).

10. Click **Test Connection**.

11. If the connection is successful, click **OK**.

Creating a Database Connection Using a Third-Party JDBC Driver

To create a database connection when using a third-party JDBC driver:

1. Install the appropriate JDBC driver JAR files and set the class path.

For more information, see Location of JDBC Driver JAR Files and Setting the Class Path.

2. In the **File** menu, click **New**.

   The New Gallery page is displayed.

3. In the **All Technologies** tab, under **General** categories, select **Connections**.

   A list of the different connections that you can make is displayed in the Items pane on the right side of the New Gallery page.

4. Select **Database Connection**, and then click **OK**.

   The Create Database Connection page is displayed.

5. For **Create Connection In**, select **IDE Connections**.

6. Enter a name for this connection in the Connection Name field.

   For example, `SQLServer`.

7. Select **Generic JDBC** from Connection Type.

8. Enter your user name, password, and role information.

9. Click **New** for Driver Class.

   The Register JDBC Driver dialog is displayed.

   Perform Steps 10, 11 and 19 in the Register JDBC Driver dialog.

10. Enter the driver name (for example, *`some`*`.jdbc.`*`Driver`*) for **Driver Class**.

    For example, `com.microsoft.sqlserver.jdbc.SQLServerDriver`.

11. Click **Browse** for Library.

    The Select Library dialog is displayed.

    Perform Steps 12 and 18 in the Select Library dialog.

12. Click **New** to create a library.

    The Create Library dialog is displayed.

    Perform Steps 13 through 17 in the Create Library dialog.

13. Specify a name in the **Library Name** field.

    For example, `SQL Server JDBC`.

14. Click **Class Path**, and then click **Add Entry** to add each JAR file of your driver to the class path.

    The Select Path Entry dialog is displayed.

15. Select a JDBC class file and click **Select**.

    For example, select `sqljdbc.jar`.

16. Click **OK** when you have added all the class files to the Class Path field.

17. Click **OK** to exit the Create Library dialog.
18. Click **OK** to exit the Select Library dialog.
19. Click **OK** to exit the Register JDBC Driver dialog.
20. Enter your connection string name for **JDBC URL** and click **Next**.

    For example, `jdbc:sqlserver://HOST-NAME:PORT;databaseName=DATABASE-NAME`

    For more information, see:

    - Table 9-14
    - Sample entries in the deployment descriptor file (`weblogic-ra.xml`).
21. Click **Test Connection**.
22. If the connection is successful, click **OK**.

Summary of Third-Party JDBC Driver and Database Connection Information

Table 9-14 summarizes the connection information for common third-party databases.

Note that driver files for Oracle, IBM DB2, Informix, Sybase, SQLServer, MySQL and Derby are bundled with the SOA Install.

Oracle has certified these WebLogic Server drivers (Oracle drivers) for these databases. For other Databases, including DB2/AS400, DB2/ZOS and ProgressDB, you must obtain the appropriate native drivers.

For information about `PlatformClassName`, see Table 9-13.

For more information, see:

- Using a Microsoft SQL Server
- Using a Sybase Database
- Using an Informix Database
- Using an IBM DB2 Database
- Using a MySQL Database

Table 9-14 Database Driver Selection (from Weblogic Server Console)

Database	JDBC Driver
Microsoft SQL Server	• Oracle's MS SQL Server Driver (Type 4 XA) • Oracle's MS SQL Server Driver (Type 4)
Sybase	• Oracle's Sybase Driver (Type 4 XA) • Oracle's Sybase Driver (Type 4)
Informix	• Oracle's Informix Driver (Type 4 XA) • Oracle's Informix Driver (Type 4)
IBM DB2	• Oracle's DB2 Driver (Type 4 XA) • Oracle's DB2 Driver (Type 4)

Table 9-14 (Cont.) Database Driver Selection (from Weblogic Server Console)

Database	JDBC Driver
MySQL	MySQL's Driver (Type 4) Versions: using `com.mysql.jdbc.Driver`
Derby	Derby's Driver(Type 4 XA)
DB2/AS400	Download driver jt400Native.jar online and copy it under user_projects/domains/<domain_name>/lib/ NOTE: Choose `other` option while creating jndi from WLS
DB2/ ZOS	Download driver `db2jcc4.jar`,`db2jcc_license_cu.jar` and `db2jcc_license_cisuz.jar` online and copy it under `user_projects/domains/<domain_name>/lib/` NOTE: Choose `other` option while creating JNDI from WebLogic Server.
ProgressDb	Copy drivers `openedge.jar`, `pool.jar` and `schema.jar` from Progress install and copy it under `user_projects/domains/<domain_name>/lib/` NOTE: Choose `other` option while creating JNDI from WebLogic Server.

Using a Microsoft SQL Server

You must note the following when connecting to a SQL Server database:

To enable XA transaction:

1. Copy all jll files from `<install_dir>/oracle_common/modules/datadirect` to `Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Binn` of your MSSQL installation.
2. Copy instjdbc.sql from `<Install_dir>/oracle_common/modules/datadirect` to MSSQL installation machine, for example to the c directory (`c:/`).
3. Copy the DLL you require, based on your operating system to the SQL Server Bin directory and rename to `sqljdbc.dll`. That is,
   - `sqljdbc.dll` - for 32 bit OS
   - `64sqljdbc.dll` - for 64 bit OS Itanium based
   - `x64sqljdbc.dll` - for 64 bit OS Intel based
4. Run the following command from the SQL command line. The command should complete without errors.

   ```
   sqlcmd -Usa -Pwelcome1! -Slocalhost -iC:\instjdbc.sql
   ```

5. Restart SQL server services.

Note that for an XA transaction to run successfully, you must also do the following:

1. Proceed to **run -> dcomcnfg**
2. Then, proceed to **component servies->Mycomp->DTC->LocalDTC->Properties** (and right-click **Properties**.)

3. Then, proceed to **Security->** and Select **Enable xa transaction**
4. Finally, restart all related services (`trxn`, `sqlserver`)

Using a SQLSERVER Weblogic JDBC Driver

URL: `jdbc:weblogic:sqlserver://<host>:<port>`

Driver Class: `weblogic.jdbcx.sqlserver.SQLServerDataSource`

Driver Jar: `wlsqlserver.jar` is bundled with the WebLogic Server and is provided with your installation.

Its location is `/<install_dir>/oracle_common/modules/datadirect`

Using a Sybase Database

This section includes the following topics:

- Using a Sybase JConnect JDBC Driver

Using a Sybase JConnect JDBC Driver

URL: `jdbc:weblogic:sybase://<host>:<port>`

Driver Class: `weblogic.jdbcx.sybase.SybaseDataSource`

Driver Jar: `wlsybase.jar` is bundled with the WebLogic Server and is provided with your installation.

Its location is `/<install_dir>/oracle_common/modules/datadirect`

For information about the Sybase JConnect JDBC driver, refer to the following link:

`http://www.sybase.com/products/middleware/jconnectforjdbc`

Using an Informix Database

This section includes the following topics:

- Using an Informix JDBC Driver

Using an Informix JDBC Driver

URL: jdbc:weblogic:informix://
<host>:<port>;informixServer=<server_name>;databaseName=<db_name>

Driver Class: weblogic.jdbcx.informix.InformixDataSource

Driver Jar: `wlinformix.jar` is bundled in Server. and is provided with installation. Its location is `/<install_dir>/oracle_common/modules/datadirect`

For information about the Informix JDBC driver, refer to the following link:

`http://www-01.ibm.com/software/data/informix/tools/jdbc/`

Using an IBM DB2 Database

This section includes the following topics:

- IBM DB2 Driver
- JT400 Driver (AS400 DB2)

- IBM Universal Driver

IBM DB2 Driver

URL: jdbc:weblogic:db2://<host>:<port>Driver Class: weblogic.jdbcx.db2.DB2DataSourceDriver Jar : `wldb2.jar` is bundled in Server. Provided with Install. Location- `/<install_dir>/oracle_common/modules/datadirect`

For information about DataDirect driver, refer to the following link:

`http://www.datadirect.com/techres/jdbcproddoc/index.ssp`

JT400 Driver (AS400 DB2)

URL: `jdbc:as400://hostname;translate binary=true`

Driver Class: `com.ibm.as400.access.AS400JDBCDriver/`

`com.ibm.as400.access.AS400JDBCXADataSource`

Driver Jar: `jt400.jar`/jt400Native.jar

For correct character set translation, use `translate binary=true`.

IBM Universal Driver

(ZOS)

URL: `jdbc:db2://hostname:port/`schemaname

Driver Class: `weblogic.jdbcx.db2.DB2DataSource`

Driver Jar: wldb2.jar is bundled in Server. Provided with Install. Location- /<install_dir>/oracle_common/modules/datadirect

Using a MySQL Database

Use the following information:

URL: `jdbc:mysql://hostname:3306/dbname`

Driver Class: `com.mysql.jdbc.Driver` / `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource`

Driver Jar: `mysql-connector-java-<version_number>-bin.jar`

Using a Derby Database

Use the following information:

URL: `jdbc:derby://<host>:<port><location of schema>`

Driver Class: `org.apache.derby.jdbc.ClientXADataSource`

Driver Jar: `Derby.jar, derbyclient.jar`

Using a Progress Database

Use the following information:

URL: `jdbc:datadirect:openedge://<host>:<port>;databaseName=<dbname>`

Driver Class: `com.ddtek.jdbc.openedge.OpenEdgeDriver`

Driver Jar: `openedge.jar, pool.jar, schema.jar`

Location of JDBC Driver JAR Files and Setting the Class Path

This section describes the location of JDBC JAR files and setting the class path at runtime and design time.

Run Time

For both Windows and Linux, you must perform the following steps:

1. Drop the vendor-specific driver JAR files to the `user_projects/domains/soainfra/lib` directory.
2. Drop the vendor-specific driver JAR files to the `<Weblogic_Home>/server/lib`.
3. Edit the classpath to include the vendor-specific jar file in `<Weblogic_HOME>/common/bin/commEnv.sh`

Design Time

For both Windows and Linux, drop the JDBC JAR to the `Oracle/Middleware/jdeveloper/jdev/lib/patches` directory.

Stored Procedure and Function Support

This section describes how the Oracle Database Adapter supports the use of stored procedures and functions.

This section includes the following topics:

- Design Time: Using the Adapter Configuration Wizard
- Supported Third-Party Databases
- Design Time: Artifact Generation
- Run Time: Before Stored Procedure Invocation
- Run Time: After Stored Procedure Invocation
- Run Time: Common Third-Party Database Functionality
- Advanced Topics

Design Time: Using the Adapter Configuration Wizard

The Adapter Configuration Wizard – Stored Procedures is used to generate an adapter service WSDL and the necessary XSD. The adapter service WSDL encapsulates the underlying stored procedure or function as a Web service with a WSIF JCA binding. The XSD file describes the procedure or function, including all the parameters and their types. This XSD provides the definition used to create instance XML that is submitted to the Oracle Database Adapter at run time.

This section includes the following topics:

- Using Top-Level Standalone APIs
- Using Packaged APIs and Overloading

Using Top-Level Standalone APIs

This section describes how to use the Adapter Configuration Wizard with APIs that are not defined in PL/SQL packages. You use the Adapter Configuration Wizard – Stored Procedures to select a procedure or function and generate the XSD file. See Use Cases if you are not familiar with how to start the Adapter Configuration Wizard.

The following are the steps to select a stored procedure or function by using the Adapter Configuration Wizard:

1. Drag and drop **Database Adapter** from the Service Adapters list to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard is displayed, as shown in Figure 9-32.

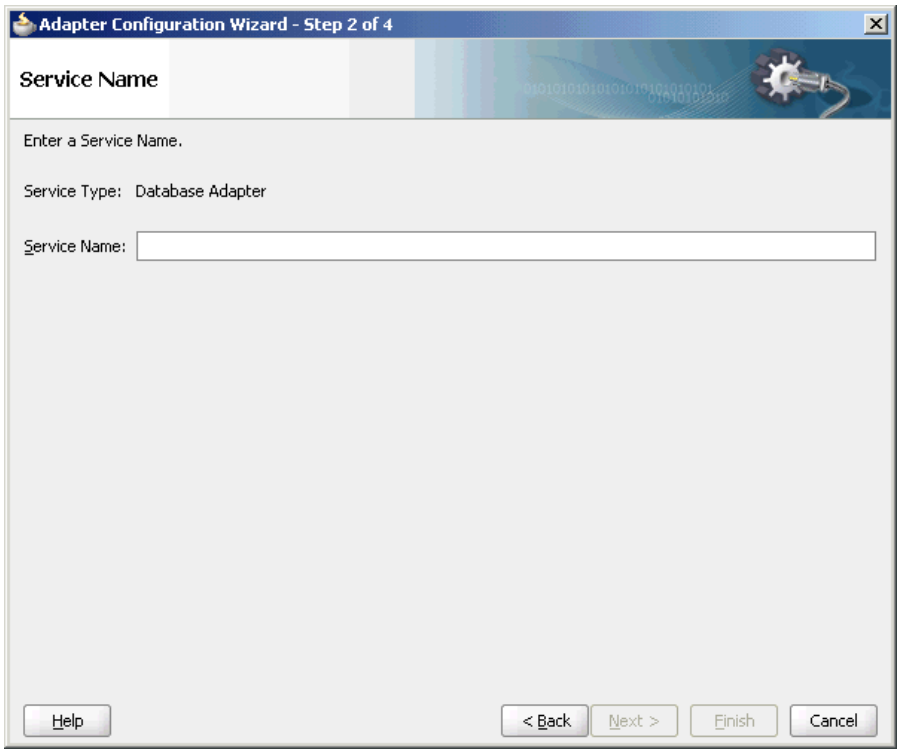
   Figure 9-32 The Adapter Configuration Wizard

2. Click **Next**. The Service Name page is displayed, as shown in Figure 9-33.

Note:

Note that the name of stored procedures or packages that refers to database or user-defined data types must not include the character $ in it. The presence of $ in the name causes the XSD file generation to fail.

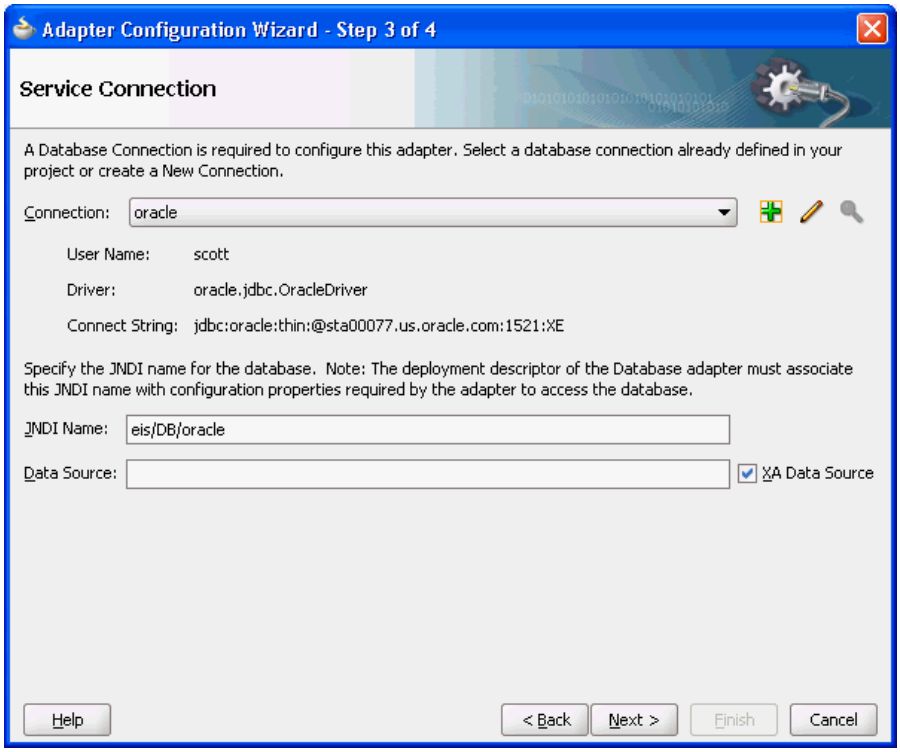
Figure 9-33 Specifying the Service Name



3. In the **Service Name** field, enter a service name, and then click **Next**. The Service Connection page is displayed.

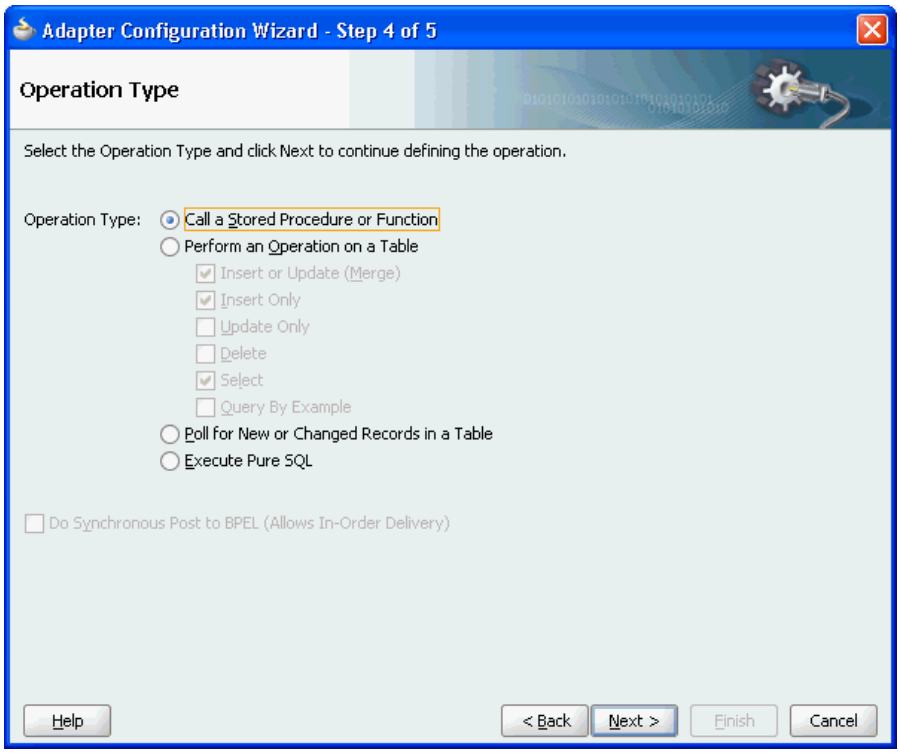
   You associate a connection with the service, as shown in Figure 9-34. A database connection is required to configure the adapter service. Select an existing connection from the list or create a new connection.

Figure 9-34 Setting the Database Connection in the Adapter Configuration Wizard



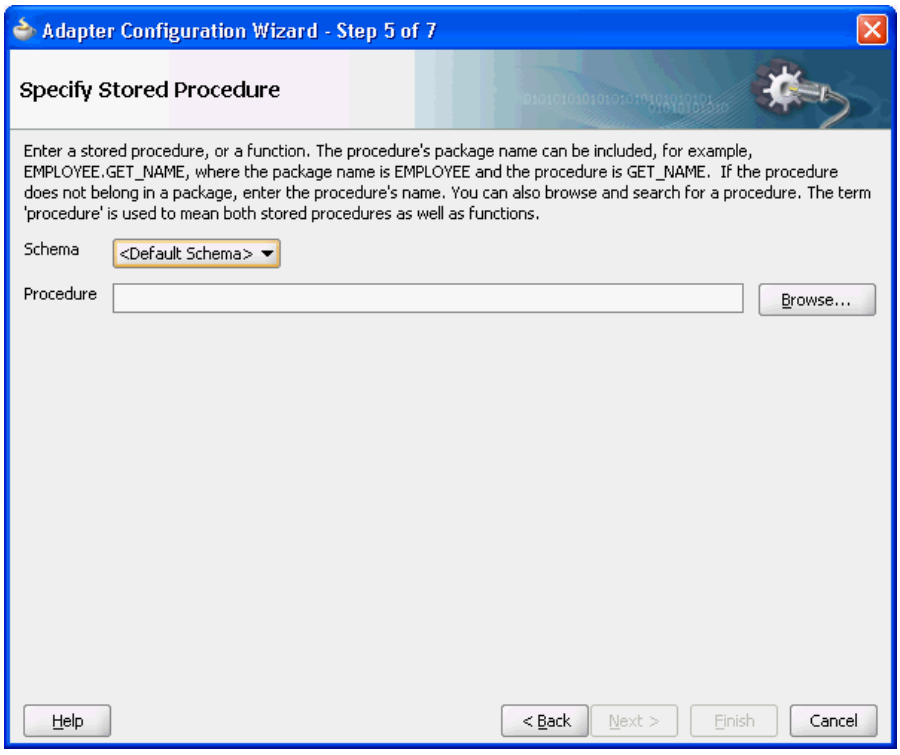
4. Click **Next**. The Operation Type page is displayed.
5. For the **Operation Type**, select **Call a Stored Procedure or Function**, as shown in Figure 9-35.

Figure 9-35 Calling a Stored Procedure or Function in the Adapter Configuration Wizard



6. Click **Next**. The Specify Stored Procedure page is displayed, as shown in Figure 9-36. This is where you specify a stored procedure or function.

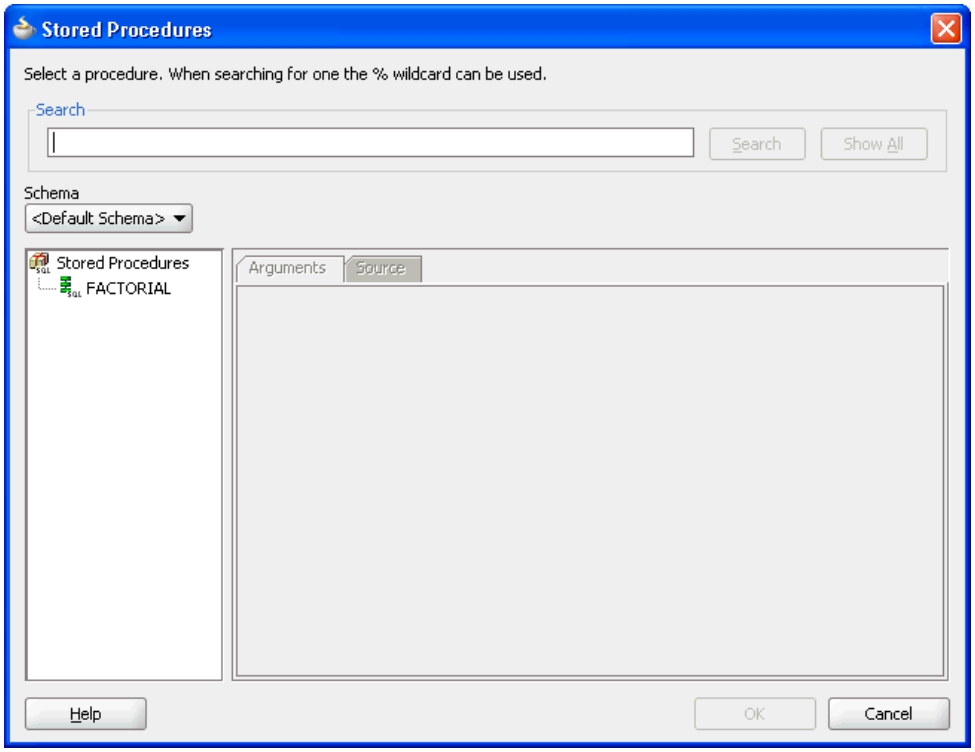
Figure 9-36 The Specify Stored Procedure Page



7. Next, you select the schema and procedure or function. You can select a schema from the list or select **<Default Schema>**, in which case the schema associated with the connection is used. If you know the procedure name, enter it in the Procedure field. If the procedure is defined inside a package, you must include the package name, as in `EMPLOYEE.GET_NAME`.

   If you do not know the schema and procedure names, click **Browse** to access the **Stored Procedures** window, as shown in Figure 9-37.

Figure 9-37 Searching for a Procedure or Function



Select a schema from the list or select **<Default Schema>**. A list of the available procedures is displayed in the left window. To search for a particular API in a long list of APIs, enter search criteria in the **Search** field. For example, to find all APIs that begin with `XX`, enter `XX%` and click the **Search** button. Clicking the **Show All** button displays all available APIs.

Figure 9-38 shows how you can select the FACTORIAL function. The Arguments tab displays the parameters of the function, including their names, type, mode (`IN`, `IN/OUT` or `OUT`) and the numeric position of the parameter in the definition of the procedure. The return value of a function has no name and is always an `OUT` parameter at position zero (0).

Figure 9-38 Viewing the Arguments of a Selected Procedure

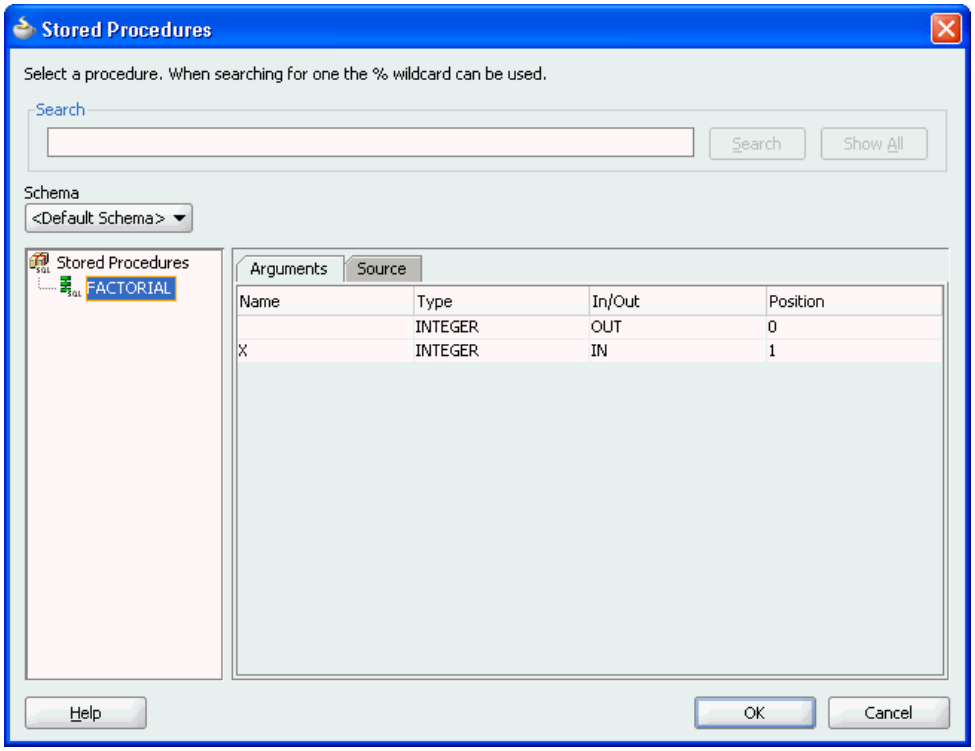
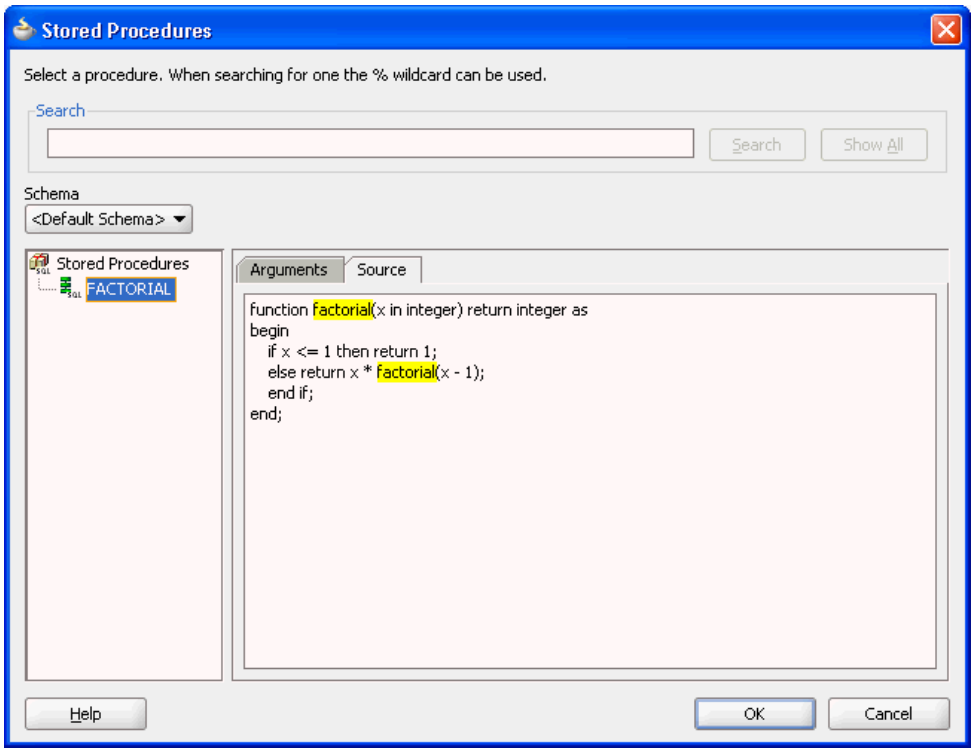


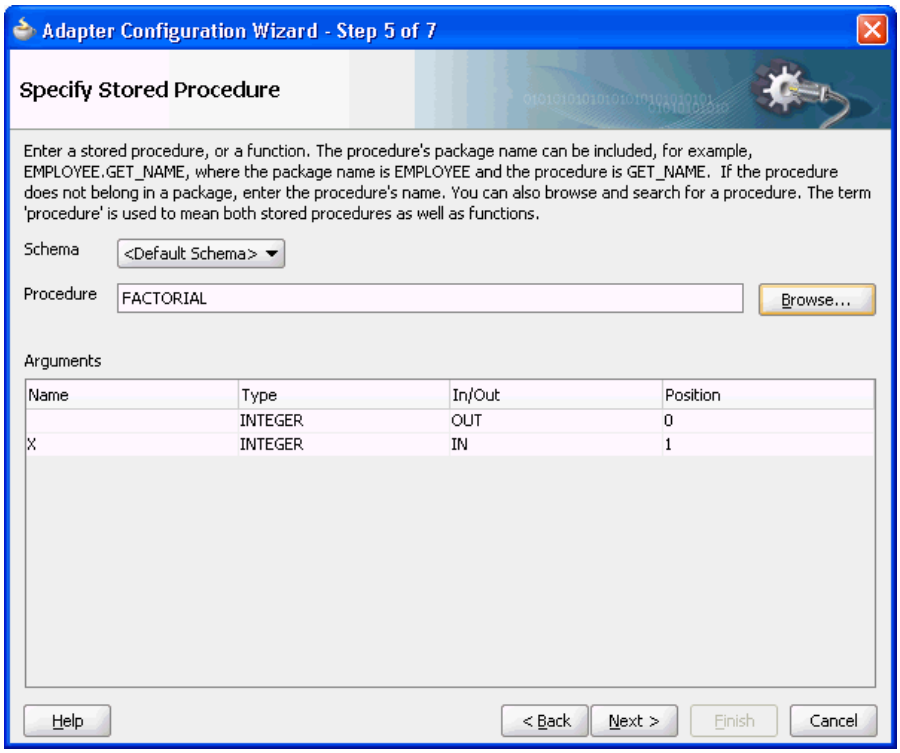
Figure 9-39 shows how the Source tab displays the code that implements the function. Text that matches the name of the function is highlighted.

Figure 9-39 Viewing the Source Code of a Selected Procedure



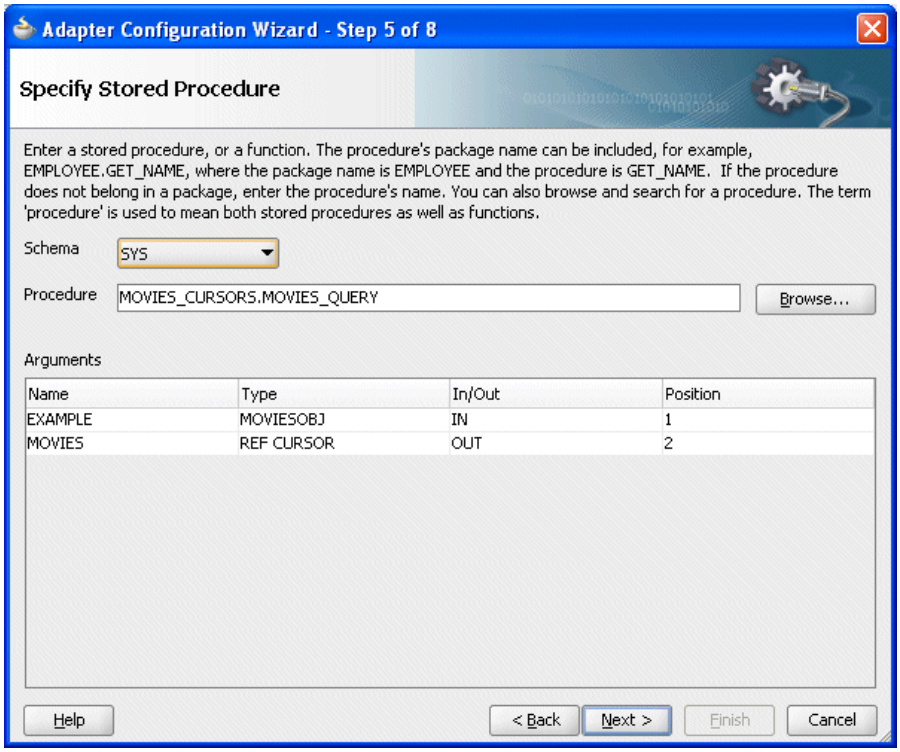
8. Click **OK** after selecting a procedure or function. Information about the API is displayed, as shown in Figure 9-40. Click **Back** or **Browse** to make revisions.

Figure 9-40 Viewing Procedure or Function Details in the Adapter Configuration Wizard



9. Click **Next**. If the stored procedure or function has an output parameter of type row set (`REF CURSOR` on Oracle Database), as Figure 9-41 shows, you can define a strongly or weakly typed XSD for this ref cursor.

Figure 9-41 Viewing Procedure or Function Details in the Adapter Configuration Wizard: Row Set Type

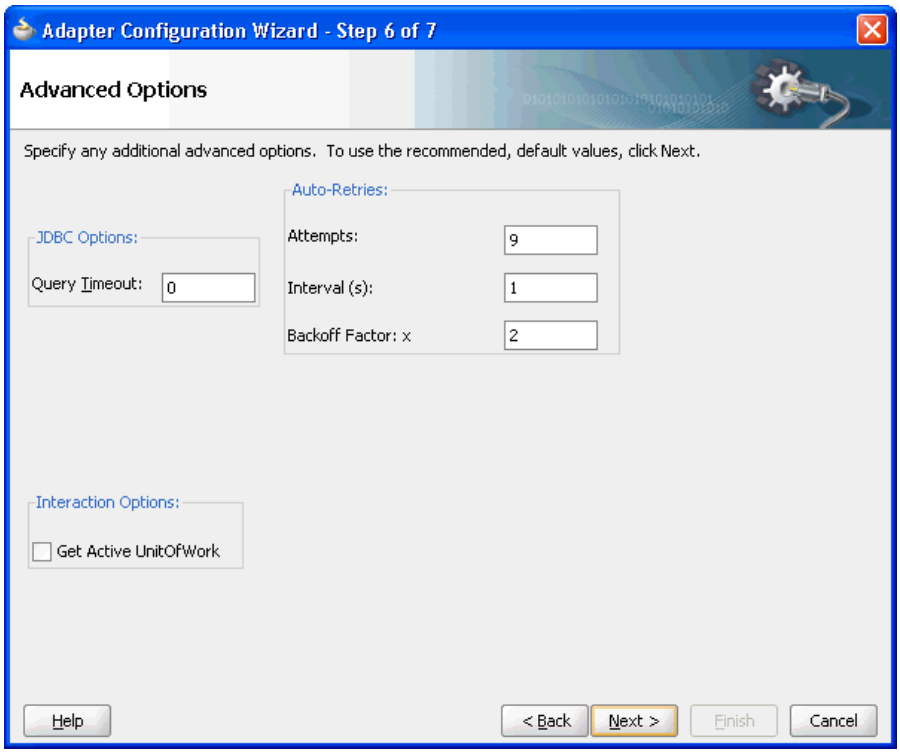


For more information, see:

- Row Set Support Using a Strongly Typed XSD
- Row Set Support Using a Weakly Typed XSD

10. Click **Next**. The **Advanced Options** page is displayed, as shown in Figure 9-42. Enter any advanced options, such as the JDBC `QueryTimeout` value. Other options include retry parameters, such as the number of retry attempts and the interval between them.

Figure 9-42 The Advanced Options Page



11. After specifying all options, click **Next**, and then click **Finish** to complete the Adapter Configuration Wizard.

    When you have finished using the Adapter Configuration Wizard, three files are added to the existing project:

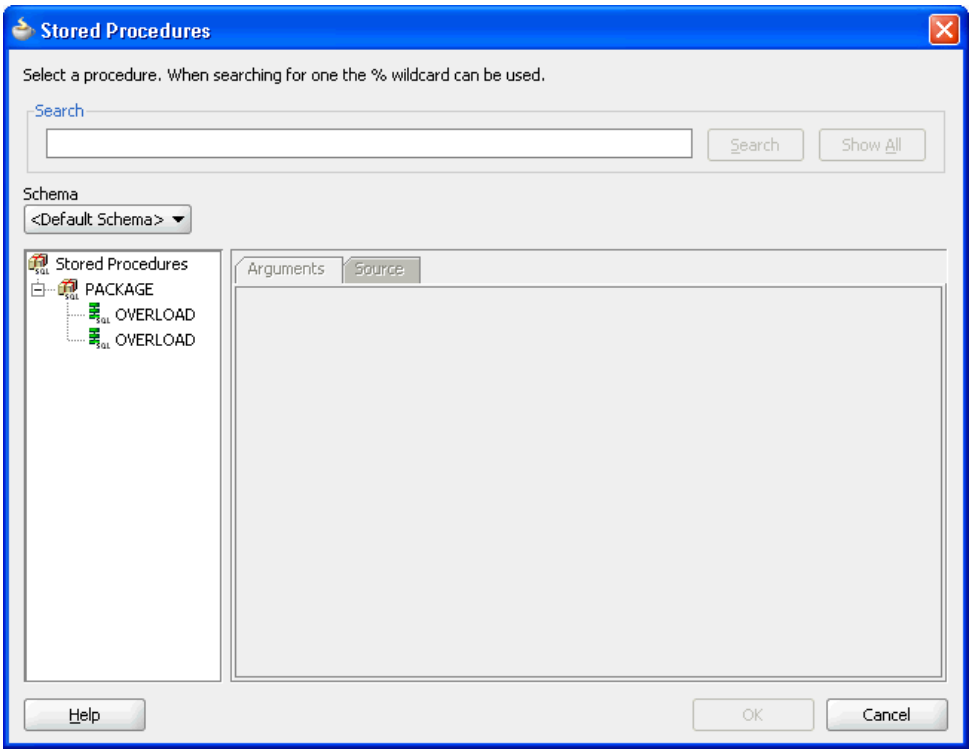
    - *servicename*`.wsdl` (for example, `Factorial.wsdl`)
    - *service_name_db*`.jca` (for example, `Factorial_db.jca`)
    - *schema_package_procedurename*`.xsd` (for example, `SCOTT_FACTORIAL.xsd`)

Using Packaged APIs and Overloading

Using APIs defined in packages is similar to using standalone APIs. The only difference is that you can expand the package name to see a list of all the APIs defined within the package, as shown in Figure 9-43.

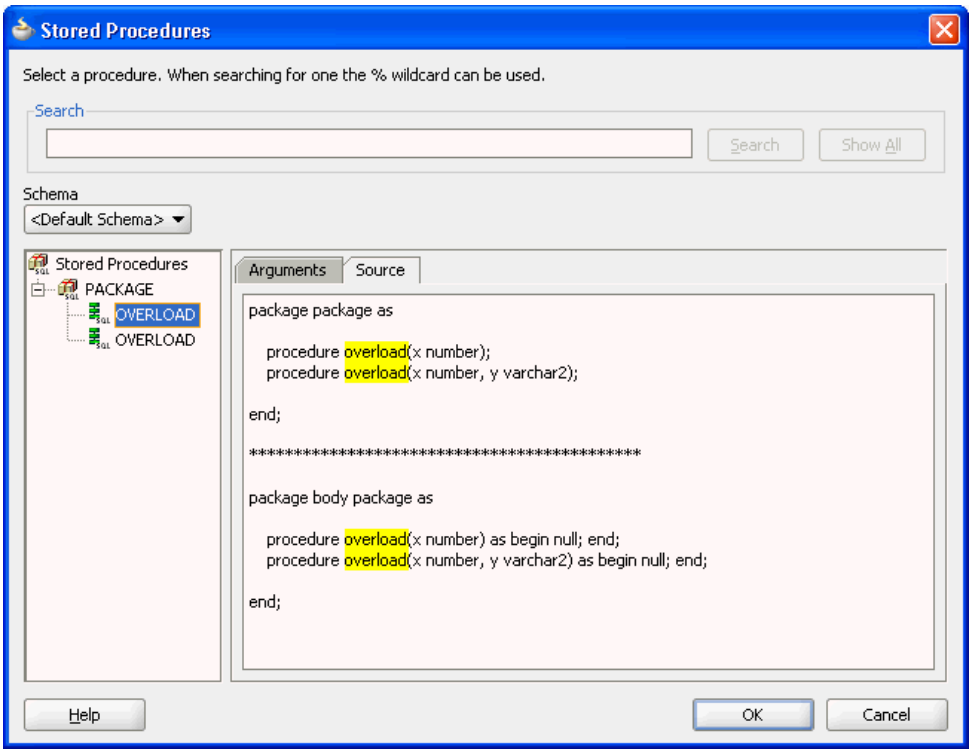
APIs that have the same name but different parameters are called overloaded APIs. As shown in Figure 9-43, the package called **PACKAGE** has two overloaded procedures called **OVERLOAD**.

Figure 9-43 A Package with Two Overloaded Procedures



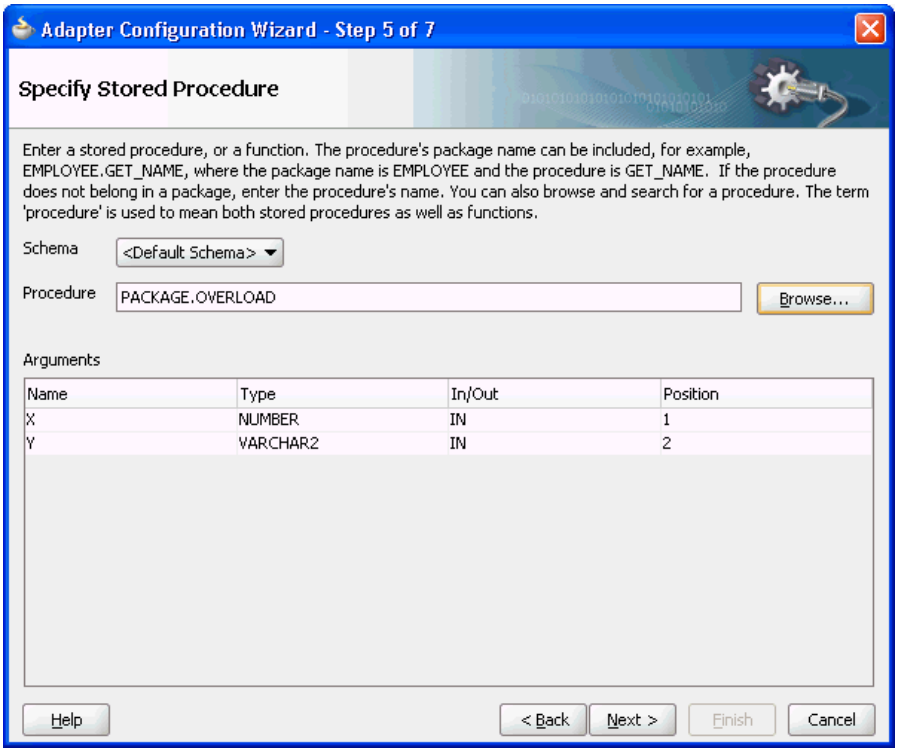
As Figure 9-44 shows, the code for the entire PL/SQL package is displayed, regardless of which API from the package is selected when you view the Source tab. Text that matches the name of the procedure is highlighted.

Figure 9-44 Viewing the Source Code of an Overloaded Procedure



After you select a procedure or function and click **OK**, information about the API is displayed, as shown in Figure 9-45. The schema, procedure name, and a list of arguments are displayed. Note how the procedure name is qualified with the name of the package (**PACKAGE.OVERLOAD**). Click **Back** or **Browse** to make revisions, or **Next**. Enter values for any of the advanced options. Click **Next** followed by **Finish** to conclude.

Figure 9-45 Viewing Procedure or Function Details in the Adapter Configuration Wizard



When you have finished using the Adapter Configuration Wizard, the following files are added to the existing project:

- `Overload.wsdl`, `Overload_db.jca`
- `SCOTT_PACKAGE_OVERLOAD_2.xsd`.

  The _2 appended after the name of the procedure in the XSD filename differentiates the overloaded APIs. Numeric indexes are used to differentiate between overloaded APIs.

Supported Third-Party Databases

For stored procedures, the following databases are supported: Oracle, DB2, Informix Dynamic Server, MySQL, Microsoft SQL Server, and Sybase Adaptive Server Enterprise. Contact support for specific versions that have been certified. If your particular version is more recent than one mentioned here it is probably supported.

For more information on Oracle JCA Adapters support for third-party JDBC drivers and databases, see JDBC Driver and Database Connection Configuration.

This section includes the following topics:

- Terms Used.

- Important Notes
- Creating Database Connections

Terms Used

ProductName

This is the name of the database.

Database Name	Supported Database
IBM DB2	IBM DB2 v 10.1
Microsoft SQL Server	SQLServer 2012
MySQL	MySQL v5.6*

DriverClassName

This is the name of the JDBC Driver Class.

Database Name	JDBC Driver
IBM DB2	`weblogic.jdbcx.db2.DB2DataSource`
Microsoft SQL Server	`weblogic.jdbcx.sqlserver.SQLServerDataSource`
MySQL	`com.mysql.jdbc.Driver` / `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource`

ConnectionString

This is the JDBC Connection URL.

Database Name	Connection String
IBM DB2	`jdbc:weblogic:db2://<host>:<port>`
Microsoft SQL Server	`jdbc:weblogic:sqlserver://<host>:<port>`
MySQL	`jdbc:mysql://host:port/database-name`

Username

This is the database user name.

Password

This is the password associated with the user name.

ProcedureName

This is the name of the stored procedure or the function.

ServiceName

This is the service name for the desired operation.

DatabaseConnection

This is the JNDI name of the connection. For example, `eis/DB/<DatabaseConnection>`.

Destination

This is the destination directory for the generated files. For example, `C:\Temp`.

Parameters

The parameters of the stored procedure (for versions of MySQL before 5.2.6 only.)

QueryTimeout

The JDBC query timeout value (in seconds.) The `QueryTimeout` property specifies the maximum number of seconds that the JDBC driver should wait for the specified stored procedure or function to execute. When the threshold is exceeded, `SQLException` is thrown. If the value is zero, then the driver waits indefinitely.

Important Notes

The Adapter Configuration Wizard supports Oracle Database, IBM DB2, AS/400, DB2/ZOS, Microsoft SQL Server, Sybase 15.7, Informix 11.7+, Progress Db 10, and MySQL v5.6 or higher.

This section includes the following topics:

- Microsoft SQL Server
- DB2 Data Types
- IBM DB2 AS/400

Microsoft SQL Server

Table 9-15 lists the supported data types for SQL Server stored procedures and functions.

Table 9-15 Data Types for SQL Server Stored Procedures and Functions

SQL Data Type	XML Schema Type
BIGINT	long
BINARY IMAGE TIMESTAMP VARBINARY	base64Binary
BIT	boolean
CHAR SQL_VARIANT SYSNAME TEXT UNIQUEIDENTIFIER VARCHAR XML (2005 only)	string

Table 9-15 (Cont.) Data Types for SQL Server Stored Procedures and Functions

SQL Data Type	XML Schema Type
DATETIME SMALLDATETIME	dateTime
DECIMAL MONEY NUMERIC SMALLMONEY	decimal
FLOAT REAL	float
INT	int
SMALLINT	short
TINYINT	unsignedByte

DB2 Data Types

Table 9-16 lists the supported data types for DB2 SQL stored procedures:

Table 9-16 Data Types for DB2 SQL Stored Procedures

SQL Data Type	XML Schema Type
BIGINT	long
BLOB CHAR FOR BIT DATA VARCHAR FOR BIT DATA	base64Binary
CHARACTER CLOB VARCHAR	string
DATE TIME TIMESTAMP	dateTime
DECIMAL	decimal
DOUBLE	double
INTEGER	int
REAL	float
SMALLINT	short

The names of other data types are also supported implicitly. For example, `NUMERIC` is equivalent to `DECIMAL` (as is `DEC` and `NUM` as well.)

IBM DB2 supports structured data types (user-defined). However, there is no support for these types in the JDBC drivers. Consequently, a structured data type may not be used as the data type of a parameter in a stored procedure. IBM DB2 also supports user-defined functions. The adapter, however, does not support these functions.

In the Adapter Configuration Wizard, stored procedures are grouped by database user. A *schema* in IBM DB2 is equivalent to a schema in Oracle. Both represent the name of a database user.

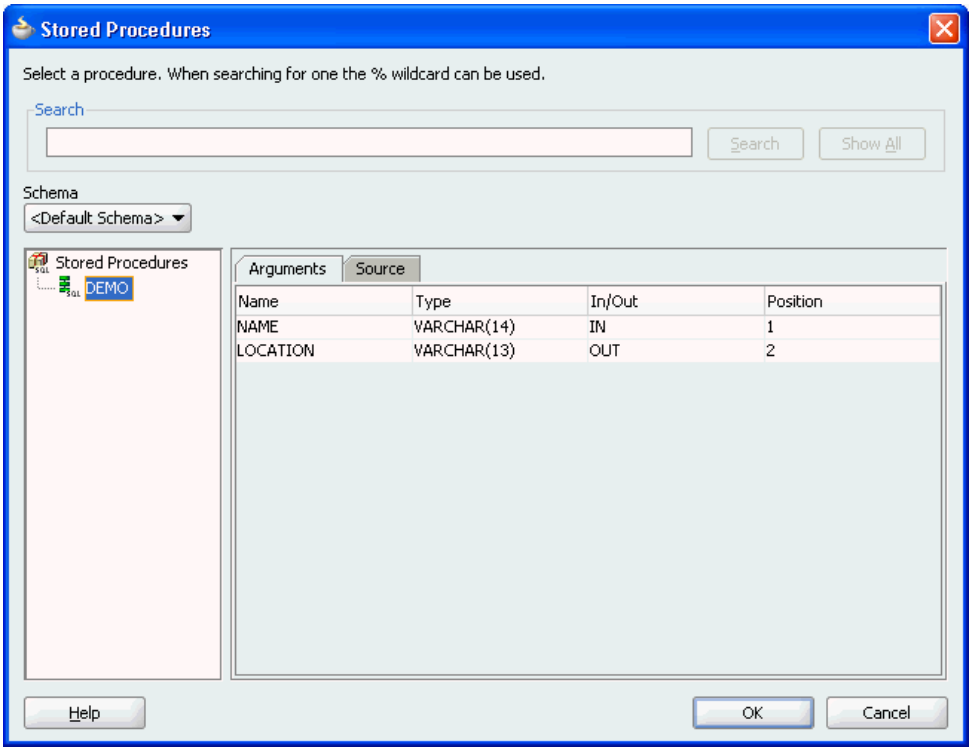
For IBM DB2, **<Default Schema>** refers to the current database user.

Click **<Default Schema>** to select a different database user. The stored procedures in the **Browse** page are those that the database user created in the database specified as <database> in the JDBC Connection URL.

The Adapter Configuration Wizard does not support changing to a different database.

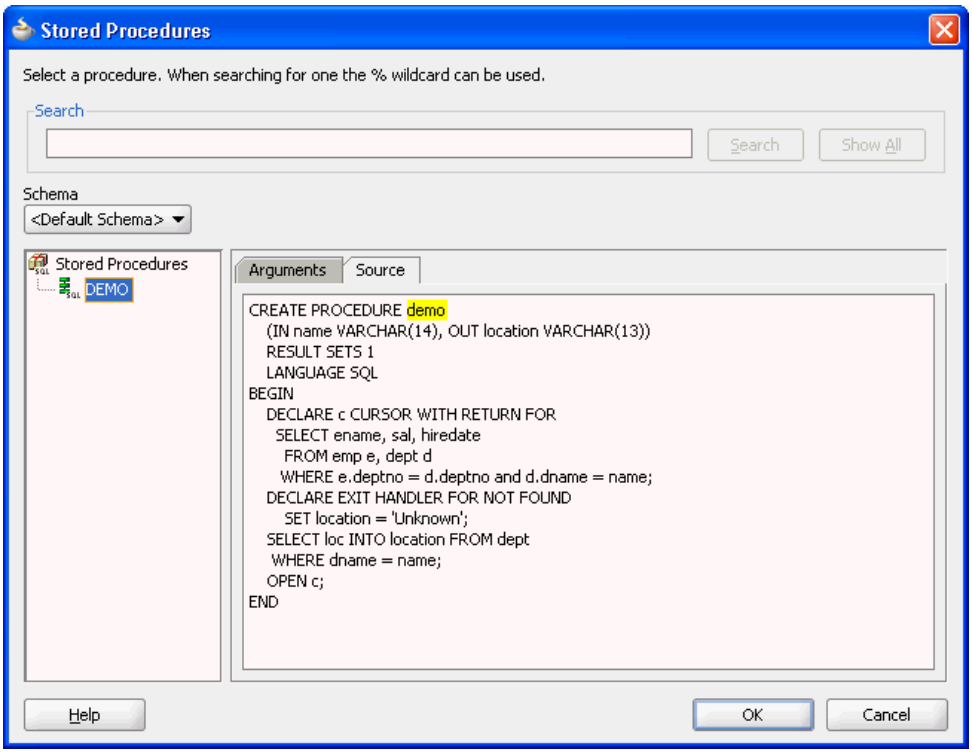
Select the stored procedure in the Stored Procedures dialog, as shown in Figure 9-46. The arguments are shown in the Arguments tab. Click **Search** to find database stored procedures that the user created in the specified database. For example, 'd%' or 'D%' would both find the `DEMO` stored procedure. Clicking **Show All** reveals all of the procedures that the current user created in the specified database.

Figure 9-46 The Stored Procedures Dialog



You can view the source code of the stored procedure by clicking the **Source** tab, as shown in Figure 9-47.

Figure 9-47 The Source Code of the Stored Procedure



IBM DB2 AS/400

Table 9-17 lists the supported data types for IBM DB2 AS/400 stored procedures:

Table 9-17 Data Types for IBM DB2 AS/400 Stored Procedures

SQL Data Type	XML Schema Type
BINARY BINARY LARGE OBJECT BINARY VARYING	base64Binary
CHARACTER CHARACTER LARGE OBJECT CHARACTER VARYING	string
DATE TIME TIMESTAMP	dateTime
DECIMAL NUMERIC	decimal
DOUBLE PRECISION	double
BIGINT	long
INTEGER	int
REAL	float

Table 9-17 (Cont.) Data Types for IBM DB2 AS/400 Stored Procedures

SQL Data Type	XML Schema Type
SMALLINT	short

Distinct types are also supported for data types that are created using the `CREATE DISTINCT TYPE` statement. These data types work in the same way as they do in IBM DB2.

The IBM DB2 AS/400 implementation is based on queries from catalog tables in the `QSYS2` schema. The adapter tries to determine whether the `QSYS2.SCHEMATA` table exists. If it does, then the Adapter Configuration Wizard queries tables in the `QSYS2` schema. Therefore, if your IBM DB2 AS/400 database supports the `QSYS2` schema, then the Adapter Configuration Wizard and the adapter run time should both work.

The Adapter Configuration Wizard checks the `SYSCAT` schema first, and then the `QSYS2` schema. The adapter does not support the catalog tables in the `SYSIBM` schema.

Creating Database Connections

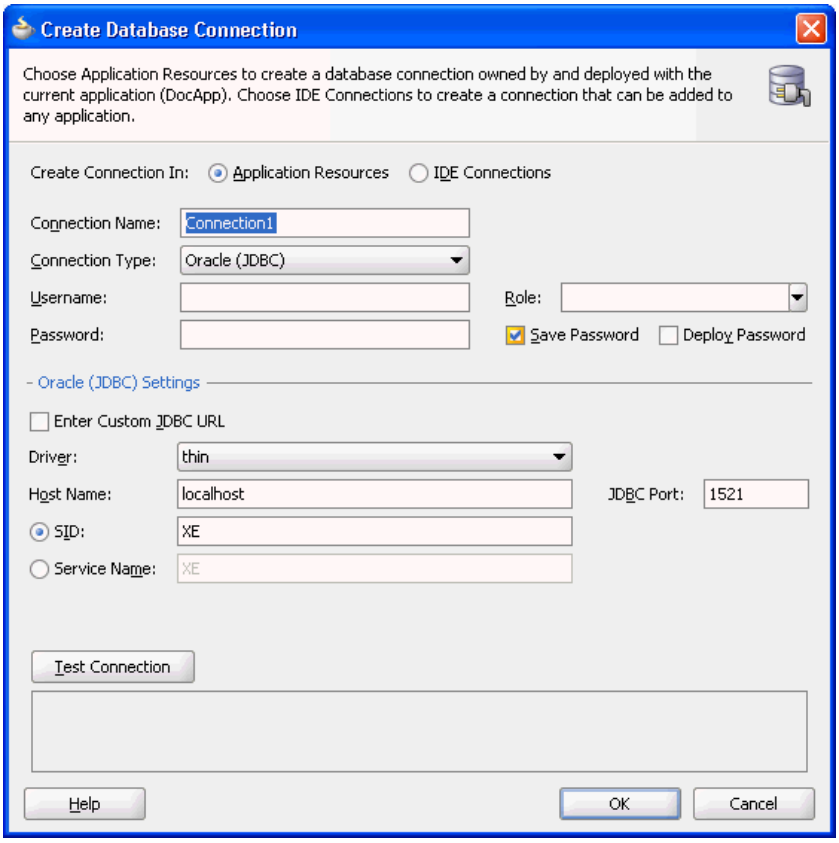
Database connections must be created in JDeveloper to access catalog tables necessary for the Adapter Configuration Wizard to work.

The following are the steps to create a database connection by using JDeveloper:

1. Select **Database Navigator** from View.
2. Right-click the application name, then click **New** followed by **Connections**. Select **Database Connection**.

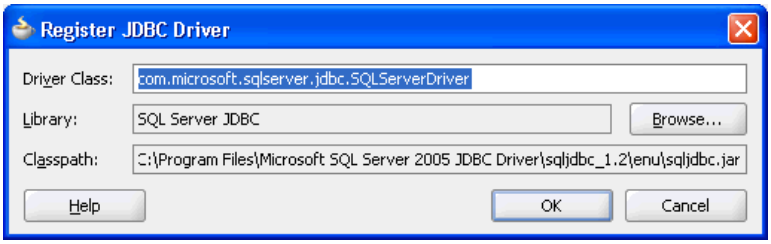
   The Create Database Connection page is displayed, as shown in Figure 9-48.

Figure 9-48 The Create Database Connection



3. Enter a connection name in the **Connection Name** field. For example, `sqlserver`.
4. Select **Generic JDBC** as the Connection Type from the Connection Type list.
5. Enter your Username, Password, and role information.
6. Click **New** for Driver Class. The Register JDBC Driver dialog is displayed, as shown in Figure 9-49.

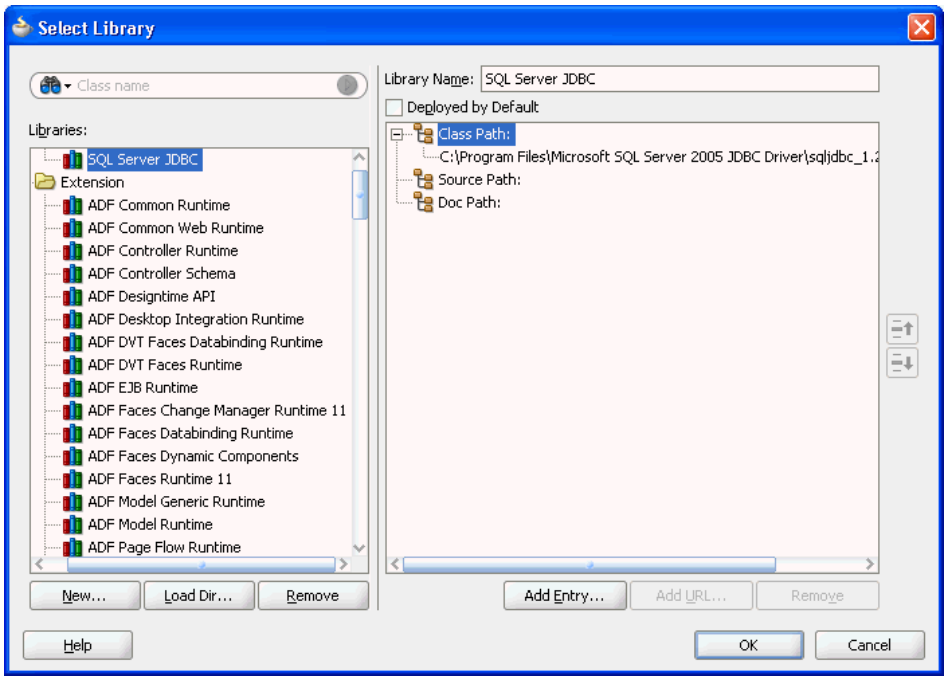
Figure 9-49 The Register JDBC Driver Dialog



7. Enter the Driver Class (for example, `com.microsoft.sqlserver.jdbc.SQLServerDriver`).
8. Create a library or edit an existing one by using the following steps:
   a. Click **Browse** in the Register JDBC Driver dialog.
   b. Click **New** in the Select Library dialog.

The Select Library dialog is displayed, as shown in Figure 9-50.

Figure 9-50 The Select Library Dialog



c. Select an existing library or click **New** to create one.

   The Create Library dialog is displayed.

d. Enter a library name, for example, `SQL Server JDBC`.

e. Click **Add Entry** to add JDBC jar files to the class path.

f. Click **OK** twice to exit the Create Library windows.

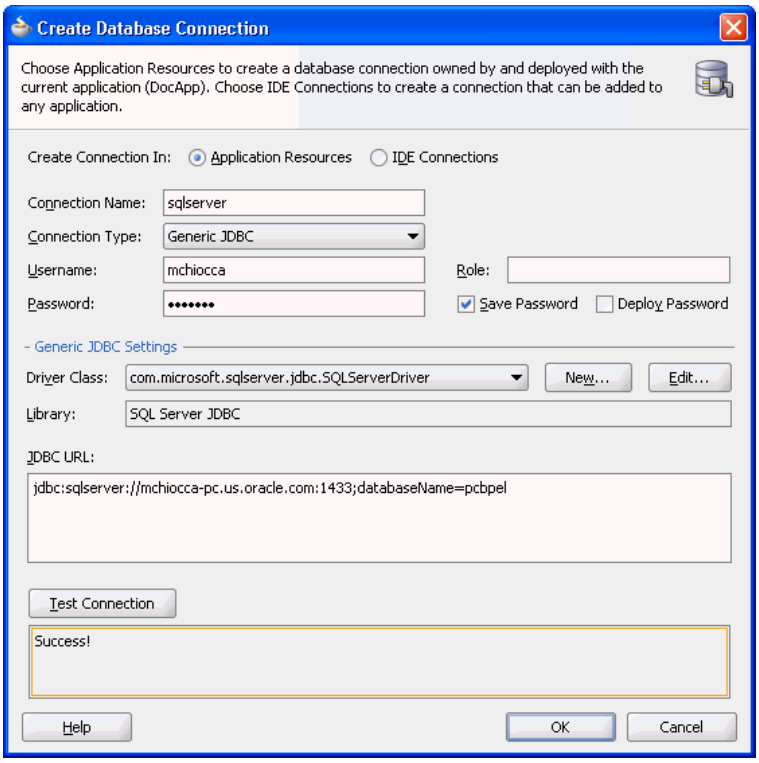
g. Click **OK** to exit the Register JDBC Driver window.

9. Enter your connection string name for **JDBC URL**.

10. Click **Test Connection**.

11. If the connection is successful, then a screen, as shown in Figure 9-51 is displayed.

Figure 9-51 The Create Database Connection Dialog



12. Click **OK** followed by **Finish**.

Design Time: Artifact Generation

The Adapter Configuration Wizard – Stored Procedures is capable of creating a WSDL file and a valid XSD file that describes the signature of a stored procedure or function. The following sections describe the relevant structure and content of both the WSDL and the XSD files, and their relationship with each other.

This section includes the following topics:

- The WSDL–XSD Relationship
- JCA File
- Oracle Data Types
- Generated XSD Attributes
- User-Defined Types
- Complex User-Defined Types
- Object Type Inheritance
- Object References
- Referencing Types in Other Schemas
- XSD Pruning Optimization

The WSDL–XSD Relationship

In the paragraphs that follow, the operation name, `Factorial`, and procedure name, `Factorial`, are taken from an example cited previously (see Figure 9-40). The generated WSDL imports the XSD file.

```
<types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace="http://xmlns.oracle.com/pcbpel/adapter/db/SCOTT/
FACTORIAL/"
      schemaLocation="xsd/SCOTT_FACTORIAL.xsd"/>
  </schema>
</types>
```

The namespace is derived from the schema, package, and procedure name, and appears as the `targetNamespace` in the generated XSD.

A root element called `InputParameters` is created in the XSD file for specifying elements that correspond to the `IN` and `IN/OUT` parameters of the stored procedure. Another root element called `OutputParameters` is also created in the XSD file for specifying elements only if there are any `IN/OUT` or `OUT` parameters. `IN/OUT` parameters appear in both root elements.

These root elements are represented in the XSD file as an unnamed `complexType` definition whose sequence includes one element for each parameter. If there are no `IN` or `IN/OUT` parameters, then the `InputParameters` root element is still created; however, `complexType` is empty. A comment in the XSD file indicates that there are no such parameters. An example of a root elements follows.

```
<element name="InputParameters"
  <complexType>
    <sequence>
      <element …>
       …
    </sequence>
  </complexType>
</element>
```

The WSDL defines message types whose parts are defined in terms of these two root elements.

```
<message name="args_in_msg"
  <part name="InputParameters" element="InputParameters"/>
</message>
<message name="args_out_msg"
  <part name="OutputParameters" element="OutputParameters"/>
</message>
```

The `db` namespace is equal to the `targetNamespace` of the generated XSD. The `args_in_msg` message type always appears in the WSDL while `args_out_msg` is included only if the `OutputParameters` root element is generated in the XSD file.

An operation is defined in the WSDL whose name is identical to the adapter service and whose input and output messages are defined in terms of these two message types.

```
<portType name="Factorial_ptt">
  <operation name="Factorial">
    <input message="tns:args_in_msg"/>
    <output message="tns:args_out_msg"/>
```

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

The input message always appears while the output message depends on the existence of the `OutputParameters` root element in the XSD file. The `tns` namespace is derived from the operation name and is defined in the WSDL as

```
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/db/Factorial/"
```

The root elements in the XSD file define the structure of the parts used in the messages that are passed into and sent out of the Web service encapsulated by the WSDL.

The input message in the WSDL corresponds to the `InputParameters` root element from the XSD file. The instance XML supplies values for the `IN` and `IN/OUT` parameters of the stored procedure. The output message corresponds to the `OutputParameters` root element. This is the XML file that gets generated after the stored procedure has executed. It holds the values of any `IN/OUT` and `OUT` parameters.

JCA File

The JCA file provides adapter configuration information for the service. A connection factory is specified so that the adapter run time can connect to the database, as shown in the following example. Non-managed connection properties should not be specified directly in the JCA file. Instead you should create a connection factory on the application server, and refer to it by name in the JCA file (<connection-factory location).

```
<connection-factory location="eis/DB/oracle" UIConnectionName="oracle"
adapterRef="">
</connection-factory>
```

The JNDI name, *eis/DB/oracle*, was earlier specified as the service connection in the Adapter Configuration Wizard.

End point properties for the interaction are also specified. The name of the schema, package, and procedure are specified, as shown in the following example. The operation name ties the JCA file back to the service WSDL.

```
<connection-factory location="eis/db/oracle" UIConnectionName="oracle"
adapterRef=""/>
<endpoint-interaction portType="Factorial_ptt" operation="Factorial">
  <interaction-spec
    className="oracle.tip.adapter.db.DBStoredProcedureInteractionSpec">
      <property name="ProcedureName" value="FACTORIAL"/>
      <property name="GetActiveUnitOfWork="false"/>
  </interaction-spec>
  </output>
</endpoint-interaction>
```

Note the operation name and procedure name. If an explicit schema had been chosen or if the procedure had been defined in a package, then values for these properties would also be listed here.

Note:

Non-managed connection details are not created in the `DBAdapter.jca` files when you start JDeveloper in the normal mode. However, non-managed connection details are created in the `DBAdapter .jca` files when you start JDeveloper in the preview mode.

Oracle Data Types

Many primitive data types have well-defined mappings and therefore are supported by both the design-time and runtime components. In addition, you can use user-defined types such as `VARRAY`, nested tables, and `OBJECT`.

Table 9-18 lists the supported data types for Oracle stored procedures and functions.

Table 9-18 Data Types for Oracle Stored Procedures and Functions

SQL or PL/SQL Type	XML Schema Type
`BINARY_DOUBLE` `DOUBLE PRECISION`	`double`

`BINARY_FLOAT`
`FLOAT`
`REAL`
	`float`

`BINARY_INTEGER`
`INTEGER`
`PLS_INTEGER`
`SMALLINT`
	`int`

`BLOB`
`LONG RAW`
`RAW`
	`base64Binary`

`CHAR`
`CLOB`
`LONG`
`STRING`
`VARCHAR2`
	`string`

`DATE`
`TIMESTAMP`
`TIMESTAMP WITH TIME ZONE`
	`dateTime`

`DECIMAL` `NUMBER`	`decimal`

Generated XSD Attributes

Table 9-19 lists the attributes used in the generated XSDs.

Table 9-19 Generated XSD Attributes

Attribute	Example	Purpose
`name`	`name="param"`	Name of an element
`type`	`type="string"`	XML schema type
`db:type`	`db:type="VARCHAR2"`	SQL or PL/SQL type
`db:index`	`db:index="1"`	Position of a parameter
`db:default`	`db:default="true"`	Has a default clause
`minOccurs`	`minOccurs="0"`	Minimum occurrences
`maxOccurs`	`maxOccurs="1"`	Maximum occurrences
`nillable`	`nillable="true"`	Permits null values

The `db` namespace is used to distinguish attributes used during run time from standard XML schema attributes. The `db:type` attribute is used to indicate what the database type is so that a suitable JDBC type mapping can be obtained at run time. The `db:index` attribute is used as an optimization by both the design-time and runtime components to ensure that the parameters are arranged in the proper order. Parameter indexes begin at `1` for procedures and `0` for functions. The return value of a function is represented as an `OutputParameter` element whose `name` is the name of the function and whose `db:index` is `0`. The `db:default` attribute is used to indicate whether or not a parameter has a default clause.

The `minOccurs` value is set to `0` to allow for an `IN` parameter to be removed from the XML file. This is useful when a parameter has a default clause defining a value for the parameter (for example, `X IN INTEGER DEFAULT 0`). At run time, if no element is specified for the parameter in the XML file, the parameter is omitted from the invocation of the stored procedure, thus allowing the default value to be used. Each parameter can appear at most once in the invocation of a stored procedure or function. Therefore, `maxOccurs`, whose default value is always `1`, is always omitted from elements representing parameters.

The `nillable` attribute is always set to `true` to allow the corresponding element in the instance XML to have a null value (for example, `<X/>` or `<X></X>`). In some cases, however, to pass an element such as this element, which does have a null value, you must state this explicitly (for example, `<X xsi:nil="true"/>`). The namespace, `xsi`, used for the `nillable` attribute, must be declared explicitly in the instance XML (for example, xmlns:xsi="`http://www.w3.org/2001/XMLSchema`-instance").

User-Defined Types

The Adapter Configuration Wizard can also generate valid definitions for user-defined types such as collections (`VARRAY` and nested tables) and `OBJECT`. These are created as `complexType` definitions in the XSD file.

For `VARRAY`, the `complexType` definition defines a single element in its sequence, called *`name`*`_ITEM`, where *`name`* is the name of the VARRAY element. All array elements in the XML file are so named. Given the following `VARRAY` type definition,

```
SQL> CREATE TYPE FOO AS VARRAY (5) OF VARCHAR2 (10);
```

and a VARRAY element, X, whose type is FOO, the following complexType is generated:

```
<complexType name="FOO">
  <sequence>
    <element name="X_ITEM" db:type="VARCHAR2"
               minOccurs="0" maxOccurs="5"
nillable="true"/>
      <simpleType>
        <restriction base="string">
          <maxLength value="10"/>
        </restriction>
      </simpleType>
  </sequence>
</complexType>
```

The minOccurs value is 0 to allow for an empty collection. The maxOccurs value is set to the maximum number of items that the collection can hold. The db:index attribute is not used. Having nillable set to true allows individual items in the VARRAY to be null.

Note the use of the restriction specified on the element of the VARRAY, FOO. This is used on types such as CHAR and VARCHAR2, whose length is known from the declaration of the VARRAY (or nested table). It specifies the type and maximum length of the element. An element value that exceeds the specified length causes the instance XML to fail during schema validation.

The attribute values of a parameter declared to be of type FOO look as follows in the generated XSD:

```
<element name="X" type="db:FOO" db:type="Array" db:index="1"
          minOccurs="0"
nillable="true"/>
```

The type and db:type values indicate that the parameter is represented as an array defined by the complexType called FOO in the XSD file. The value for db:index is whatever the position of that parameter is in the stored procedure.

A nested table is treated almost identically to a VARRAY. The following nested table type definition,

```
SQL> CREATE TYPE FOO AS TABLE OF VARCHAR2 (10);
```

is also generated as a complexType with a single element in its sequence, called *name*_ITEM. The element has the same attributes as in the VARRAY example, except that the maxOccurs value is unbounded because nested tables can be of arbitrary size.

```
<complexType name="FOO">
  <sequence>
    <element name="X_ITEM" … maxOccurs="unbounded" nillable="true">
      …
    </element>
  </sequence>
</complexType>
```

An identical restriction is generated for the X_ITEM element in the VARRAY. The attributes of a parameter, X, declared to be of this type, are the same as in the VARRAY example.

collections (Varray and nested table) are not supported if they are defined inside of a PL/SQL package specification. For example:

```
SQL> create package pkg as
   >   type vary is varray(10) of number;
   >   type ntbl is table of varchar2(100;
   >   procedure test(v in vary, n in ntbl);
   > end;
   > /
```

If a user selects the *test* procedure in the Adapter Configuration Wizard for stored procedures, an error occurs stating that the types are not supported. However, if the `vary` and `ntbl` type definitions were defined at the root level, outside of the package, then choosing the *test* procedure works without issue. The supported way to use collection types (`Varray` and nested table) is shown in the following example:

```
SQL> create type vary as varray(10) of number;
SQL> create type ntbl as table of varchar2(10);
SQL> create package pkg as
   >   procedure test(v in vary, n in ntbl);
   > end;
   /
```

An `OBJECT` definition is also generated as a `complexType`. Its sequence holds one element for each attribute in the `OBJECT`.

The following `OBJECT`,

```
SQL> CREATE TYPE FOO AS OBJECT (X VARCHAR2 (10), Y NUMBER);
```

is represented as a `complexType` definition called `FOO` with two sequence elements.

```
<complexType name="FOO">
  <sequence>
    <element name="X" db:type="VARCHAR2" minOccurs="0"
                            nillable="true"/>
      <simpleType>
        <restriction base="string">
          <maxLength value="10"/>
        </restriction>
      </simpleType>
    <element name="Y" type="decimal" db:type="NUMBER"
                    minOccurs="0"
nillable="true"/>
  </sequence>
</complexType>
```

The `minOccurs` value is `0` to allow for the element to be removed from the XML file. This causes the value of the corresponding attribute in the `OBJECT` to be set to null at run time. The nillable value is `true` to allow empty elements to appear in the XML file, annotated with the `xsi:nil` attribute, to indicate that the value of the element is null. Again, the `db:index` attribute is not used.

Note the use of a restriction on the `VARCHAR2` attribute. The length is known from the declaration of the attribute in the `OBJECT`.

Complex User-Defined Types

User-defined types can be defined in arbitrarily complex ways. An `OBJECT` can contain attributes whose types are defined as any of the user-defined types mentioned in the preceding section. The type of an attribute in an `OBJECT` can be another `OBJECT`, `VARRAY`, or a nested table, and so on. The base type of a `VARRAY` or a nested table can also be an `OBJECT`. Allowing the base type of a collection to be another collection supports multidimensional collections.

Object Type Inheritance

The Adapter Configuration Wizard is capable of generating a valid XSD for parameters whose types are defined using OBJECT-type inheritance. Given the following type hierarchy,

```
SQL> CREATE TYPE A AS OBJECT (A1 NUMBER, A2 VARCHAR2 (10)) NOT FINAL;
SQL> CREATE TYPE B UNDER A (B1 VARCHAR2 (10));
```

and a procedure containing a parameter, X, whose type is B,

```
SQL> CREATE PROCEDURE P (X IN B) AS BEGIN … END;
```

the Adapter Configuration Wizard generates an InputParameters element for parameter X as

```
<element name="X" type="db:B" db:index="1" db:type="Struct"
     minOccurs="0"
nillable="true"/>
```

where the definition of OBJECT type B in the XSD file is generated as the following complexType.

```
<complexType name="B">
  <sequence>
    <element name="A1" type="decimal" db:type="NUMBER" minOccurs="0"
nillable="true"/>
    <element name="A2" db:type="VARCHAR2" minOccurs="0"  nillable="true">
      ...
    </element>
    <element name="B1" db:type="VARCHAR2" minOccurs="0"  nillable="true">
      ...
    </element>
  </sequence>
</complexType>
```

Restrictions on the maximum length of attributes A2 and B1 are added appropriately. Notice how the OBJECT type hierarchy is flattened into a single sequence of elements that corresponds to all of the attributes in the entire hierarchy.

Object References

The Adapter Configuration Wizard can also generate a valid XSD for parameters that are references to OBJECT types (for example, object references) or are user-defined types that contain an object reference somewhere in their definition. In this example,

```
SQL> CREATE TYPE FOO AS OBJECT (…);
SQL> CREATE TYPE BAR AS OBJECT (F REF FOO, …);
SQL> CREATE PROCEDURE PROC (X OUT BAR, Y OUT REF FOO) AS BEGIN … END;
```

the Adapter Configuration Wizard generates complexType definitions for FOO and BAR as indicated, except that for BAR, the element for the attribute, F, is generated as

```
<element name="F" type="db:FOO" db:type="Ref" minOccurs="0" nillable="true"/>
```

where the type and db:type attribute values indicate that F is a reference to the OBJECT type FOO.

For a procedure PROC, the following elements are generated in the OutputParameters root element of the XSD file:

```
<element name="X" type="db:BAR" db:index="1" db:type="Struct" minOccurs="0"
nillable="true"/>
<element name="Y" type="db:FOO" db:index="2" db:type="Ref" minOccurs="0"
nillable="true"/>
```

For Y, note the value of the db:type attribute, Ref. with the type attribute, the element definition indicates that Y is a reference to FOO.

There is a restriction on the use of object references that limits their parameter mode to OUT only. Passing an IN or IN/OUT parameter into an API that is either directly a REF or, if the type of the parameter is user-defined, contains a REF somewhere in the definition of that type, is not permitted.

Referencing Types in Other Schemas

You can refer to types defined in other schemas if the necessary privileges to access them have been granted. For example, suppose type OBJ was declared in SCHEMA1:

```
SQL> CREATE TYPE OBJ AS OBJECT (…);
```

The type of a parameter in a stored procedure declared in SCHEMA2 can be type OBJ from SCHEMA1:

```
CREATE PROCEDURE PROC (O IN SCHEMA1.OBJ) AS BEGIN … END;
```

This is possible only if SCHEMA1 granted permission to SCHEMA2 to access type OBJ:

```
SQL> GRANT EXECUTE ON OBJ TO SCHEMA2;
```

If the required privileges are not granted, an error occurs when trying to create procedure PROC in SCHEMA2:

```
PLS-00201: identifier "SCHEMA1.OBJ" must be declared
```

Because the privileges have not been granted, type OBJ from SCHEMA1 is not visible to SCHEMA2; therefore, SCHEMA2 cannot refer to it in the declaration of parameter O.

XSD Pruning Optimization

Some user-defined object types can have a very large number of attributes. These attributes can also be defined in terms of other object types that also have many attributes. In short, one object type can become quite large depending on the depth and complexity of its definition.

Depending on the situation, many attributes of a large object type may not even be necessary. It is sometimes desirable to omit these attributes from the object's schema definition. This can be done by physically removing the unwanted XSD elements from the definition of the object type.

See the following example where a stored procedure has a parameter whose type is a complex user-defined type:

```
SQL> CREATE TYPE OBJ AS OBJECT (A, NUMBER, B <SqlType>, C <SqlType>, ...);
SQL> CREATE PROCEDURE PROC (O OBJ) AS BEGIN ... END;
```

The InputParameters root element contains a single element for the parameter, O from the API's signature. A complexType definition is to be added to the generated XSD for the object type, as shown in the following code snippet:

```
<complexType name="OBJ">
  <sequence>
    <element name="A" type="decimal" db:type="NUMBER" minOccurs="0"
```

```
nillable="true"/>
    <element name="B" .../>
    <element name="C" .../>
    ...
  </sequence>
</complexType>
```

If attributes B and C are not required, then their element in the `complexType` definition of OBJ can be removed regardless of its type. Values are not required for these attributes in the instance XML. If parameter O had been an output parameter, then elements corresponding with the pruned attributes are also omitted in the generated XML.

Suppose that the type of parameter A was also a user-defined object type and that the definition of OBJ changed accordingly, as shown in the following example:

```
SQL> CREATE TYPE FOO AS OBJECT (X NUMBER, Y NUMBER, Z NUMBER);
SQL> CREATE TYPE OBJ AS OBJECT (A FOO, B <SqlType>, C <SqlType, ...);
```

In such a case, the API remains unchanged. Elements corresponding to unwanted attributes in the definition of FOO can also be removed regardless of their type. So, for example, if Y is not required, then its element in the `complexType` definition of FOO can be removed in the XSD file.

Pruning the XSD file in this fashion improves the runtime performance of the adapter and can significantly reduce memory consumption, as well.

Note:

Only attributes in user-defined object types can be pruned. You cannot prune (remove) a parameter of the stored procedure by removing its element from the `InputParameters` root element. This can result in an error at run time unless the parameter has a default clause.

Run Time: Before Stored Procedure Invocation

This section discusses important considerations of stored procedure support and a brief overview of some important details regarding what happens before the invocation of a stored procedure or function.

This section includes the following topics:

- Value Binding
- Data Type Conversions

Value Binding

Consider the extraction of values from the XML file and how the run time works given those values. The possible cases for data in the XML file corresponding to the value of a parameter whose type is a supported primitive data type are as follows:

1. The value of an element is specified (for example, `<X>100</X>`, here X=100.)
2. The value of an element is not specified (for example, `<X/>`, here X=null.)
3. The value is explicitly specified as null (for example, `<X xsi:nil="true"/>`, here X=null.)

4. The element is not specified in the XML file at all (for example, `X = <default value>`).

Note:

There is one notable difference that distinguishes Microsoft SQL Server from IBM DB2, MySQL, and AS/400. SQL Server supports parameters that can include a default value in the definition of a stored procedure. Because IBM DB2, MySQL, and AS/400 do not support parameter defaults, every parameter must be represented as an element in the instance XML.

In the first case, the value is taken from the XML file as is and is converted to the appropriate object according to its type. That object is then bound to its corresponding parameter during preparation of the stored procedure invocation.

In the second and third cases, the actual value extracted from the XML file is null. The type converter accepts null and returns it without any conversion. The null value is bound to its corresponding parameter regardless of its type. Essentially, this is equal to passing null for parameter `X`.

The fourth case has two possibilities. The parameter either has a default clause or it does not. If the parameter has a default clause, then the parameter can be excluded from the invocation of the stored procedure. This allows the default value to be used for the parameter. If the parameter is included, then the value of the parameter is used, instead. If the parameter does not have a default clause, then the parameter must be included in the invocation of the procedure. Elements for all parameters of a function must be specified. If an element in the instance XML is missing, then the function is invoked with fewer arguments than is expected.

A null value is bound to the parameter by default:

```
SQL> CREATE PROCEDURE PROC (X IN INTEGER DEFAULT 0) AS BEGIN … END;
```

Here, no value is bound to the parameter. In fact, the parameter can be excluded from the invocation of the stored procedure. This allows the value of `0` to default for parameter `X`.

To summarize, the following PL/SQL is executed in each of these three cases:

1. `"BEGIN PROC (X=>?); END;" - X = 100`
2. `"BEGIN PROC (X=>?); END;" - X = null`
3. There are two possibilities:
4. a. `"BEGIN PROC (); END;" - X = 0` (`X` has a default clause)
   b. `"BEGIN PROC (X=>?); END;" - X = null` (`X` does not have a default clause)

With the exception of default clause handling, these general semantics also apply to item values of a collection or attribute values of an `OBJECT` whose types are a supported primitive data types. The semantics of `<X/>` when the type is user-defined are, however, quite different.

For a collection, whether it is a `VARRAY` or a nested table, the following behavior can be expected, given a type definition such as

```
SQL> CREATE TYPE ARRAY AS VARRAY (5) OF VARCHAR2 (10);
```

and XML for a parameter, X, which has type ARRAY, that appears as follows:

```
<X>
    <X_ITEM xsi:nil="true"/>
    <X_ITEM>Hello</X_ITEM>
    <X_ITEM xsi:nil="true"/>
    <X_ITEM>World</X_ITEM>
</X>
```

The first and third elements of the VARRAY are set to null. The second and fourth are assigned their respective values. No fifth element is specified in the XML file; therefore, the VARRAY instance has only four elements.

Assume an OBJECT definition such as

```
SQL> CREATE TYPE OBJ AS OBJECT (A INTEGER, B INTEGER, C INTEGER);
```

and XML for a parameter, X, which has type OBJ, that appears as

```
<X>
    <A>100</A>
    <C xsi:nil="true"/>
</X>
```

The value 100 is assigned to attribute A, and null is assigned to attributes B and C. Because there is no element in the instance XML for attribute B, a null value is assigned.

The second case, <X/>, behaves differently if the type of X is user-defined. Rather than assigning null to X, an initialized instance of the user-defined type is created and bound instead.

In the preceding VARRAY example, if <X/> or <X></X> is specified, then the value bound to X is an empty instance of the VARRAY. In PL/SQL, this is equivalent to calling the type constructor and assigning the value to X. For example,

```
X := ARRAY();
```

Similarly, in the preceding OBJECT example, an initialized instance of OBJ, whose attribute values have all been null assigned, is bound to X. Similar to the VARRAY case, this is equivalent to calling the type constructor. For example,

```
X := OBJ(NULL, NULL, NULL);
```

To specifically assign a null value to X when the type of X is user-defined, add the xsi:nil attribute to the element in the XML file, as in

```
<X xsi:nil="true"/>
```

Data Type Conversions

This section describes the conversion of data types such as CLOB, DATE, TIMESTAMP, and binary data types including RAW, LONG RAW and BLOB, and similar data types supported by third-party databases.

Microsoft SQL Server, IBM DB2, AS/400, and MySQL support binding various forms of binary and date data types to parameters of a stored procedure, as summarized in Table 9-20.

Table 9-20 Third-Party Database: Binding Binary and Date Values to Parameters of a Stored Procedure

XML Schema Type	IBM DB2 Data Type	AS/400 Data Type	Microsoft SQL Server Data Type	MySQL Data Type
base64Binary	BLOB CHAR FOR BIT DATA VARCHAR FOR BIT DATA	BINARY BINARY LARGE OBJECT BINARY VARYING	BINARY IMAGE TIMESTAMP VARBINARY	BINARY TINYBLOB BLOB MEDIUMBLOB LONGBLOB VARBINARY
dateTime	DATE TIME TIMESTAMP	DATE TIME TIMESTAMP	DATETIME SMALLDATETIME	DATE DATETIME TIMESTAMP

For a CLOB parameter, if the length of the CLOB parameter is less than 4 kilobytes, then the text extracted from the XML file is bound to the parameter as a String type with no further processing. If the length of the CLOB parameter is greater than 4 kilobytes or if the mode of the parameter is IN/OUT then a temporary CLOB parameter is created. The XML file data is then written to the temporary CLOB before the CLOB is bound to its corresponding parameter. The temporary CLOB parameter is freed when the interaction completes. For other character types, such as CHAR and VARCHAR2, the data is simply extracted and bound as necessary. It is possible to bind an XML document to a CLOB parameter (or VARCHAR2 if it is large enough). However, appropriate substitutions for <, >, and so on, must first be made (for example, < for < and > for >).

A few data types require special processing before their values are bound to their corresponding parameters. These include data types represented by the XML Schema types base64Binary and dateTime.

The XML schema type, dateTime, represents TIME, DATE, and TIMESTAMP. The XML values for these data types must adhere to the XML schema representation for dateTime. Therefore, a simple DATE string, 01-JAN-05, is invalid. XML schema defines dateTime as YYYY-MM-DDTHH:mm:ss. Therefore, the correct DATE value is 2005-01-01T00:00:00. Values for these parameters must be specified using this format in the instance XML.

Data for binary data types must be represented in a human readable manner. The chosen XML schema representation for binary data is base64Binary. The type converter uses the javax.mail.internet.MimeUtility encode and decode APIs to process binary data. The encode API must be used to encode all binary data into base64Binary form so that it can be used in an XML file. The type converter uses the decode API to decode the XML data into a byte array. The decode API is used to convert the base64Binary data into a byte array.

For a BLOB parameter, if the length of a byte array containing the decoded value is less than 2 kilobytes, then the byte array is bound to its parameter with no further processing. If the length of the byte array is greater than 2 kilobytes or if the mode of the parameter is IN/OUT, then a temporary BLOB is created. The byte array is then written to the BLOB before it is bound to its corresponding parameter. The temporary BLOB is freed when the interaction completes. For other binary data types, such as RAW and LONG RAW, the base64Binary data is decoded into a byte array and bound as necessary.

Conversions for the remaining data types are straightforward and require no additional information.

Run Time: After Stored Procedure Invocation

After the procedure (or function) executes, the values for any `IN/OUT` and `OUT` parameters are retrieved. These correspond to the values of the elements in the `OutputParameters` root element in the generated XSD.

This section includes the following topics:

- Data Type Conversions
- Null Values
- Function Return Values

Data Type Conversions

Conversions of data retrieved are straightforward. However, `CLOB` (and other character data), `RAW`, `LONG RAW`, and `BLOB` conversions, and conversions for similar data types supported by third-party databases, require special attention.

When a `CLOB` is retrieved, the entire contents of that `CLOB` are written to the corresponding element in the generated XML. Standard DOM APIs are used to construct the XML file. Hence, character data, for types such as `CLOB`, `CHAR`, and `VARCHAR2`, is messaged as needed to make any required substitutions so that the value is valid and can be placed in the XML file for subsequent processing. Therefore, substitutions for <and>, for example, in an XML document stored in a `CLOB` are made so that the value placed in the element within the generated XML for the associated parameter is valid.

Raw data, such as for `RAW` and `LONG RAW` data types, is retrieved as a byte array. For `BLOB`s, the `BLOB` is first retrieved, and then its contents are obtained, also as a byte array. The byte array is then encoded using the `javax.mail.internet.MimeUtility` encode API into `base64Binary` form. The encoded value is then placed in its entirety in the XML file for the corresponding element. The `MimeUtility` decode API must be used to decode this value back into a byte array.

Conversions for the remaining data types are straightforward and require no additional information.

Null Values

Elements whose values are null appear as empty elements in the generated XML and are annotated with the `xsi:nil` attribute. Thus, the `xsi` namespace is declared in the XML file that is generated. Generated XML for a procedure `PROC`, which has a single `OUT` parameter, `X`, whose value is null, looks as follows:

```
<OutputParameters … xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <X xsi:nil="true"/>
</OutputParameters>
```

XML elements for parameters of any type (including user-defined types) appear this way if their value is null.

Function Return Values

The return value of a function is treated as an `OUT` parameter at position `0` whose `name` is the name of the function itself. For example,

```
CREATE FUNCTION FACTORIAL (X IN INTEGER) RETURN INTEGER AS
BEGIN
    IF (X <= 0) THEN RETURN 1;
    ELSE RETURN FACTORIAL (X - 1);
    END IF;
END;
```

An invocation of this function with a value of `5`, for example, results in a value of `120` and appears as `<FACTORIAL>120</FACTORIAL>` in the `OutputParameters` root element in the generated XML.

Run Time: Common Third-Party Database Functionality

The common third-party database functionality at run time includes the following:

- Processing ResultSets
- Returning an INTEGER Status Value

Processing ResultSets

All third-party databases share the same functionality for handling `ResultSets`. The following is a SQL Server example of an API that returns a `ResultSet`:

```
 1> create procedure foo ... as select ... from ...;
 2> go
```

A `RowSet` defined in the generated XSD represents a `ResultSet`. A `RowSet` consists of zero or more rows, each having one or more columns. A row corresponds with a row returned by the query. A column corresponds with a column item in the query. The generated XML for the API shown in the preceding example after it executes is shown in the following example:

```
<RowSet>
  <Row>
    <Column name="<column name>" sqltype="<sql datatype">value</Column>
    ...
  </Row>
    ...
</RowSet>
…
```

The `name` attribute stores the name of the column appearing in the query while the `sqltype` attribute stores the SQL datatype of that column, for example `INT`. The *value* is whatever the value is for that column.

It is possible for an API to return multiple `ResultSets`. In such cases, there is one `RowSet` for each `ResultSet` in the generated XML. All `RowSets` always appear first in the generated XML.

Returning an INTEGER Status Value

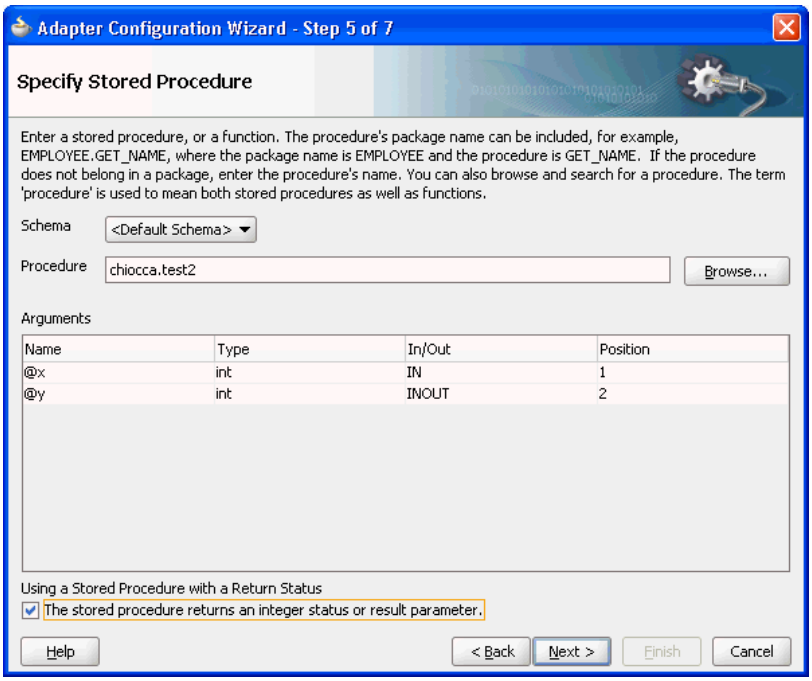
Some databases support returning an `INTEGER` status value using a `RETURN` statement in a stored procedure. Microsoft SQL Server and AS/400 both support this feature. In both cases, the Adapter Configuration Wizard cannot determine whether a

stored procedure returns a status value. Therefore, you must specify that the stored procedure is returning a value. You can use a check box to make this indication.

After choosing a stored procedure in the **Stored Procedures** dialog, the Specify Stored Procedure page appears, as shown in Figure 9-52. The check box appears at the bottom of the page. Select the box to indicate that the procedure contains a `RETURN` statement. You can view the source code of the procedure to determine whether a `RETURN` statement exists.

The check box appears only for stored procedures on databases that support this feature. The check box is not displayed for functions. The value returned by the stored procedure appears as an element in the `OutputParameters` root element in the generated XSD. The name of the element is the name of the stored procedure. The value of a `return` statement is lost after the execution of the stored procedure if the check box is not selected.

Figure 9-52 The Specify Stored Procedure Page



Advanced Topics

This section discusses scenarios for types that are not supported directly using the stored procedure functionality that the Oracle Database Adapter provides. The following sections describe workarounds that address the have to use these data types:

- Row Set Support Using a Strongly Typed XSD
- Row Set Support Using a Weakly Typed XSD
- Support for PL/SQL Boolean_ PL/SQL Record_ and PL/SQL Table Types

Row Set Support Using a Strongly Typed XSD

Currently a `REF CURSOR` by nature can support any arbitrary result set, so the XSD generated at design time is weakly typed.

However the XML output from this is hard to use. It is very difficult to write an Xpath expression or XSL based on a weakly typed XSD and column names as attribute values instead of element names.

Although a row set can represent any result set, it is possible to assume for some procedures that it has the same structure each time, and hence can be described with a strongly typed XSD. A strongly typed XSD is almost a necessity to transform the result set to another XSD later on. You can use the Adapter Configuration Wizard to generate a strongly typed XSD for a `REF CURSOR`.

If a weakly typed XSD is sufficient for your use case, see Row Set Support Using a Weakly Typed XSD.

This section includes the following topics:

- Design Time
- Run Time

For more information, see Row Set Support Using a Strongly or Weakly Typed XSD.

Design Time

If the stored procedure or function you select contains an output parameter of type `RowSet`, you can define a strongly typed XSD for this ref cursor as follows:

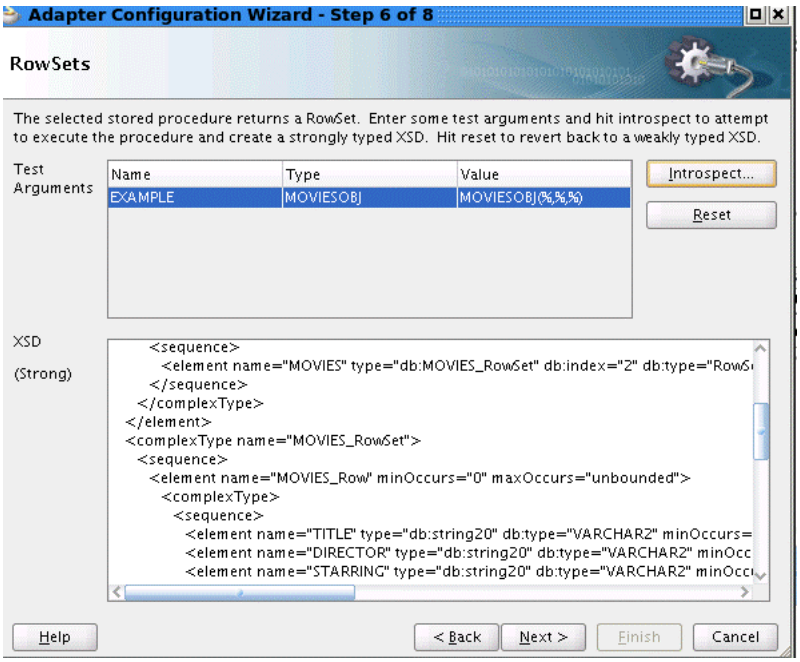
1. Using the Adapter Configuration Wizard, select a stored procedure or function that contains an output parameter of type `RowSet`.

   See steps 1 through 8 in Using Top-Level Standalone APIs.

2. Click **Next**. The RowSets page is displayed, as shown in Figure 9-53.

   By default, the Adapter Configuration Wizard generates a weakly typed XSD for this ref cursor shown in the **XSD** text field. The following example shows this default, weakly typed XSD.

Figure 9-53 RowSets Page



Example - Default Weakly Typed XSD

```
<schema targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/db/SYS/MOVIES_
CURSORS/MOVIES_QUERY/" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:db="http://xmlns.oracle.com/pcbpel/adapter/db/SYS/MOVIES_CURSORS/MOVIES_QUERY/"
elementFormDefault="qualified">
   <element name="InputParameters">
      <complexType>
         <sequence>
            <element name="EXAMPLE" type="db:SYS.MOVIESOBJ" db:index="1" db:type="Struct"
minOccurs="0" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <element name="OutputParameters">
      <complexType>
         <sequence>
            <element name="MOVIES" type="db:RowSet" db:index="2" db:type="RowSet"
minOccurs="0" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="RowSet">
      <sequence>
         <element name="Row" minOccurs="0" maxOccurs="unbounded">
            <complexType>
               <sequence>
                  <element name="Column" maxOccurs="unbounded" nillable="true">
                     <complexType>
                        <simpleContent>
                           <extension base="string">
                              <attribute name="name" type="string" use="required"/>
                              <attribute name="sqltype" type="string" use="required"/>
                           </extension>
                        </simpleContent>
                     </complexType>
                  </element>
               </sequence>
            </complexType>
         </element>
      </sequence>
   </complexType>
   <complexType name="SYS.MOVIESOBJ">
      <sequence>
         <element name="TITLE" db:type="VARCHAR2" minOccurs="0" nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
               </restriction>
            </simpleType>
         </element>
         <element name="DIRECTOR" db:type="VARCHAR2" minOccurs="0" nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
               </restriction>
            </simpleType>
         </element>
         <element name="STARRING" db:type="VARCHAR2" minOccurs="0" nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
               </restriction>
            </simpleType>
         </element>
      </sequence>
   </complexType>
</schema>
```

3. For each of the stored procedure or function arguments:

   - Double-click in the **Value** column.
   - Enter a value for the argument.

     Enter numbers and strings directly, dates as literals (for example, 2009/11/11), and structs as say `MYOBJ('a', 'b')`.
   - Press **Enter**.

   Note:

   You must choose values that are valid for the argument type and that exist in the database.

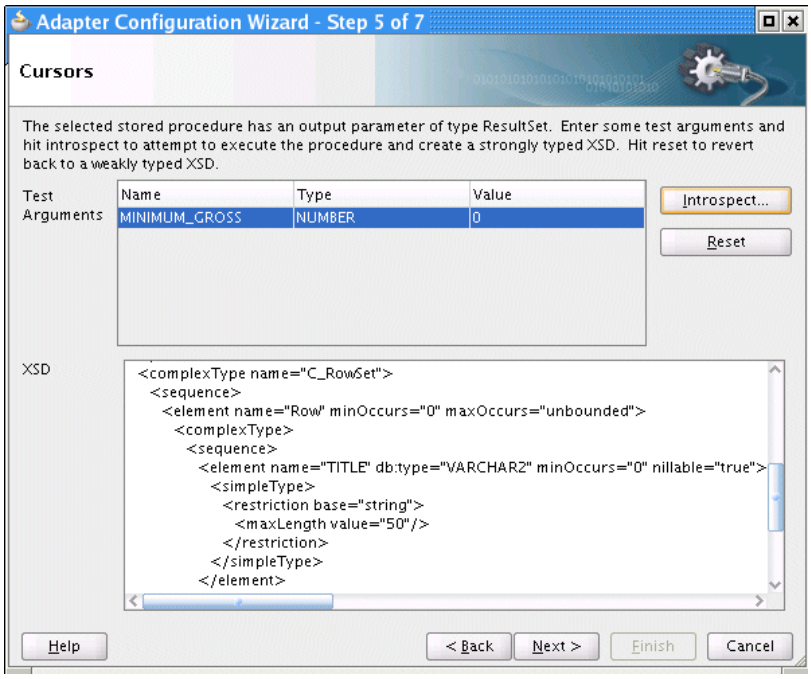
   Oracle recommends that you specify a value for all arguments to ensure that the correct stored procedure or function signature is executed.

4. Click **Introspect**.

   The Adapter Configuration Wizard executes the stored procedure or function using the arguments you specify:

   a. If the stored procedure or function returns a row set with at least 1 row, the RowSets page is updated to display a strongly typed XSD in the **XSD** text field. The following example shows the strongly typed XSD that replaces the default, weakly typed XSD that the previous example shows.

Figure 9-54 RowSets Page: Successful Introspection



Example - Strongly Typed XSD

```
<schema targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/db/SYS/
MOVIES_
CURSORS/MOVIES_QUERY/" xmlns="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:db="http://xmlns.oracle.com/pcbpel/adapter/db/SYS/MOVIES_CURSORS/
MOVIES_
QUERY/" elementFormDefault="qualified">
   <element name="InputParameters">
      <complexType>
         <sequence>
            <element name="EXAMPLE" type="db:SYS.MOVIESOBJ" db:index="1"
db:type="Struct" minOccurs="0" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <element name="OutputParameters">
      <complexType>
         <sequence>
            <element name="MOVIES" type="db:MOVIES_RowSet" db:index="2"
db:type="RowSet" minOccurs="0" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="MOVIES_RowSet">
      <sequence>
         <element name="MOVIES_Row" minOccurs="0" maxOccurs="unbounded">
            <complexType>
               <sequence>
                  <element name="TITLE" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="50"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="DIRECTOR" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="20"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="STARRING" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="100"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="SYNOPSIS" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="255"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="GENRE" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="70"/>
```

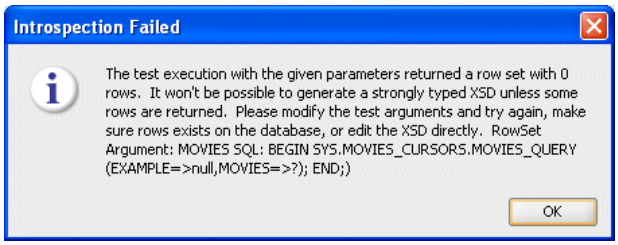
```
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="RUN_TIME" type="decimal" db:type="NUMBER"
minOccurs="0" nillable="true"/>
                  <element name="RELEASE_DATE" type="dateTime"
db:type="DATE"
minOccurs="0" nillable="true"/>
                  <element name="RATED" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="6"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="RATING" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="4"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="VIEWER_RATING" db:type="VARCHAR2"
minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="5"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="STATUS" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="11"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="TOTAL_GROSS" type="decimal"
db:type="NUMBER"
minOccurs="0" nillable="true"/>
                  <element name="DELETED" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="5"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="SEQUENCENO" type="decimal"
db:type="NUMBER"
minOccurs="0" nillable="true"/>
                  <element name="LAST_UPDATED" type="dateTime"
db:type="DATE"
minOccurs="0" nillable="true"/>
                  <element name="POLLING_STRATEGY" db:type="VARCHAR2"
minOccurs="0" nillable="true">
                     <simpleType>
```

```
                        <restriction base="string">
                           <maxLength value="30"/>
                        </restriction>
                     </simpleType>
                  </element>
               </sequence>
            </complexType>
         </element>
      </sequence>
   </complexType>
   <complexType name="SYS.MOVIESOBJ">
      <sequence>
         <element name="TITLE" db:type="VARCHAR2" minOccurs="0"
nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
               </restriction>
            </simpleType>
         </element>
         <element name="DIRECTOR" db:type="VARCHAR2" minOccurs="0"
nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
               </restriction>
            </simpleType>
         </element>
         <element name="STARRING" db:type="VARCHAR2" minOccurs="0"
nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
               </restriction>
            </simpleType>
         </element>
      </sequence>
   </complexType>
</schema>
```

Proceed to step 5.

b. If no rows are returned, the Introspection Failed dialog is displayed, as shown in Figure 9-55.

Figure 9-55 Introspection Failed Dialog

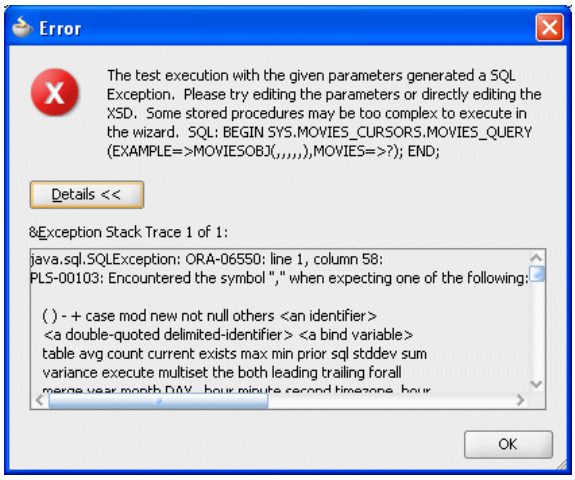


The Adapter Configuration Wizard generates a weakly typed XSD and displays it in the XSD text field by default, overwriting any edits you may have made to a previous version of the XSD.

Go back to step 3 and enter test argument values that returns a row set with at least 1 row.

c. If the stored procedure or function throws an exception, the Introspection Error dialog is displayed, as shown in Figure 9-56.

Figure 9-56 Introspection Error Dialog



The Adapter Configuration Wizard generates a weakly typed XSD and displays it in the XSD text field by default, overwriting any edits you may have made to a previous version of the XSD.

Go back to step 3 and enter test argument values that returns a row set with at least 1 row.

5. Optionally, fine tune the strongly typed XSD by manually editing the schema shown in the **XSD** text filed.

6. Proceed to step 10 in Using Top-Level Standalone APIs.

Run Time

Suppose you have the following package:

```
CREATE PACKAGE PKG AS
       TYPE REF_CURSOR IS REF CURSOR;
       PROCEDURE TEST(C OUT REF_CURSOR);
END;

CREATE PACKAGE BODY PKG AS
       ROCEDURE TEST(C OUT REF_CURSOR) AS
       BEGIN
           OPEN C FOR SELECT DEPTNO, DNAME FROM DEPT;
       END;
END;
```

After using the Adapter Configuration Wizard to define a strongly typed XSD, after the procedure executes, the following XML is generated for parameter, `C`:

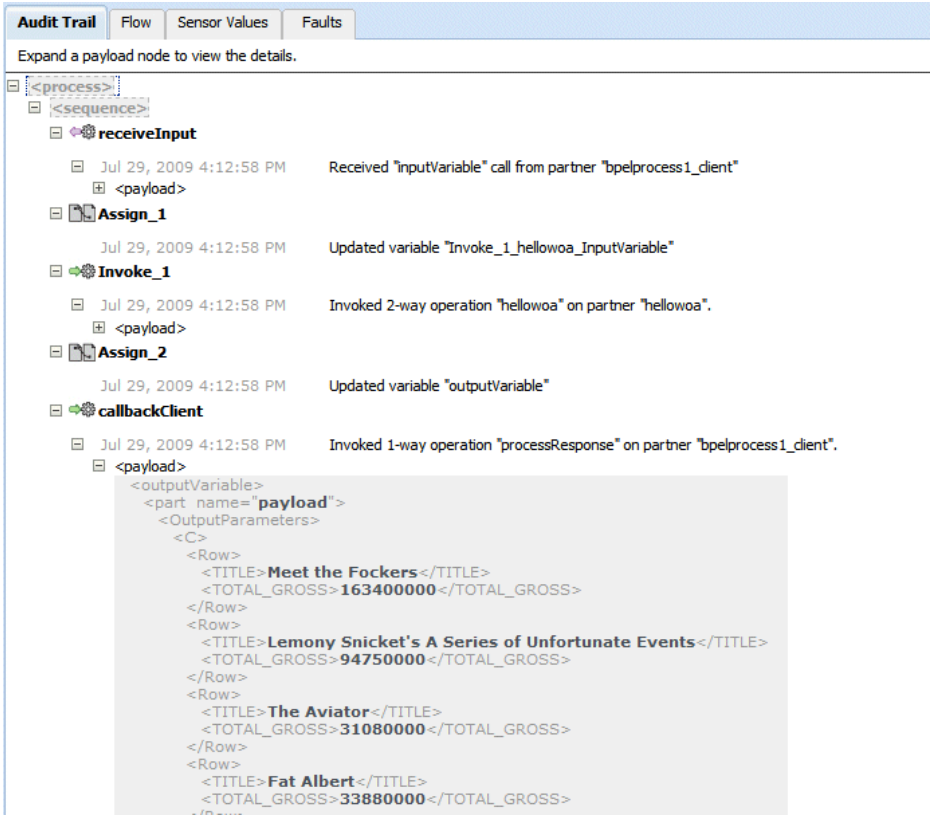
```
<C>
    <C_Row>
        <DEPTNO>10</DEPTNO>
        <DNAME>ACCOUNTING</DNAME>
    </C_Row>
```

```
    <C_Row>
        <DEPTNO>11</DEPTNO>
        <DNAME>DEVELOPMENT</DNAME>
    </C_Row>
    …
</C>
```

Using the Oracle Database Adapter, at run time, it does not matter if the XSD describing the strongly typed ref cursor is inline or imported.

The strongly typed XSD is applied by the SOA runtime and is visible in the Oracle Enterprise Manager Console, where appropriate. For example, Figure 9-57 shows the audit trail for an invoke that returns a ref cursor payload using a strongly typed XSD.

Figure 9-57 Audit Trail for Stongly Typed Payload



Row Set Support Using a Weakly Typed XSD

Currently a `REF CURSOR` by nature can support any arbitrary result set, so the XSD generated at design time is weakly typed. By default, the Adapter Configuration Wizard generates a weakly typed XSD for a `REF CURSOR`.

However the XML output from this is hard to use. It is very difficult to write an Xpath expression or XSL based on a weakly typed XSD and column names as attribute values instead of element names.

Although a row set can represent any result set, it is possible to assume for some procedures that it has the same structure each time, and hence can be described with a strongly typed XSD. A strongly typed XSD is almost a necessity to transform the result set to another XSD later on.

If a strongly typed XSD is better suited to your use case, see Row Set Support Using a Strongly Typed XSD.

This section includes the following topics:

- Design Time
- Run Time

For more information, see Row Set Support Using a Strongly or Weakly Typed XSD.

Design Time

If the stored procedure or function you select contains an output parameter of type `ResultSet`, you can define a weakly typed XSD for this ref cursor as follows:

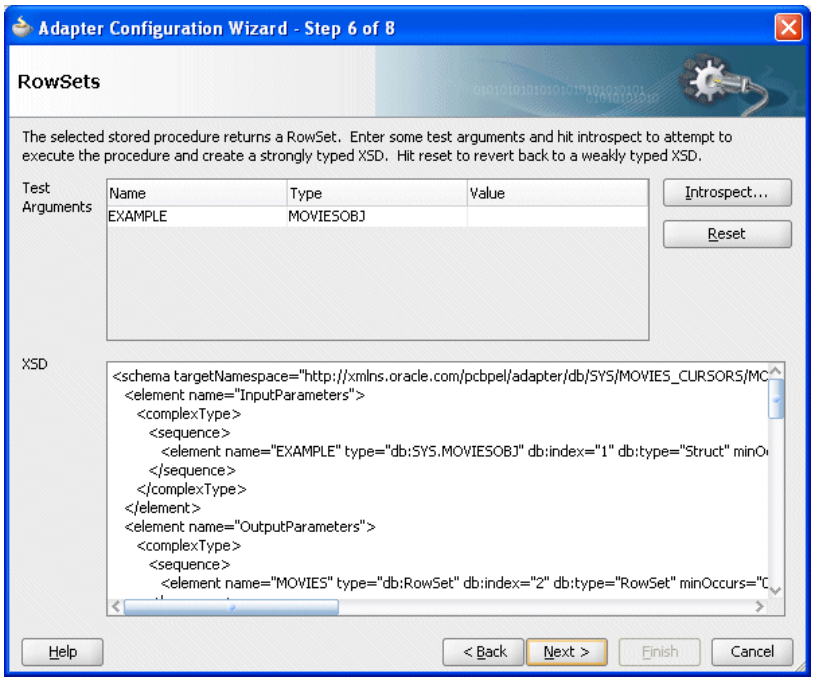
1. Using the Adapter Configuration Wizard, select a stored procedure or function that contains an output parameter of type `ResultSet`.

   See steps 1 through 8 in Using Top-Level Standalone APIs.

2. Click **Next**. The RowSets page is displayed, as shown in Figure 9-58.

   By default, the Adapter Configuration Wizard generates a weakly typed XSD for this ref cursor shown in the **XSD** text field.

   Figure 9-58 RowSets Page



3. Optionally, fine tune the weakly typed XSD by manually editing the schema shown in the **XSD** text filed.

4. Proceed to step 10 in Using Top-Level Standalone APIs.

Run Time

Suppose you have the following package:

```
CREATE PACKAGE PKG AS
       TYPE REF_CURSOR IS REF CURSOR;
       PROCEDURE TEST(C OUT REF_CURSOR);
END;

CREATE PACKAGE BODY PKG AS
       ROCEDURE TEST(C OUT REF_CURSOR) AS
       BEGIN
           OPEN C FOR SELECT DEPTNO, DNAME FROM DEPT;
       END;
END;
```

The REF_CURSOR is a weakly typed cursor variable because the query is not specified. After the procedure executes, the following XML is generated for parameter, C:

```
<C>
    <Row>
        <Column name="DEPTNO" sqltype="NUMBER">10</Column>
        <Column name="DNAME" sqltype="VARCHAR2">ACCOUNTING</Column>
    </Row>
    <Row>
        <Column name="DEPTNO" sqltype="NUMBER">20</Column>
        <Column name="DNAME" sqltype="VARCHAR2">RESEARCH</Column>
    </Row>
    …
</C>
```

There is a total of four rows, each consisting of two columns, DEPTNO and DNAME.

Ref cursors are represented by Java ResultSets. It is not possible to create a ResultSet programmatically by using APIs provided by the JDBC driver. Therefore, ref cursors may not be passed IN to a stored procedure. They can only be passed as IN/OUT and OUT parameters with one caveat. An IN/OUT ref cursor is treated strictly as an OUT parameter. Because no IN value can be provided for an IN/OUT parameter, a null is bound to that parameter when invoking the stored procedure.

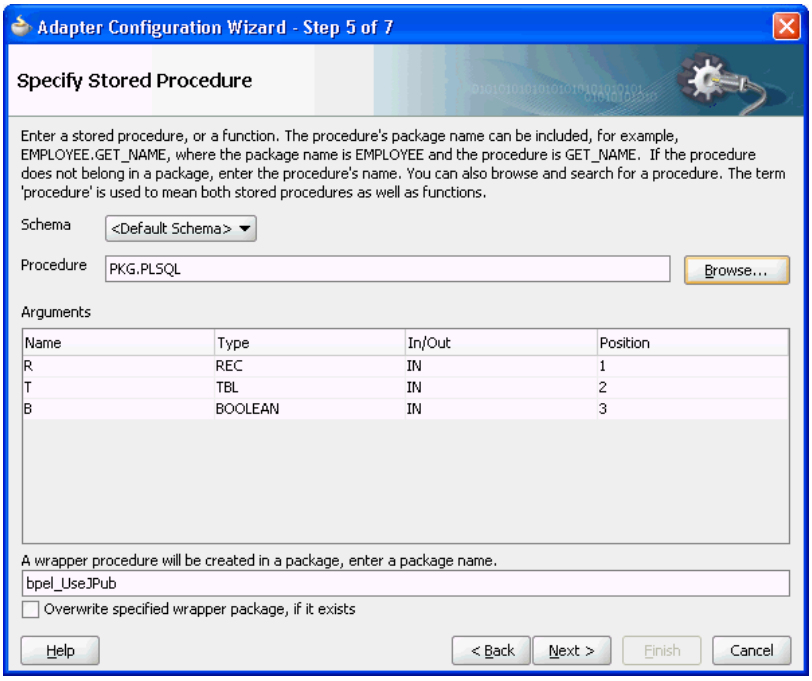
Support for PL/SQL Boolean, PL/SQL Record, and PL/SQL Table Types

The Adapter Configuration Wizard provides a mechanism that detects when these types are used and then invokes Oracle JPublisher to generate the necessary wrappers automatically. Oracle JPublisher generates two SQL files, one to create schema objects, and another to drop them. The SQL that creates the schema objects is automatically executed from within the Adapter Configuration Wizard to create the schema objects in the database schema before the XSD file is generated. For example, suppose the following package specification is declared:

```
CREATE PACKAGE PKG AS
  TYPE REC IS RECORD (X NUMBER, Y VARCHAR2 (10));
  TYPE TBL IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
  PROCEDURE PLSQL (R REC, T TBL, B BOOLEAN);
END;
```

Figure 9-59 shows the step in the Adapter Configuration Wizard that is displayed when PROC procedure from PKG package is selected.

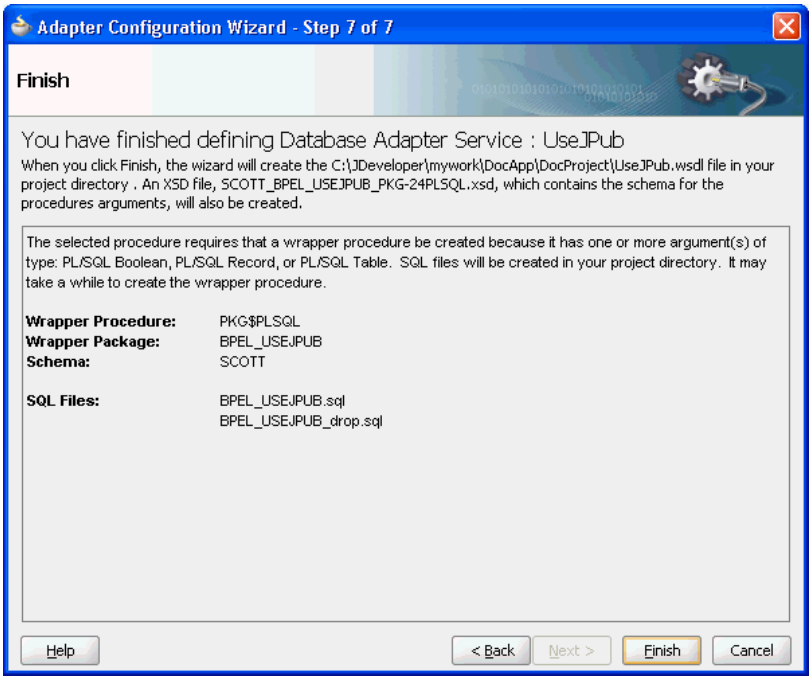
Figure 9-59 Specifying a Stored Procedure in the Adapter Configuration Wizard



As Figure 9-59 shows, the original procedure name is fully qualified, `PKG.PLSQL`. The type of parameter, `R`, is the name of the `RECORD`. The type of `T` is the name of the `TABLE`. The type of `B` is `Boolean`. The name of the wrapper package that is generated is derived from the service name, `bpel_`*`ServiceName`* (for example, `bpel_UseJPub`). This is the name of the generated package that contains the wrapper procedure. You can use the check box to force the Adapter Configuration Wizard to overwrite an existing package when the schema objects are created.

Clicking **Next** twice reveals the **Finish** page of the Adapter Configuration Wizard, as shown in Figure 9-60.

Figure 9-60 Defining a Database Adapter Service: Finish Page



The contents of this page describe what the Adapter Configuration Wizard has detected and what actions are performed when the **Finish** button is clicked. The following summarizes the contents of this page:

1. The name of the generated WSDL is `UseJPub.wsdl`.
2. The name of the JCA file is `UseJPub_db.jca`.
3. Two SQL scripts are created and added to the BPEL process project:
   a. `BPEL_USEJPUB.sql` – Creates the schema objects.
   b. `BPEL_USEJPUB_drop.sql` – Drops the schema objects.
4. The name of the generated XSD is `SCOTT_USEJPUB_PKG-24PLSQL.xsd`.

When you click **Finish**, Oracle JPublisher is invoked to generate the SQL files and load the schema objects into the database. The process of generating wrappers may take quite some time to complete. Processing times for wrappers that are generated in the same package usually require less time after an initial wrapper has been generated for another procedure within the same package.

Note:

You must execute `BPEL_XXXX_drop.sql` when re-creating an Oracle Database Adapter. This is likely due to the JPublisher functionality, which uses a cache when generating wrappers.

The following user-defined types are generated to replace the PL/SQL types from the original procedure:

```
SQL> CREATE TYPE PKG_REC AS OBJECT (X NUMBER, Y VARCHAR2 (10));
SQL> CREATE TYPE PKG_TBL AS TABLE OF NUMBER;
```

The naming convention for these types is *OriginalPackageName_OriginalTypeName*. Boolean is replaced by INTEGER in the wrapper procedure.

Acceptable values for the original Boolean parameter, now that it is an INTEGER are 0 for FALSE and any nonzero INTEGER value for TRUE. Any value other than 1 is considered false. The generated wrapper procedure uses APIs from the SYS.SQLJUTL package to convert from INTEGER to Boolean and vice-versa.

A new wrapper package called BPEL_USEJPUB is created that contains the wrapper for procedure PLSQL, called PKG$PPLSQL, and conversion APIs that convert from the PL/SQL types to the user-defined types and vice-versa. If the original procedure is a root-level procedure, then the name of the generated wrapper procedure is TOPLEVEL $*OriginalProcedureName*.

The generated XSD represents the signature of wrapper procedure PKG$PLSQL and not the original procedure. The name of the XSD file is URL-encoded, which replaces $ with -24.

Note the naming conventions for the generated artifacts:

- The service name is used in the names of the WSDL and SQL files. It is also used as the name of the wrapper package.
- The name of the generated XSD is derived from the schema name, service name, and the original package and procedure names.
- The name of a SQL object or collection data types are derived from the original package name and the name of its corresponding PL/SQL type.
- The name of the wrapper procedure is derived from the original package and procedure names. TOPLEVEL$ is used for root-level procedures.

Note: Before you use the adapter configuration, examine the JCA file and compare it with the generated SQL script. Ensure that the parameters in the JCA file have the values in the generated SQL file. For example, the parameters should have values like,

```
<property name="PackageName" value="BEPL_USEJPUB"/>
<property name="ProcedureName" value="PKG$PLSQL"/>
```

The name of the generated wrapper package is limited to 30 characters. The name of the wrapper procedure is limited to 29 characters. If the names generated by Oracle JPublisher are longer than these limits, then they are truncated.

When the PartnerLink that corresponds with the service associated with the procedure is invoked, then the generated wrapper procedure is executed instead of the original procedure.

Default Clauses in Wrapper Procedures

If a procedure contains a special type that requires a wrapper to be generated, then the default clauses on any of the parameters are *not* carried over to the wrapper. For example, consider

```
SQL> CREATE PROCEDURE NEEDSWRAPPER (
        >     B BOOLEAN DEFAULT TRUE, N NUMBER DEFAULT 0) IS BEGIN … END;
```

Assuming that this is a root-level procedure, the signature of the generated wrapper procedure is

```
TOPLEVEL$NEEDSWRAPPER (B INTEGER, N NUMBER)
```

The `Boolean` type has been replaced by `INTEGER`. The default clauses on both parameters are missing in the generated wrapper. Parameters of generated wrapper procedures never have a default clause even if they did in the original procedure.

In this example, if an element for either parameter is not specified in the instance XML, then an error occurs stating that an incorrect number of arguments have been provided. The default value of the parameter that is specified in the original procedure is not used.

To address this situation, the generated SQL file that creates the wrapper must be edited, restoring the default clauses to the parameters of the wrapper procedure. The wrapper and any additional schema objects must then be reloaded into the database schema. After editing the SQL file, the signature of the wrapper procedure is as follows:

```
TOPLEVEL$NEEDSWRAPPER (B INTEGER DEFAULT 1, N NUMBER DEFAULT 0)
```

For `Boolean` parameters, the default value for true is `1`, and the default value for false is `0`.

As a final step, the XSD file generated for the wrapper must be edited. A special attribute must be added to elements representing parameters that now have default clauses. Add `db:default="true"` to each element representing a parameter that now has a default clause. For example,

```
<element name="B" … db:default="true" …/>
<element name="N" … db:default="true" …/>
```

This attribute is used at run time to indicate that if the element is missing from the instance XML, then the corresponding parameter must be omitted from the procedure call. The remaining attributes of these elements remain exactly the same.

Oracle Database Adapter Use Cases

This describes the Oracle Database Adapter and Oracle Database Adapter - stored procedures use cases.

This section includes the following topics:

- Use Cases for Oracle Database Adapter
- Use Cases for Oracle Database Adapter - Stored Procedures
- Database Adapter/Coherence Integration

Use Cases for Oracle Database Adapter

To obtain Oracle Database Adapter use cases, access the Oracle SOA Sample Code site.

Table 9-21 summarizes the Database Adapter samples on the Sample Code site.

Table 9-21 Adapter Samples on Sample Page Site

Sample	Description
adapters-db-101-MasterDetail.zip	The MasterDetail tutorial shows a simple scenario for replicating data in one set of tables on one database to tables on same/another database.
adapters-db-103-File2StoredProcedure.zip	This sample illustrates the use of the File Adapter interfacing with a stored procedure invocation.
adapters-db-102-Select.zip	The Select tutorial shows how to invoke a DML select/insert/update/delete as part of a larger BPEL process or independently as a web service call.
adapters-db-104-InformixStoredProcedure.zip	This scenario showcases a Database Adapter partner link (Outbound Adapter Service) that invokes a stored procedure on an Informix instance.
adapters-db-105-SybaseStoredProcedure.zip	This scenario showcases a Database Adapter partner link (Outbound Adapter Service) that invokes a stored procedure on a Sybase instance.
adapters-db-107-Polling.zip	This sample shows the three basic polling strategies or how to translate events on the database into initiating instances of a BPEL or SOA process.
adapters-db-201-MovieImages.zip	This sample shows how to read binary files such as JPGs into a blob column in a database using SOA, and then to read them from that table back into a file.
adapters-db-203-RefCursors.zip	The Ref Cursor tutorial shows how to work with stored procedures that return row sets.
adapters-db-207-AdvancedPolling.zip	This sample is subsequent to 107-Polling, and shows more realistic albeit advanced versions of the core polling strategies.
adapters-db-307-ExpertPolling.zip	This bonus sample is subsequent to 207-AdvancedPolling, and shows some ways to customize polling for events beyond what is exposed in the UI.

Use Cases for Oracle Database Adapter - Stored Procedures

This section includes the following use cases:

- Creating and Configuring a Stored Procedure in BPEL Designer
- File To StoredProcedure Use Case

In addition to the uses cases documented in this section, refer to the sample Oracle Database Adapter use cases available by accessing the Oracle SOA Sample Code site.

Table 9-22 shows the Oracle Database Adapter stored procedure samples that are provided with Oracle BPEL PM, and Mediator.

Table 9-22 Oracle Database Adapter Use Cases - Stored Procedures

Tutorial Name	Description
`JPublisherWrapper`	Illustrates a workaround for using PL/SQL `RECORD` types. JPublisher is used to create a corresponding `OBJECT` type whose attributes match the fields of the `RECORD`, and conversion APIs that convert from `RECORD` to `OBJECT` and vice versa. JPublisher also generates a wrapper procedure (or function) that accepts the `OBJECT` and invokes the underlying method using the conversion APIs in both directions. The invoked methods must be installed in an Oracle database (not Oracle Lite).
`RefCursors`	Illustrates how to use a `REF CURSOR` with a strongly typed or weakly typed XSD. You can use the Adapter Configuration Wizard to create a strongly typed XSD for a row set returned by an Oracle Database stored procedure or function `REF CURSOR` variable. For more information, see Row Set Support Using a Strongly or Weakly Typed XSD.
`ResultSetConverter`	Illustrates a workaround for using a `REF CURSOR`. The solution involves the use of a Java stored procedure to convert the corresponding `java.sql.ResultSet` into a collection (either `VARRAY` or `NESTED TABLE`) of OBJECTs.

See Table 9-4 for the structure of the `MOVIES` table, which is used for many of the use cases. The `readme.txt` files that are included with most of the samples provide instructions.

Creating and Configuring a Stored Procedure in JDeveloper BPEL Designer

This use case describes how to integrate a stored procedure into BPEL Process Manager with JDeveloper BPEL Designer.

This use case includes of the following sections:

- Prerequisites
- Creating an Application and an SOA Composite
- Creating the Outbound Service
- Add an Invoke Activity
- Change the Message Part of the Request Message
- Change the Message Part of the Response Message
- Add a Assign Activity for the Input Variable
- Add an Assign Activity for the Output Variable
- Deploying with JDeveloper
- Creating a DataSource in Oracle WebLogic Server Administration Console
- Monitoring Using the Fusion Middleware Control Console

Prerequisites

To perform this use case, you must define the following stored procedure in the SCOTT schema:

```
SQL> CREATE PROCEDURE hello (name IN VARCHAR2, greeting OUT VARCHAR2) AS
  2  BEGIN
  3     greeting := 'Hello ' || name;
  4  END;
  5/
```

Creating an Application and an SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the Application Navigator of JDeveloper, click **New Application**.

   The Create Generic Application - Name your application page is displayed.

2. Enter `MyHelloApp` in the **Application Name** field, and click **Next**.

   The Create Generic Application - Name your project page is displayed.

3. Enter `HelloProject` in the **Project Name** field.

4. In the Available list in the Project Technologies tab, double-click **SOA** to move it to the Selected list.

5. Click **Next**.

   The Create Generic Application - Configure SOA Settings page is displayed.

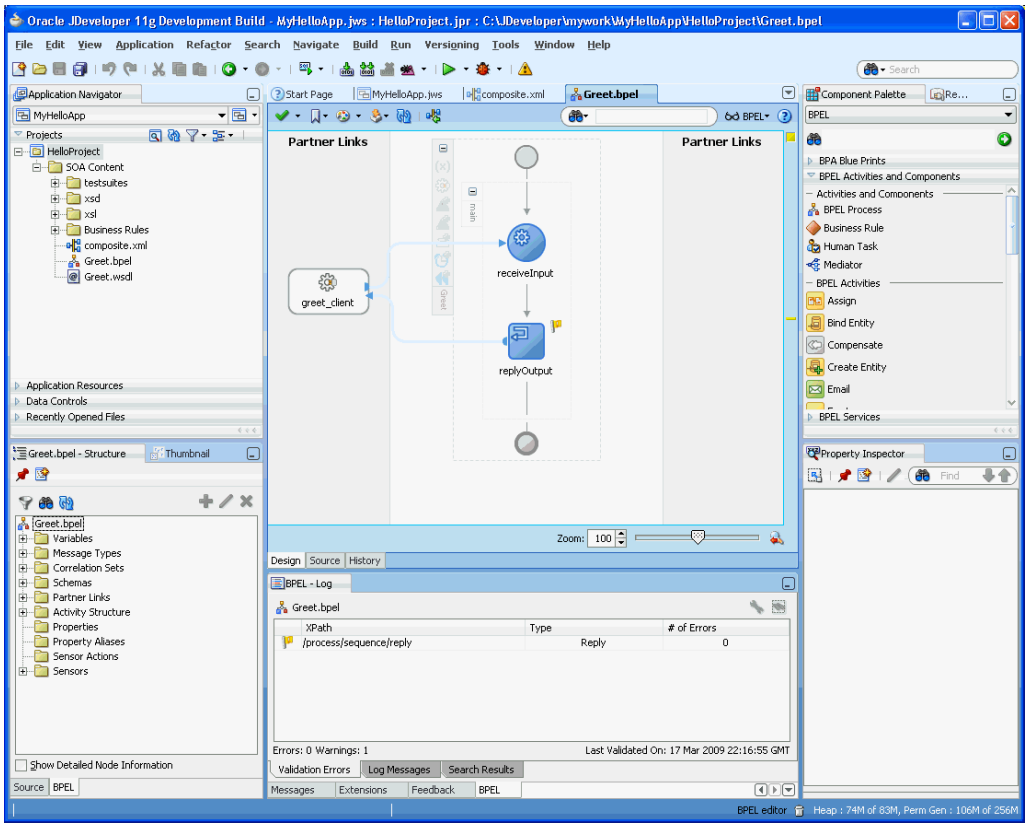
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process page is displayed.

7. Enter `Greet` in the **Name** field, and then select **Synchronous BPEL Process** from the Template box.

8. Click **OK**.

   The Greet BPEL process in the HelloProject of MyHelloApp is displayed in the design area, as shown in Figure 9-61.

Figure 9-61 The JDeveloper - Composite.xml



Creating the Outbound Oracle Database Adapter Service

Perform the following steps to create an outbound Oracle Database Adapter service:

1. Drag and drop **Database Adapter** from the Component Palette to the External References swim lane.

   The Adapter Configuration Wizard Welcome page is displayed.

2. Click **Next**.

   The Service Name page is displayed.

3. Enter `Hello` in the **Service Name** field.

4. Click **Next**.

   The Service Connection page is displayed.

   Note:

   Ensure that you have configured the JNDI name in the `weblogic-ra.xml` file before deploying this application.

   For more information, refer to Creating a Data Source and Recommended Setting for Data Sources Used by Oracle JCA Adapters.

   ---

5. Click the **Create a New Database Connection** icon.

   The Create Database Connection dialog is displayed.

6. Enter the following details in the Create Database Connection dialog:

   a. Enter a connection name in the **Connection Name** field. For example, `Myconnection`.

   b. Select **Oracle (JDBC)** for Connection Type.

   c. Enter the user name and password as `scott/tiger`.

   d. Enter the host name in the **Host Name** field and the JDBC port in the **JDBC Port** field.

   e. Select **SID** and enter the SID name. Alternatively, select **Service Name** and enter the service name.

   f. Click **Test Connection**. A success message is displayed in the Status pane.

   g. Click **OK**.

   The Connection field is populated with the MyConnection connection and the JNDI field is populated with eis/DB/MyConnection.

7. Click **Next**.

   The Operation Type page is displayed.

8. Select **Call a Stored Procedure or Function**, and then click **Next**.

   The Specify Stored Procedure page is displayed.

9. Click **Browse**. Select `HELLO` in the **Stored Procedures** pane.

   The Arguments tab displays the parameters of the stored procedure and the Source tab displays the source code of the stored procedure.

10. Click **OK**.

    The Specify Stored Procedure page is displayed. The Procedure field is populated with the HELLO stored procedure and the arguments for the HELLO stored procedure are also displayed.

11. Click **Next**.

    The Advanced Options page is displayed.

12. Specify any additional advanced options, and then click **Next**.

    The Adapter Configuration Wizard - Finish page is displayed.

13. Click **Finish**.

    The Create Partner Link dialog box is displayed. The name of the partner link is Hello, which is the same as the service name.

14. Click **OK**.

    The outbound Oracle Database Adapter is now configured and the Greet BPEL process is displayed.

Add an Invoke Activity

The following are the steps to add an invoke activity:

1. Drag and drop an **Invoke** activity from the Component Palette to the design area between the `receiveInput` activity and the `replyOutput` activity.
2. Double-click the **Invoke** activity.

   The Edit Invoke dialog is displayed.
3. Enter `Input` in the **Name** field.
4. Click the **Automatically Create Input Variable** icon to the right of the Input Variable field in the Invoke box.

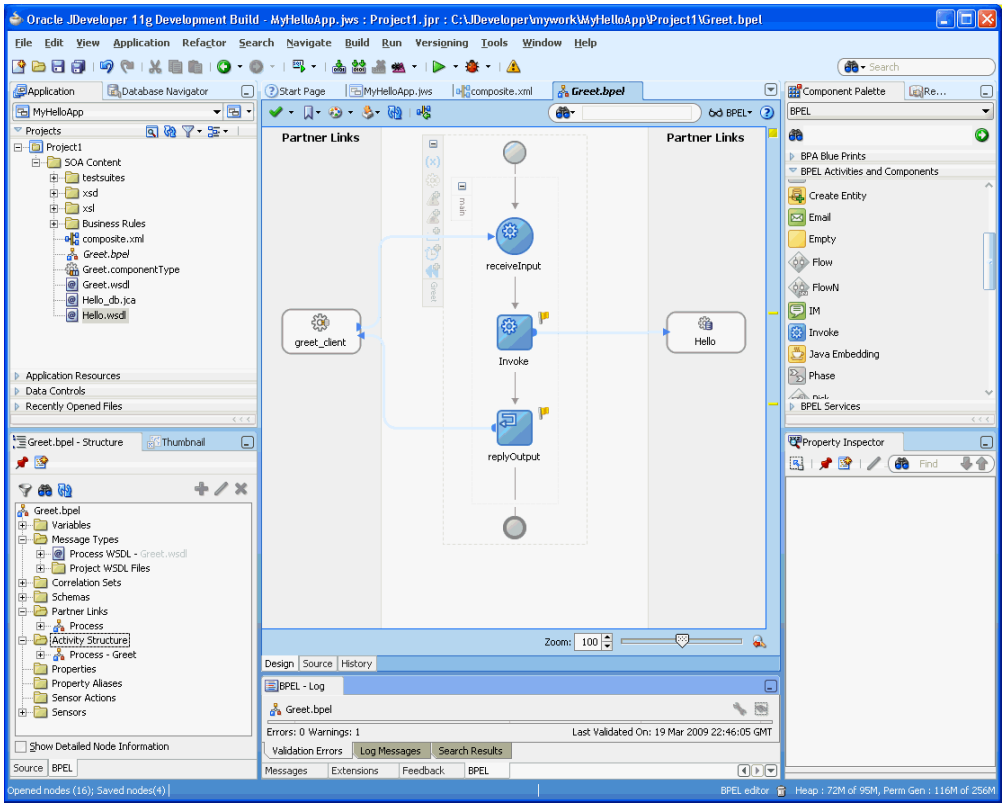
   The Create Variable dialog is displayed.
5. Select the default variable name and click **OK**.

   The Input Variable field is populated with the default variable name. The Invoke dialog is displayed.
6. Repeat the same procedure to select output variable in the Output Variable field.

   In the Variables section of the Edit Invoke dialog the Input and Output variable names are displayed.
7. Click **OK**.

   A line with a right arrow is connected to the Hello partner link is displayed, as shown in Figure 9-62.

Figure 9-62 The Greet.bpel Page



Change the Message Part of the Request Message

When the payload of the request matches the InputParameters, then all of the IN parameters is included in the request. The only IN parameter in this example is *name*.

The following are the steps to change the message part for the `GreetRequestMessage` message:

1. In the Structure Pane for the Greet BPEL process, which is beneath the Application pane, expand **Message Types**, then **Process WSDL - Greet.wsdl**, and then **GreetRequestMessage**.
2. Select **payload**, and then click the **Edit** icon.

   The Edit Message Part - payload dialog is displayed.
3. Choose **Element** and then click the **Search** icon.

   The Type Chooser dialog is displayed.
4. Expand **Project Schema Files**, then **SCOTT_HELLO.xsd**, and select **InputParameters**.
5. Click **OK**.

   The Edit Message Part - payload dialog is displayed.
6. Click **OK**.

Change the Message Part of the Response Message

When the payload of the response matches the OutputParameters, then all of the OUT parameters is included in the response. The only OUT parameter in this example is *greeting*.

The steps for the `GreetResponseMessage` message part are the same as that of GreetRequestMessage with the following exceptions:

1. Expand the **GreetResponseMessage** message type, and then select **payload**.
2. Expand **SCOTT_HELLO.xsd** in the Type Chooser dialog and select **OutputParameters**.
3. Select **OutputParameters**.

Add a Assign Activity for the Input Variable

The following are the steps to add an Assign activity for the input variable:

1. Drag and drop an **Assign** activity from the Component Palette in between the receiveInput and Greet invoke activities in the design area.
2. Double-click the **Assign** activity.

   The Assign dialog is displayed.
3. Click **General** to change the name to `NAME` in the **Name** field.
4. In the **Copy Operation** tab, click the plus icon, and select **Copy Operation** from the list of operations displayed.

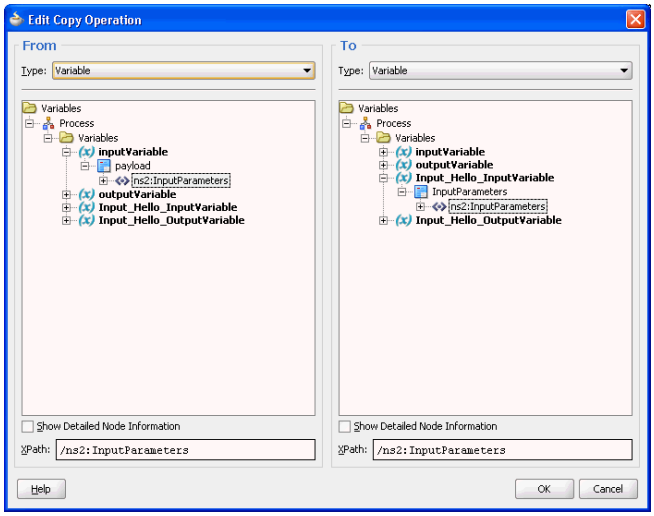
   The Create Copy Operation dialog is displayed.

5. In the From area expand **Variables**, **inputVariable**, **payload**, and then select **ns2:InputParameters**.
6. In the To area expand **Variables**, **Input_Hello_InputVariable**, **InputParameters**, and then select **ns2:InputParameters**.
7. Click **OK**.

   You have assigned a value to the input parameter.

   The Assign dialog is displayed, as shown in Figure 9-63. This dialog shows the assign from the inputVariable payload to the Input_Hello_InputVariable payload.

Figure 9-63 The Create Copy Operation Dialog



8. Click **File**, **Save All**.

Add an Assign Activity for the Output Variable

In the second assign activity, you assign a value to the output parameter.

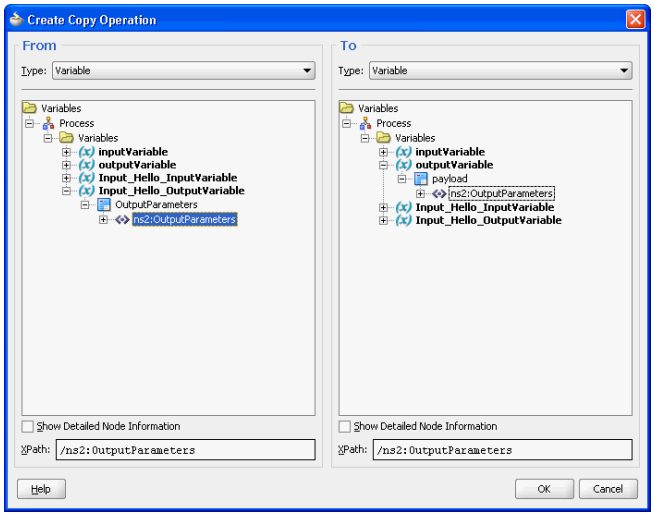
The steps for assigning a value to the output parameter are the same as assigning value to the input parameter with the following exceptions:

1. Drag and drop an **Assign** activity from the Component Palette in between the Greet invoke and replyOutput activities in the design area.
2. Double-click the **Assign** activity.

   The Assign dialog is displayed.
3. Enter `Greeting` in the **Name** field.
4. In the **Copy Operation** tab, click the plus icon, and select **Copy Operation** from the list of operations displayed.

   The Create Copy Operation dialog is displayed.
5. In the From pane expand **Input_Hello_OutputVariable**, **OutputParameters**, and then select **ns2:OutputParameters**, as shown in Figure 9-64.
6. In the To pane expand **outputVariable**, **payload**, and then select **ns2:OutputParameters**, as shown in Figure 9-64

Figure 9-64 The Create Copy Operation Dialog



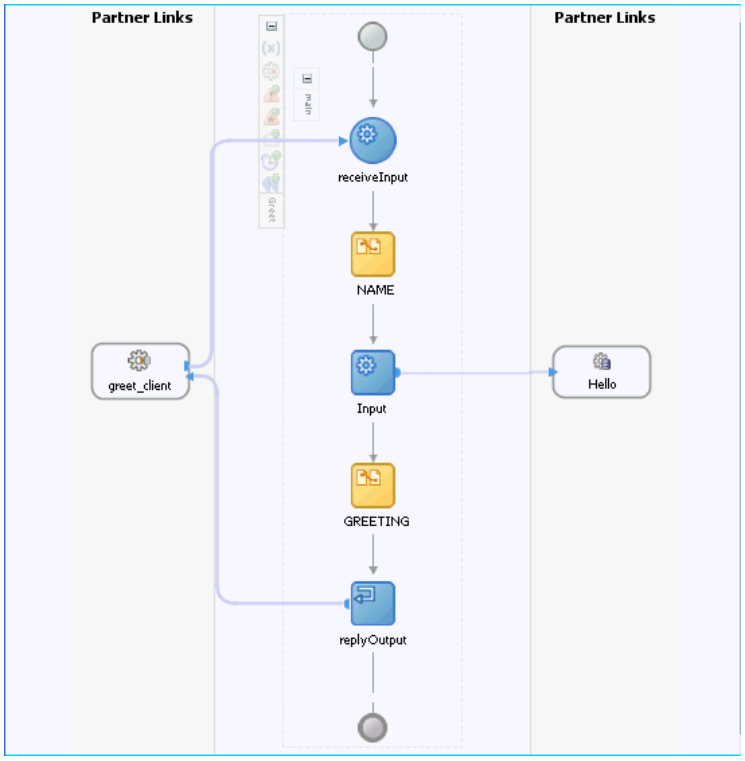
7. Click **OK**.

   You have assigned a value to the output parameter.

8. Click **File**, **Save All**.

   You have completed modeling a BPEL Process. The final BPEL process is displayed, as shown in Figure 9-65.

Figure 9-65 The Final BPEL Process Screen



Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, use the following steps:

1. Create an application server connection using the procedure described in Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application using the procedure described in Deploying Oracle JCA Adapter Applications from .

Creating a DataSource in Oracle WebLogic Server Administration Console

Before you can test the `HelloProject` you must create a data source using the Oracle WebLogic Server Administration Console.

The following are the steps:

1. Enter `http://<hostname>:<port>/console` in your Web browser.
2. Enter a user name and password and click **Log In**.

   The administration console is displayed.
3. In the Services area under JDBC click **Data Sources**.

   A summary of JDBC Data Sources is displayed.
4. Click **New**.

   The Create a New JDBC Data Source page is displayed.
5. In the Create a New JDBC Data Source page, enter the following details:
   - `MyDataSource` in the **Name** field.
   - `jdbc/MyDataSource` in the **JNDI Name** field.
   - The Database Type is `Oracle`.
   - The Database Driver is `Oracle's Driver (Thin XA) for Instance Connections; Versions 9.0.1, 9.2.0, 10, 11`.
6. Click **Next**.

   A message stating that no other transaction configuration options are available is displayed.
7. Click **Next**.

   The Create a New Data Source page is displayed.
8. Enter the following details:.
   - **Database Name**: This is usually the `SID`.
   - **Host Name**: Enter the name of the host computer.
   - **Port Number**: Enter the port number.

     The default port is `1521`.

- **Database User Name**: Enter `SCOTT`
- **Password**: Enter `TIGER`.
- **Confirm Password**: Enter `TIGER`.

9. Click **Next**.

   A summary of the data source configuration is displayed.

10. Click **Test Configuration**.

    The Messages area indicates that the connection test succeeded.

11. Click **Next**. Select **AdminServer** as the target by selecting the check box.

12. Click **Finish**.

    The summary of JDBC Data Sources now includes the MyDataSource data source that you created in the preceding steps.

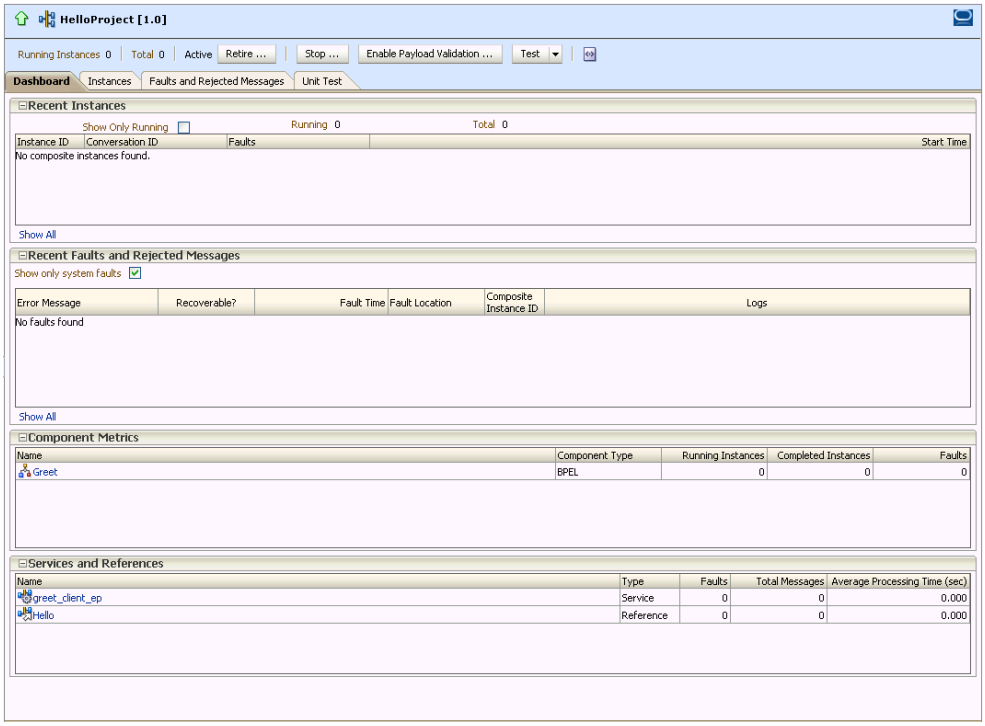
Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. A list of SOA composites is displayed, including the HelloProject[1.0] that you created in the preceding steps.

2. Click the **HelloProject[1.0]** link. The **Dashboard** tab is displayed, as shown in Figure 9-66.

   Figure 9-66 The Dashboard Tab of the HelloProject[1.0] Project



3. Click Test. A new browser window is displayed.

4. Enter your name in the **NAME** field that is marked `xsd:string` and then click **Invoke**.

   The browser window displays the Test Result.

5. To view the XML file in readable form, click **Formatted XML**. Figure 9-67 shows the formatted XML file.

Figure 9-67 The Formatted XML File

```
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <env:Header>
  <wsa:ReplyTo>
   <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
  </wsa:ReplyTo>
 </env:Header>
 <env:Body>
  <OutputParameters
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.oracle.com/pcbpel/adapter/db/SCOTT/HELLO/">
   <GREETING
     xmlns="http://xmlns.oracle.com/pcbpel/adapter/db/SCOTT/HELLO/">Hello Michael</GREETING>
  </OutputParameters>
 </env:Body>
</env:Envelope>
```

File To StoredProcedure Use Case

This use case illustrates the execution of an Oracle stored procedure. The input to the stored procedure is obtained by reading a file using the File Adapter. The stored procedure executes, populating a table with the data from its input parameter.

To obtain the `adapter-db-101-file2storedprocedure` use case, access the Oracle SOA Sample Code site.

This use case includes the following topics:

- Prerequisites
- Creating an Application and an SOA Project
- Creating the Outbound Service
- Creating an Invoke Activity
- Creating the Inbound File Adapter Service
- Adding a Receive Activity
- Adding an Assign Activity
- Wiring Services and Activities
- Deploying with JDeveloper
- Creating a Data Source
- Adding a Connection-Instance
- Testing using the File Adapter Service and SQL*Plus
- Monitoring Using the Fusion Middleware Control Console

Prerequisites

To perform the file to stored procedure use case, the following schema objects and stored procedure must be defined in the SCOTT/TIGER schema before modeling the BPEL Composite using JDeveloper.

```
create type address as object
(
  street varchar2(20),
  city   varchar2(15),
  state  char(2),
  zip    char(5)
);
create type customer as object
(
  fname varchar2(10),
  lname varchar2(10),
  loc   address,
  email varchar2(25),
  phone varchar2(15)
);
create type collection as table of customer;
create table customers
(
  name  varchar2(25),
  loc   varchar2(45),
  email varchar2(25),
  phone varchar2(15)
);
create procedure add_customers(c in collection) as
begin
  for i in c.first .. c.last loop
    insert into customers values (
      c(i).lname || ', ' || c(i).fname,
      c(i).loc.street || ', ' || c(i).loc.city || ', ' || c(i).loc.state || ' ' ||
 c(i).loc.zip,
      c(i).email,
      c(i).phone    );
  end loop;
end;
```

You can define these schema objects and stored procedure using the `adapters-db-101-file2storedprocedure/artifacts/sql/customers.sql` file from the `adapters-db-101-file2storedprocedure` sample by accessing the Oracle SOA Sample Code site.

Creating an Application and an SOA Project

You must create a JDeveloper application to contain the SOA composite. Use the following steps to create a new application, an SOA project:

1. Open JDeveloper.
2. In the Application Navigator, click **New Application**. The Create Generic Application - Name your Application page is displayed.
3. Enter `File2SPApp` in the **Application Name** field.
4. In the Application Template list, select **Generic Application**.
5. Click **Next**.

   The Create Generic Application - Name your project page is displayed.

6. In the **Project Name** field, enter a descriptive name. For example, `File2SPProject`.
7. In the **Available list** in the **Project Technologies** tab, double-click **SOA** to move it to the Selected list.
8. Click **Next**. The Create Generic Application - Configure SOA Settings page is displayed.
9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

   You have created a new application, and an SOA project. This automatically creates an SOA composite.

   The Create BPEL Process page is displayed.
10. Enter a name for the BPEL process in the **Name** field. For example, `File2SP`.
11. Select **Define Service Later** in the Template list, and then click **OK**.

    The `File2SP` BPEL process in the `File2SPProject` of `File2SPApp` is displayed in the design area.

Creating the Outbound Oracle Database Adapter Service

Perform the following steps to create an outbound Oracle Database Adapter service:

1. Drag and drop **Database Adapter** from the Service Adapters list to the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `File2SPService` in the **Service Name** field.
4. Click **Next**.

   The Service Connection page is displayed.
5. Click the **Create a New Database Connection** icon.

   The Create Database Connection dialog is displayed.
6. Enter the following details in the Create Database Connection dialog:
   a. Enter a connection name in the **Connection Name** field. For example, `MyConnection`.
   b. Select **Oracle (JDBC)** for Connection Type.
   c. Enter the user name and password as `scott/tiger`.
   d. Enter the host name in the **Host Name** field and the JDBC port in the **JDBC Port** field.
   e. Select **SID** and enter the SID name. Alternatively, select **Service Name** and enter the service name.
   f. Click **Test Connection**. A success message is displayed in the Status pane.

g. Click **OK**.

The Connection field is populated with the MyConnection connection and the JNDI field is populated with eis/DB/MyConnection.

7. Click **Next**.

The Adapter Interface page is displayed.

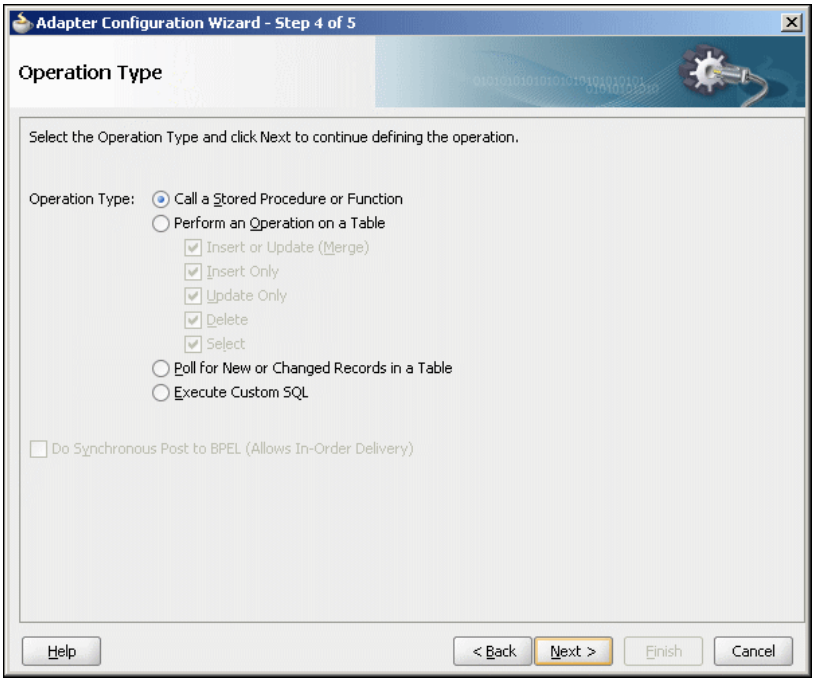
8. In the Adapter Interface page, select **Define from operation and schema (specified later)**, and the click **Next**.

The Operation Type page is displayed.

9. Select **Call a Stored Procedure or Function**, as shown in Figure 9-68, and click **Next**.

The Specify Stored Procedure page is displayed.

Figure 9-68 The Adapter Configuration Wizard - Operation Type Page



10. Click **Browse**. Select `ADD_CUSTOMERS` in the **Stored Procedures** pane.

The Arguments tab displays the parameters of the stored procedure and the Source tab displays the source code of the stored procedure.

11. Click **OK**.

The Specify Stored Procedure page is displayed.

The Procedure field is populated with the `ADD_CUSTOMERS` stored procedure and the arguments for the `ADD_CUSTOMERS` stored procedure are also displayed.

12. Click **Next**.

The Advanced Options page is displayed.

13. Specify any additional options, and then click **Next**.

The Finish page is displayed.

14. Click **Finish**.

    The Create Partner Link dialog is displayed.

    The name of the partner link is File2SPService, which is the same as the service name.

15. Click **OK**.

    The outbound Oracle Database Adapter is now configured and the File2SP BPEL process is displayed.

Creating an Invoke Activity

You must complete the BPEL process by creating an Invoke activity. This creates the input variables.

The following are the steps to create an Invoke activity:

1. Click **File**, **Save All**.
2. Drag and drop an **Invoke** activity from the Component Palette to the design area.
3. Drag the right arrow on the right of the Invoke activity and connect it to the `File2SPService` partner link.

   The Edit Invoke dialog is displayed.

4. Enter `Invoke` in the **Name** field.
5. Click the **Automatically Create Input Variable** icon to the right of the Input Variable field in the Invoke dialog.

   The Create Variable dialog is displayed.

6. Select the default variable name and click **OK**.

   The Input variable name is displayed in the Variables area of the Edit Invoke dialog.

7. Click **OK**.

   A line with a right arrow connecting to the is `File2SPService` partner link is displayed.

Creating the Inbound File Adapter Service

Perform the following steps to create an inbound File adapter service. This creates the service that reads input XML from a file directory:

1. Drag and drop the **File Adapter** from the Component Palette to the External References swim lane.

   The Adapter Configuration Wizard Welcome page is displayed.

2. Click **Next**. The Service Name page is displayed.
3. Enter `ReadCustomers` in the **Service Name** field.
4. Click **Next**.

   The Adapter Interface page is displayed.

5. Select **Define from operation and schema (specified later)**, and then click **Next**. The Operation page is displayed.
6. Select **Read File** as the Operation Type and **Read** as the Operation Name. Do not select the other check boxes.
7. Click **Next**.

   The File Directories page is displayed.
8. Select **Physical Path**, and enter a physical path in the **Directory for Incoming Files** field.
9. Select **Process files recursively** and **Delete files after successful delivery**, and the click **Next**.

   The File Filtering page is displayed.
10. Specify **File Wildcards**, enter `customers.xml` in the **Include Files with Name Pattern** field, and then click **Next**.

    The File Polling page is displayed.
11. Specify any value in the **Polling Frequency** field, and click **Next**.

    The Message page is displayed.
12. Click **Browse For Schema File** that is displayed at the end of the URL field.

    The Type Chooser dialog is displayed.
13. Click **Project Schema Files**, **SCOTT_ADD_CUSTOMERS.xsd**, and **InputParameters**.
14. Click **OK**.

    The Messages page is displayed again. The URL is `xsd/SCOTT_ADD_CUSTOMERS.xsd`, and the Schema Element is `InputParameters`.
15. Click **Next**.

    The Finish page is displayed.
16. Click **Finish**.

    This terminates the inbound File Adapter service.
17. Click **OK** to complete the partner link.
18. Click **File**, **Save All**.

Adding a Receive Activity

The File Adapter Service provides input to the Receive Activity, which then initiates the rest of the BPEL Process.

The following are the steps to add a Receive activity:

1. Double-click **File2SP**. The File2SP.bpel page is displayed.
2. Drag and drop a **Receive** activity from the Component Palette to the design area.

3. Drag the left arrow on the left of the Receive activity and connect it to the `ReadCustomers` partner link.

   The Edit Receive dialog is displayed.

4. Enter `Receive` in the **Name** field.

5. Click the **Automatically Create Input Variable** icon to the right of the Variable field in the Edit Receive dialog.

   The Create Variable dialog is displayed.

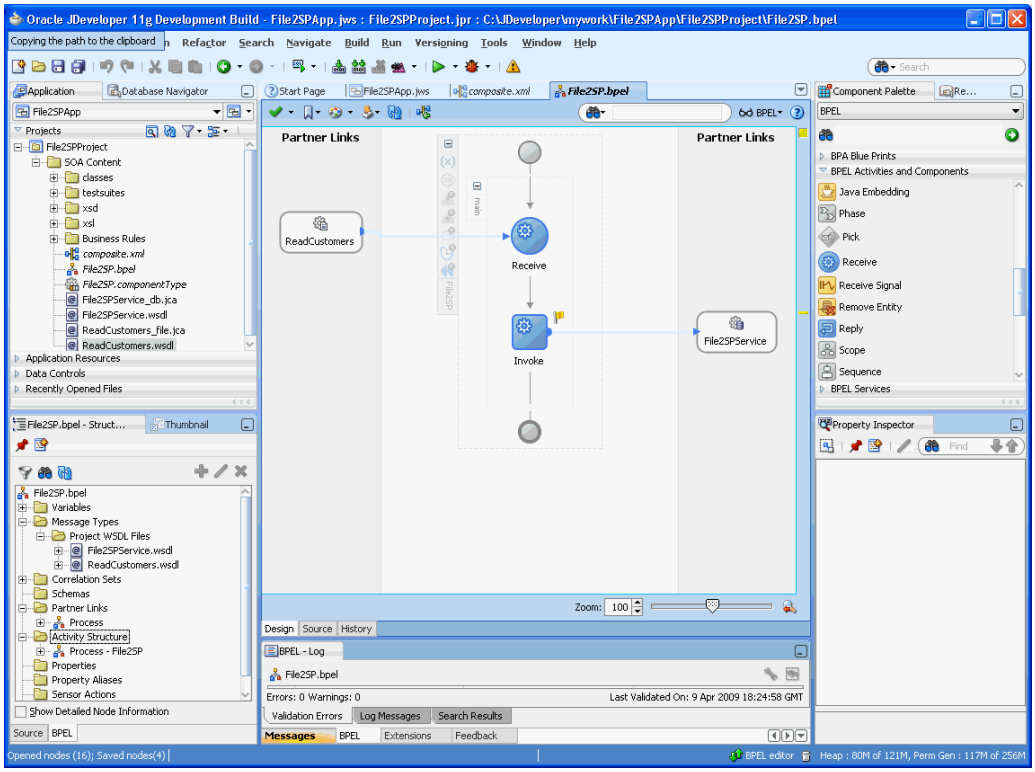
6. Select the default variable name and click **OK**.

   The Variable field is populated with the default variable name.

7. Select **Create Instance**, and click **OK**. The JDeveloper File2SP.bpel page is displayed.

   After adding the Receive activity, the JDeveloper window appears, as shown in Figure 9-69.

Figure 9-69 Adding a Receive Activity



8. Click **File**, **Save All**.

Adding an Assign Activity

Next, you must assign a value to the input parameter.

The following are the steps to add an Assign activity:

1. Drag and drop an **Assign** activity from the Component Palette in between the Receive and Invoke activities in the design area.

2. Double-click the **Assign** activity.

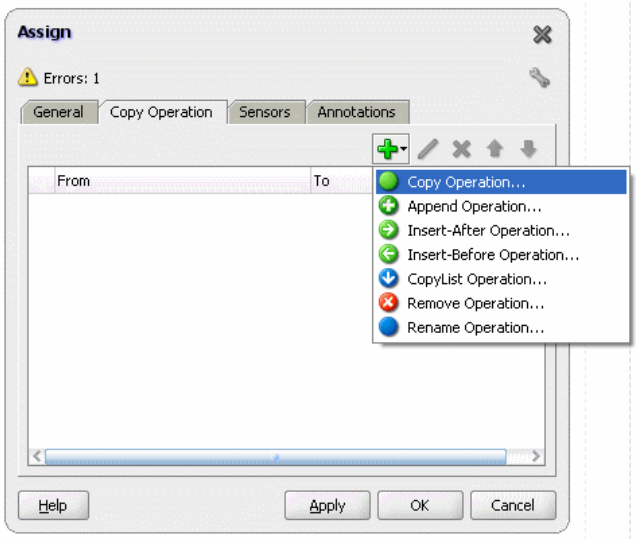
   The Assign dialog is displayed.

3. Click **General**, and then `CUSTOMER` in the **Name** field.

4. Click the **Copy Operation** tab.

   The Assign dialog is displayed, as shown in Figure 9-70.

Figure 9-70 The Assign Dialog - Copy Operation Tab



5. Click the icon with the plus sign, as shown in Figure 9-70, and then select **Copy Operation**.

   The Create Copy Operation dialog is displayed.

6. In the **From** area expand **Process**, **Variables**, **Receive_Read_InputVariable** and then body.

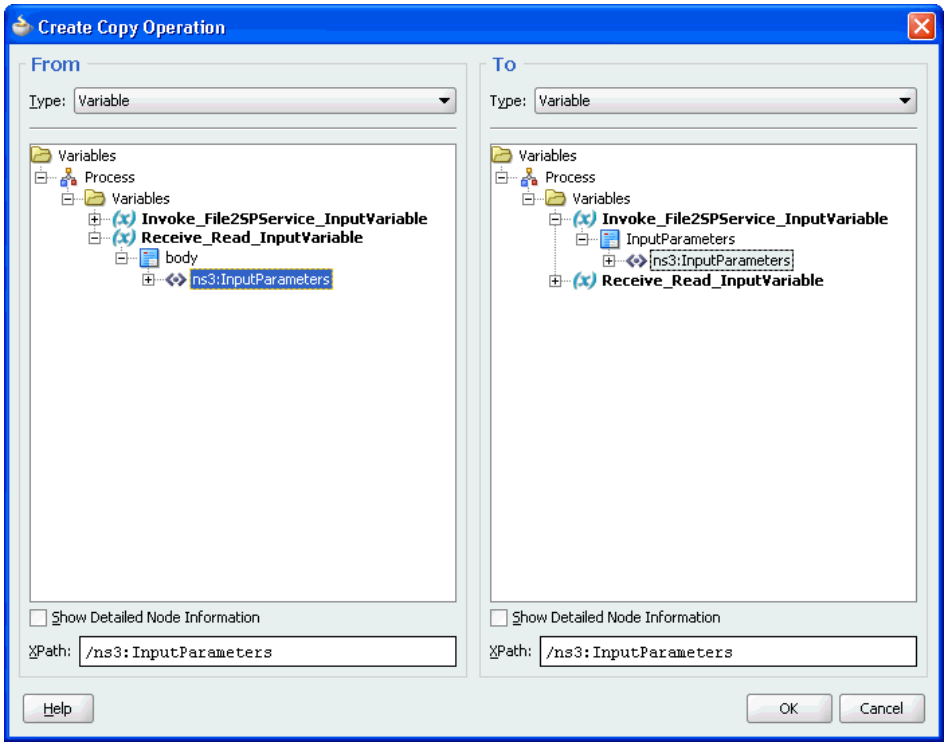
7. Select **ns3:InputParameters**.

8. In the **To** area expand **Process**, **Variables**, **Invoke_File2SPService_InputVariable**, and then **InputParameters**.

9. Select **ns3:InputParameters**.

10. Click **OK**. The Assign dialog is displayed, as shown in Figure 9-71.

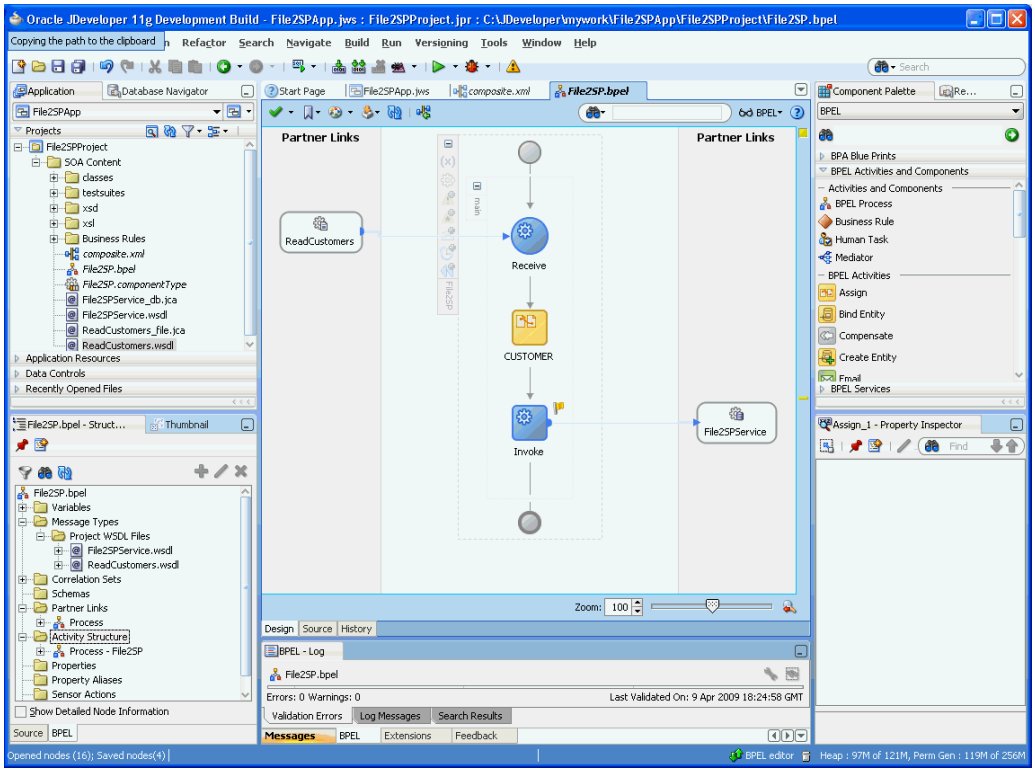
Figure 9-71 The Assign Dialog



11. Click **OK**.

The JDeveloper File2SP.bpel page is displayed, as shown in Figure 9-72.

Figure 9-72 The JDeveloper - File2SP.bpel



12. Click **File**, **Save All**.

Wiring Services and Activities

You must assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire components:

1. Drag the small triangle in `ReadCustomer` in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in `File2SPService` in the External References area.
3. Click **File**, **Save All**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, use the following steps:

1. Create an application server connection using the procedure described in Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application using the procedure described in Deploying Oracle JCA Adapter Applications from .

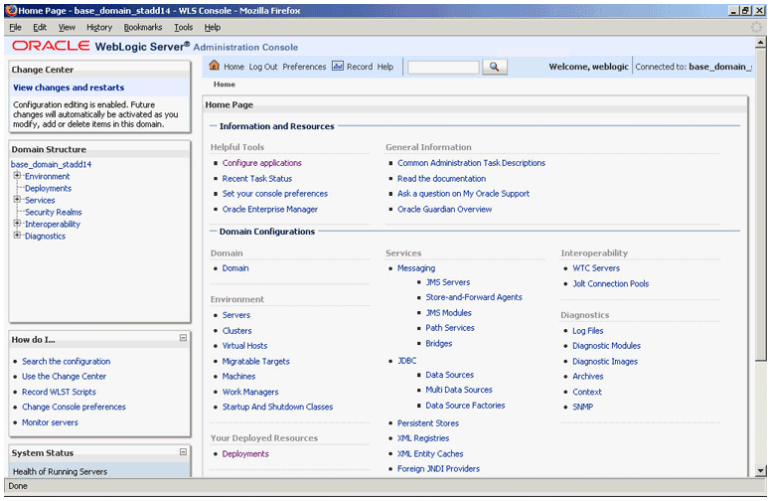
Creating a Data Source

Before you can test the `File2SPProject` you must create a data source using the Oracle WebLogic Server Administration Console, by using the following steps:

1. Navigate to `http://`*`servername`*`:`*`portnumber`*`/console`.
2. Use the required credentials to open the Home page of the Oracle WebLogic Server Administration Console.

   The Home page of the Oracle WebLogic Server Administration Console is displayed, as shown in Figure 9-73.

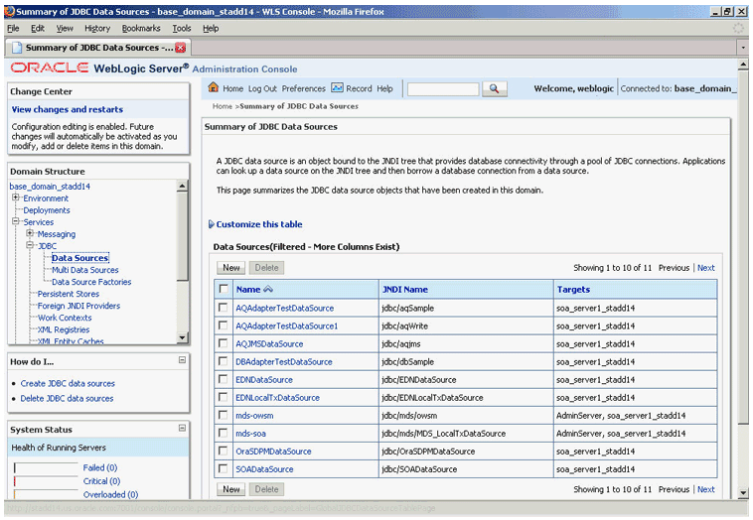
Figure 9-73 Oracle WebLogic Server Administration Console Home Page



3. Under Domain Structure, select **Services**, **JBDC**, and then click **DataSources**.

   The Summary of JDBC Data Sources page is displayed, as shown Figure 9-74.

Figure 9-74 The Summary of JDBC Data Sources Page



4. Click **New**.

   The Create a New JDBC Data Source page is displayed.

5. Enter the following values for the properties to be used to identify your new JDBC data source:

   - Enter `MyDataSource` in the **Name** field.
   - Enter `jdbc/MyDataSource` in the **JNDI Name** field.
   - Retain the default value `Oracle` for **Database Type**.
   - Retain the default value `Oracle's Driver (Thin XA) for Instance Connections; Versions 9.0.1, 9.2.0, 10, 11` for **Database Driver**.

6. Click **Next**.

The Create a New JDBC Data Source - Transaction Options page is displayed. A message stating, "No other transaction configuration options are available." is displayed.

7. Click **Next**.

   The Create a New JDBC Data Source - Connection Properties page is displayed.

8. Enter the following connection properties in the Connection Properties page:

   - Enter a name in the **Database Name** field, which is usually the SID.
   - Enter the host name in the **Host Name** field.
   - Enter the port number in the **Port** field.
   - Enter `SCOTT` in the **Database User Name** field.
   - Enter `TIGER` in the **Password** field.
   - Enter `TIGER` in the **Confirm Password** field.

9. Click **Next**. The Create a New JDBC Data Source - Test Database Connection page is displayed.

10. Click **Test Configuration** to test the database availability and the connection properties you provided. A message stating that the, "Connection test succeeded" is displayed at the top of the Create a New JDBC Data Source - Test Database Connection page.

11. Click **Next**.

    The Create a New JDBC Data Source - Select Targets page is displayed.

12. Select `AdminServer` as target, and then click **Finish**.

    The Summary of JDBC Data Sources page is displayed. This page summarizes the JDBC data source objects that have been created in this domain. The Data Source that you created appears in this list.

Adding a Connection-Instance

The database adapter needs an instance entry, which points to a data source.

The following are the steps to add a connection instance:

1. Search beahome/ for DbAdapter.rar.
2. Unzip the file.
3. Edit `META-INF/weblogic-ra.xml` (and possibly `ra.xml`.), as shown in the following example:

```
<connection-instance>
                <jndi-name>eis/DB/MyConnection</jndi-name>
                <connection-properties>
                    <properties>
                        <property>
                            <name>xADataSourceName</name>
                            <value>jdbc/MyDataSource</value>
                        </property>
```

```
                    </properties>
                </connection-properties>
            </connection-instance>
```

4. Use the same database connection name, **MyConnection**, for the JNDI name.
5. Use the same data source name, **MyDataSource**, as the `xADataSourceName`.
6. Jar the file again.
7. Restart the application server.

   You can also create a new database adapter instance using the Oracle WebLogic Server Administration Console.

Testing using the File Adapter Service and SQL*Plus

You must test the BPEL process by providing input file for the File Adapter. The results of the BPEL process are seen using a simple query from a table. The `customers.xml` file contains the following input:

```
<InputParameters xmlns="http://xmlns.oracle.com/pcbpel/adapter/db/SCOTT/
ADD_CUSTOMERS/">
  <C>
    <C_ITEM>
      <FNAME>John</FNAME>
      <LNAME>Doe</LNAME>
      <LOC>
        <STREET>123 Main Street</STREET>
        <CITY>Anytown</CITY>
        <STATE>CA</STATE>
        <ZIP>12345</ZIP>
      </LOC>
      <EMAIL>john.smith@gmail.com</EMAIL>
      <PHONE>567-123-9876</PHONE>
    </C_ITEM>
    <C_ITEM>
      <FNAME>Jane</FNAME>
      <LNAME>Doe</LNAME>
      <LOC>
        <STREET>987 Sunset Blvd</STREET>
        <CITY>Sometown</CITY>
        <STATE>CA</STATE>
        <ZIP>34567</ZIP>
      </LOC>
      <EMAIL>JaneDoe@yahoo.com</EMAIL>
      <PHONE>567-123-9876</PHONE>
    </C_ITEM>
  </C>
</InputParameters>
```

The following are the steps for testing the BPEL process you created:

1. Place a copy of `customers.xml` in the input directory that you specified when you created the inbound File Adapter Service.
2. The Oracle File Adapter polls the directory for new files. The Receive activity initiates the BPEL process once the file is received by the File Adapter Service.
3. The data for all of the customers is assigned to the InputParameters of the stored procedure.

4. The stored procedure executes. It transforms the data for each customer and then inserts the customer data into a table.
5. Query the table to see the following results:

```
SQL> select * from customers;

NAME                      LOC
------------------------- --------------------------------------------
EMAIL                     PHONE
------------------------- --------------
Doe, John                 123 Main Street, Anytown, CA 12345
john.smith@gmail.com      567-123-9876

Doe, Jane                 987 Sunset Blvd, Sometown, CA 34567
JaneDoe@yahoo.com         567-123-9876
```

Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed EM Composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`.

   A list of SOA composites is displayed, including `File2SPProject[1.0]` that you created in the preceding steps.
2. Click **File2SPProject[1.0]**.

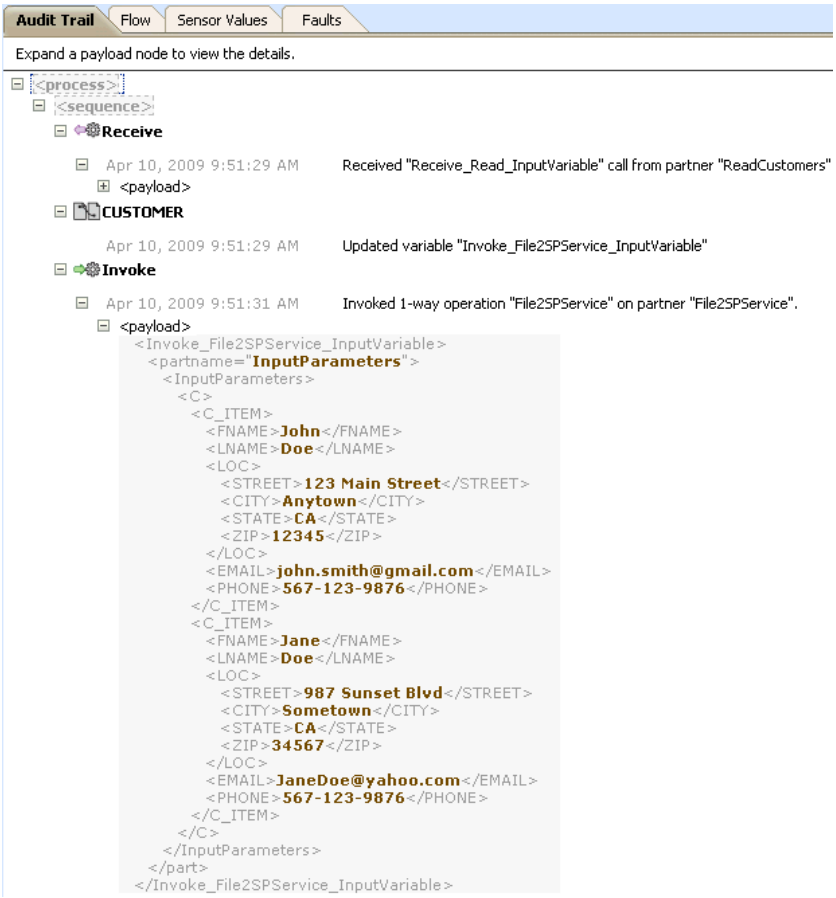
   The Dashboard is displayed. Note your Instance ID in the Recent Instances area.
3. Click the **Instances** tab.

   A Search dialog is displayed. The default search displays all instances by their Instance ID.
4. Select the **Instance ID** that you noted.

   A new window opens. It lists any faults (No faults found) and enables you to view the Audit Trail of your instance. Your instance trace is displayed in a new window.
5. The instance tree is expanded from **ReadCustomers** (service) to **File2SP** (BPEL Component) to **File2SPService** (reference).
6. Click **File2SP** BPEL Component.

   The Audit Trail of your process is displayed.
7. Expand the <payload> node to see the input provided to the stored procedure, as shown in Figure 9-75.

Figure 9-75 The Audit Trail Tab



8. Additionally, click the **Flow** tab to view the process flow, as shown in Figure 9-76.

Figure 9-76 Viewing the Process Flow

Database Adapter/Coherence Integration

There is a performance improvement when the Database Adapter is used with Coherence Cache on an Exalogic system. The feature that provides this improvement is called Database Adapter/Coherence Integration

There are two specific use cases where there is an advantage to using the Database Adapter with Coherence Cache on an Exalogic system. Specifically, performance can be improved when performing the following operations:

- Insert/Update to a database
- Select by primary key

Inserts/Updates to a Database

Inserts and updates to a database using the Database Adapter and Coherence cache are improved through the internal use of an *intermediary* Coherence data source, called a Coherence Cache, basically an in-memory database.

In the typical case, you perform insert/delete/update operations directly on the database. To improve performance, these operations can first be performed on this Coherence-fronting in-memory database, called a *write-behind map*, which enables read-write operations using the Cache.

Note:

When using the Database Adapter for outbound inserts with Coherence write-behind make sure the Database table has an index on the primary key column.

Using such a Coherence map improves the latency of BPEL/OSB processes performing insert/delete/update operations, as these processes can return immediately to the caller without a trip to the database; the actual and intensive work of updating the database is done instead by the Coherence Cache intermediary.

You can use Coherence in this manner when processing large batches of records, which makes record updating more convenient and efficient.

Note:

Database Adapter use cases **that do not** leverage Coherence Cache include the following operations: inbound polling, pure SQL invokes, stored procedure calls, and general Selects that return multiple rows.

Select Optimization

The second use case that Database Adapter/Coherence Integration improves is query performance, specifically in optimizing Select statement use cases. Database Adapter/Coherence Integration provides benefits to the query performance by caching data that might be accessed frequently by many different process instances.

When select optimization is used, to optimize queries, the Database Adapter/Coherence Integration uses a *read-only* Coherence Cache (also called an L2-read cache), which the Database Adapter checks first for a cache hit before proceeding to the database. In other words, queries are optimized by checking to see if the data being

queried against is in the Coherence Cache first; if not found there, the database is checked for the same data.

When a Coherence miss occurs, the data is read from the database and loaded into the Coherence Cache. The presumption is that checking the Coherence Cache is faster than executing a query on the database, as the ratio of cache visits to cache misses is typically high.

Queries that Do Not Benefit from Coherence Database Adapter Integration

Not all queries can benefit from cache visits and hence Coherence Database Adapter Integration.There is no indication if there was a Coherence cache hit on all records that meet a specified query criterion, or if there are additional database records that could have been hit but which were not in cache.

For this reason, the query optimization feature includes a new kind of Database Adapter operation, which is a *Select by Primary Key*. Unlike the existing *Select* and *queryByExample* operations, when using *Select by Primary Key* you can only return a single row. With the primary key selected to return a single row, you are in effect requesting more specific records to be returned from the Coherence Cache, thus improving the performance of the feature against the Cache.

Database Adapter/Coherence Integration Architecture

You can choose whether to use Database Adapter/Coherence Integration by making a simple choice among `none`, `read`, or `read-write` in the Operation Type screen of the Database Adapter Wizard. However, it is useful to know some of the background related to the architecture of the Database Adapter/Coherence Integration, as detailed in the following subsections.

For more information on Eclipselink, see http://www.oracle.com/technetwork/middleware/toplink/documentation/index.html

Some background on Coherence can be found at http://www.oracle.com/technetwork/middleware/coherence/overview/index.html

Using Coherence Database Adapter Integration with WebLogic Server 10.3.5

You must perform the following steps to use the Coherence Database Adapter with WebLogic Server 10.3.5.

1. Install Oracle 11.1.1.7.1 SOA Suite with WebLogic Server 10.3.5.
2. Find the `DbAdapter.rar` in your SOA install.

```
cd $BEAHOME/AS11gR1SOA/soa/connectors
mkdir tmp
```

3. Remove the bundled 10.3.6 version of `toplink-grid.jar` from `DbAdapter.rar`:

```
unzip DbAdapter.rar -d tmp
cd tmp
rm toplink-grid.jar
```

4. Rebuild `DbAdapter.rar` with its existing manifest, which looks for shared library toplink-grid

```
cd $BEAHOME/AS11gR1SOA/soa
cd $BEAHOME/AS11gR1SOA/soa/connectors
```

```
mkdir tmp
jar cvmf META-INF/MANIFEST.MF DbAdapter.rar *
mv DbAdapter.rar ..
cd ..
rm -rf tmp
```

5. Deploy $BEAHOME/modules/com.oracle.toplinkgrid_1.0.0.0_11-1-1-5-0.jar as a shared library named `toplink-grid` from the WebLogic Server Host console;

   From shared library (Deployments > Install) named 'toplink-grid'

6. Restart the WebLogic Server.

Current Design of the Database Adapter (No Coherence Cache)

With the current design of the Database Adapter, the Adapter performs selects and inserts to the EclipseLink layer, which directly communicates with the data sources without the Coherence Cache.

When you choose `none` in the Cache Usage dropdown on the Operations Type screen in the Database Configuration Wizard, you indicate you do not want to use cache.

Read-Write Coherence Cache Database Adapter Integration

You can choose to use read-write cache by choosing `read-write` from the Cache-Usage dropdown of the Operation Type screen of the Database Adapter Wizard

Eclipselink is in two layers, with Coherence Cache (a Coherence Cache Store) between the two layers. There is actually only one Eclipselink project, but two copies of that project.

- The top copy of Eclipselink redirects all insert/select queries from the data store to Coherence Cache.
- The bottom copy of Eclipselink handles requests by Coherence Cache to load a particular record by ID from the database, or to store a particular record to the database.

A Select you execute in the read-write scenario might not uniquely identify the rows to retrieve.

Such a case could be a `SELECT *` or `SELECT where total gross > ?`

The write-behind Coherence Cache can only receive requests to load a record by ID. Thus, in either of these cases, if all queries were directed to Coherence Cache, no results would be returned. In this case, the query proceeds to the data source directly, and then the Coherence Cache is updated.

Read Coherence Cache Database Adapter Integration

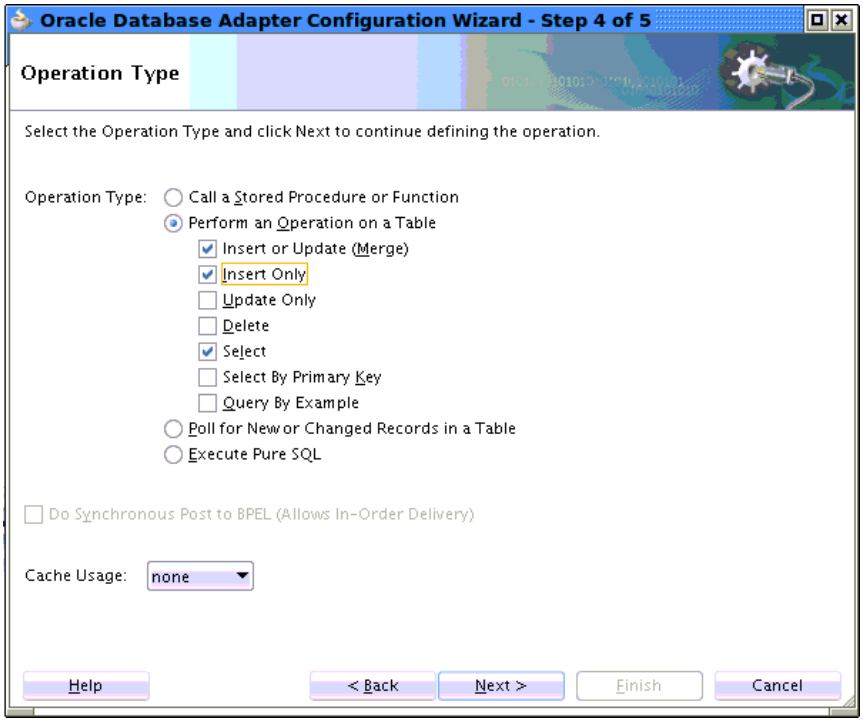
You can choose to use read cache by choosing `read` from the Cache Usage dropdown on the Operation Type screen of the Database Adapter Wizard.

With Read Cache, when the Database Adapter inserts a record to the Database or selects a record from the database, the Coherence Cache is updated. Any query that identifies a row (that is, by specifying primary key) first checks the Coherence cache, possibly saving a trip to the database.) As the Coherence Cache is distributed and can be simply thought of as a hash map, selecting by a specific primary key enables faster lookups through the Coherence Cache Map.

Enabling No Cache Using the Operations Type Screen

Figure 9-77 shows the No Caching option as it appears on the Operations type screen of the Database Adapter Wizard, with `none` selected.

Figure 9-77 The Database Adapter Configuration Wizard Operation Type Screen, with No Caching Selected



All outbound operations are enabled on this screen with the `none` option selected. Once you select this option, and choose `Next` or `Finish`, none of the selected operations contain the property `CacheUsage`. This absence of a property is equivalent to the JCA activation property `CacheUsage` being equal to the value `none`.

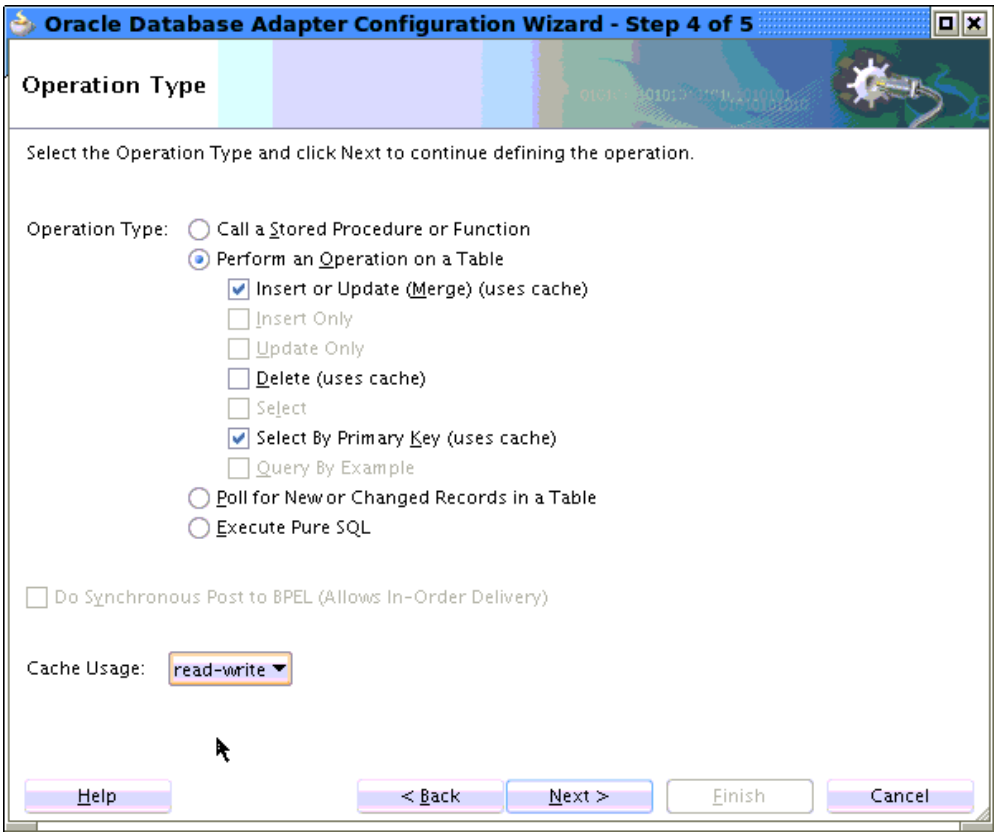
The following options are the only operations pre-selected when you choose the `none` option as the cache usage:

- `Merge`
- `Insert Only`
- `Select`

Enabling Read-Write Caching Using the Operation Type Screen

You can choose to enable read-write caching through the Operation Type screen. See Figure 9-78. Once you select this option and press `Next` or `Finish`, the JCA property `CacheUsage` value is set to `read-write`.

Figure 9-78 Enabling Read-Write Caching Using the Operation Type Screen



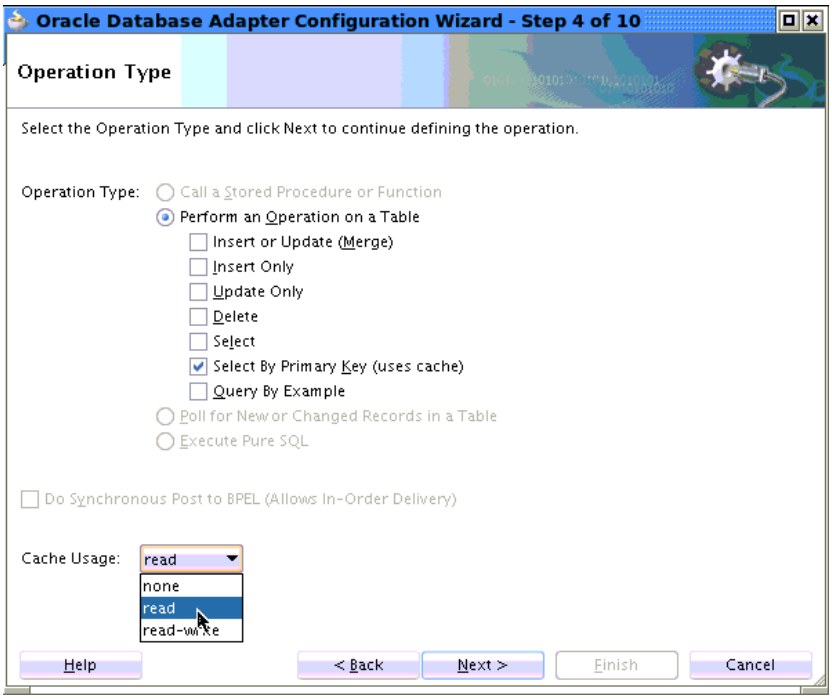
Refer to the following list of operations to understand how they are used on this screen when you choose the `read-write` option from the `Cache Usage` dropdown:

- `Insert or Update (Merge)` are enabled and have the string `uses cache` appended to their label.
- `Insert only` is disabled, as the underlying cache store always performs a merge.
- `Update Only` is disabled, as the underlying cache store always performs a merge.
- `Delete` is selectable but not pre-selected and has the string `uses cache` appended to it when it is selected.
- `Select` is disabled, as this query is converted into a Coherence filter executed on a Coherence map.
- `Query by Example` is disabled, as the Database Adapter/Coherence Integration query is converted into a Coherence filter executed on a Coherence map.
- `Select by Primary Key` has the string `uses cache` appended to the label.

Enabling Read Caching Using the Operation Type Screen

You can enable read caching using the `Cache Usage` option on the Operation Type screen. See Figure 9-79. Once you select this option on the screen and press `Next` or `Finish`, the JCA property `CacheUsage` value is set to `read`.

Figure 9-79 Enabling Read-Write Caching Using the Operation Type Screen



For the read cache option, only the `Select by Primary Key` operation is pre-selected. `Select by Primary Key` is the only operation that can be meaningfully executed by the Coherence Database Adapter Integration feature through the cache, although other operations can update cache. Because read cache is not intrusive on the cache, any of the operations on the Operation Type on this screen are disabled.

`Select` and `Query By Example` are not disabled, although they do not directly update the cache. The Database Adapter/Coherence Integration feature executes the `Select` against the database, but updates the Coherence Cache with any rows that are returned.

The general operation of read caching is that if any objects returned exist in the Coherence Cache, the *objects in the cache are returned*, rather than the Database Adapter/Coherence Integration feature building a *new copy* from the result set.

This operation improves performance where the master database record has several details; a query on the details does not have to be executed again.

A query behaves the same as a `Select`. This is true, for example with XML data where the primary key is set (and it does not get a cache hit).

XA Transactions, Read-Write and Read Operations with Coherence/Database Adapter Integration

When using Database Adapter/Coherence Integration, you cannot use **XA transactions with read-write operations.** This is because the Database Adapter, with Coherence Integration, performs inserts to the Coherence Cache and subsequently to the database, a sequence which breaks the XA transaction contract.

However, you **can** use XA transactions with **read** operations using Database Adapter/Coherence Integration.

Database transactions using the Database Adapter that do not use Database Adapter/Coherence Integration can still use the XA transaction model.

Coherence Cache Lifecycle and Configuration

When you deploy a Database Adapter containing a composite application with cacheUsage "read" or "read-write", a dedicated Coherence cache is created. Its name will be `DbAdapter/WriteBehindCache/<serviceName>/<tableName>` for read-write cache or `DbAdapter/L2Cache/<serviceName>/<tableName>` for read cache.

This cache name is stored in a property in `or-mappings.xml`:

```
<properties>
<property name="eclipselink.coherence.cache.name">
<value>DBAdapter/WriteBehindCache/insertReference.Movies</value>
</property>
</properties>
```

Two Cache configuration templates are defined in `soa-coherence-cache-config.xml`, which resides in `fabric-runtime.jar`. One template is for all cache names beginning with `DbAdapter/WriteBehindCache/` (read-write) and another for those starting with `DbAdapter/L2Cache` (read).

You can edit these definitions or change the cache name within `or-mappings.xml` and create new definitions. The following example shows two templates as defined in `soa-coherence-cache-config.xml`:

Example - soa-coherence-cache-conifg.xml Templates

```
<cache-mapping>
 <cache-name>DBAdapter/WriteBehindCache/*</cache-name>
   <scheme-name>db-adapter-write-behind-cache</scheme-name>
</cache-mapping>
<cache-mapping>
  <cache-name>DBAdapter/L2Cache/*</cache-name>
  <scheme-name>db-adapter-l2-cache</scheme-name>
</cache-mapping>

<distributed-scheme>
<scheme-name>db-adapter-write-behind-cache</scheme-name>
<backup-count-after-write-behind>0</backup-count-after-write-behind>
<!-- for DbAdapter must be true on SOA nodes and false on dedicated Coherence
nodes.-->
<local-storage>true</local-storage>
<thread-count>1</thread-count>
<task-hung-threshold>20000</task-hung-threshold>
<backing-map-scheme>
<read-write-backing-map-scheme>

<internal-cache-scheme>
<local-scheme>
<eviction-policy>HYBRID</eviction-policy>
<high-units>10000</high-units>
<low-units></low-units>
<unit-calculator>FIXED</unit-calculator>
<expiry-delay>120s</expiry-delay>
</local-scheme>
</internal-cache-scheme>
<cachestore-scheme>
<class-scheme>
<class-name>oracle.tip.adapter.db.toplinkext.coherence.DBAdapterCacheStore</class-
name>
```

```
<init-params>
<init-param>
<param-type>java.lang.String</param-type>
<param-value>{cache-name}</param-value>
</init-param>
</init-params>
</class-scheme>
</cachestore-scheme>
<write-delay>5s</write-delay>
<write-batch-factor>0.1</write-batch-factor>
<write-requeue-threshold>1000</write-requeue-threshold>
</read-write-backing-map-scheme>
</backing-map-scheme>
<autostart>false</autostart>
</distributed-scheme>

<distributed-scheme>
<scheme-name>db-adapter-l2-cache</scheme-name>
<!-- for DbAdapter must be true on SOA nodes and false on dedicated Coherence
nodes.-->
<local-storage>true</local-storage>
<thread-count>4</thread-count>
<task-hung-threshold>500</task-hung-threshold>
<backing-map-scheme>
<local-scheme>
<eviction-policy>HYBRID</eviction-policy>
<high-units>1000</high-units>
<low-units>1000</low-units>
<unit-calculator>FIXED</unit-calculator>
<expiry-delay>120s</expiry-delay>
</local-scheme>

</backing-map-scheme>
<autostart>true</autostart>
</distributed-scheme>
</caching-schemes>
```

If you are defining your own definition, you must set `local-storage true`.

The lifecycle of objects is different with a CacheUsage-enabled project. First, the EclipseLink project (`or-mappings.xml`) associated with a composite application is only loaded once, even if the same composite is redeployed with an updated `or-mappings.xml` file. Normally, each redeployment of a composite is a clean redeploy of all artifacts.

Secondly, a Coherence cache is not destroyed on undeploying a CacheUsage composite. This must be done manually. This means that you can redeploy a composite and the underlying Coherence cache will still be there, with the contents it had before the redeployment. As distinct from other Database Adapter use cases, you can thus have multiple Database Adapter references, across one or multiple composites, connecting to the same Coherence-named cache.

Thus, the lifecycle of the EclipseLink project (`or-mappings.xml`) and Coherence cache is per WebLogic Application server restart instead of per composite revision/deployment.

Finally, the Database Adapter uses byte-code generation rather than real Java classes to represent the objects being inserted into the data store. This makes it difficult to connect directly to the same Coherence NamedCache programatically, outside the Database Adapter. This is because the class definitions are not easily available for

deserialization. As mentioned, however, multiple Database Adapter references with similar `or-mappings.xml` projects can share the same Coherence cache.

Query by Example

Query-by-example enables you to specify query selection criteria in the form of a sample object instance that you populate with only the attributes you want to use for the query. EclipseLink support for query-by-example includes a query-by-example policy that a user can modify. The Eclipse link user can edit the policy to modify query-by-example default behavior.

The Database Adapter provides, in essence, an implementation or encapsulaion of a Query by Example policy by providing a basic set of Query By Example operators,. These include the following types of operators:

- Use LIKE or other operations to compare attributes. By default, query-by-example allows only EQUALS.
- Modify the set of values query-by-example ignores (the IGNORE set). The default ignored values are zero (0), empty strings, and FALSE
- Force query-by-example to consider attribute values, even if the value is in the IGNORE set.
- Use isNull or notNull for attribute values. (But refer to Constraints on Use).

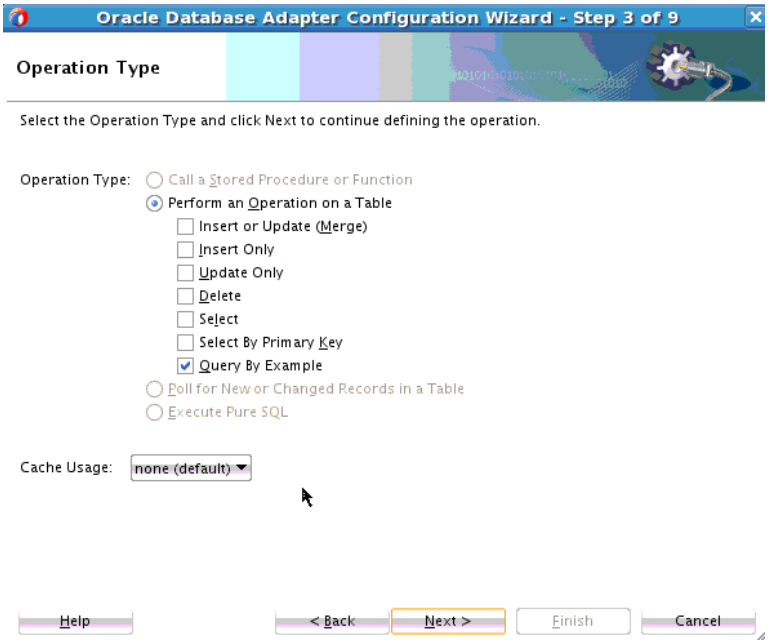
The Database Adapter provides this facility by adding additional operation and values to the existing operations and values available through the Advanced Operations screen of the Database Adapter Configuration Wizard.

When you select the Query by Example option in the Database Adapter Configuration Wizard, you have the option to use additional queries, or to combine those queries with other queries you have created.

Note:

QueryByExample does not support use of multiple table. It is better to use PureSQL over Query-by-example for related tables.

Figure 9-80 Database Adapter Configuration Wizard Operation Type Screen Showing Query By Example Chosen



Within the following types of queries, which are db.jca.properties added to `DBReadinteractionSpec`, you can use the following queries as indicated:

- `QueryByExampleTextOperator`
- `QueryByExampleNumericOperator`
- `QuerybyExampleDateOperator`

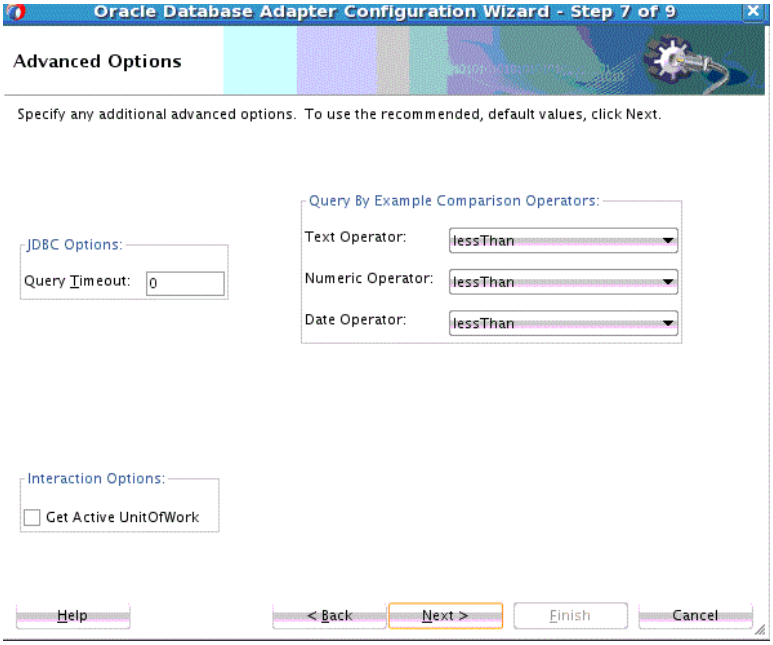
Each one of these types of queries can be set to one of the following Query By Example values:

- `equal`
- `notEqual`
- `equalsIgnoreCase`
- `lessThan`
- `lessThanEqual`
- `greaterThan`
- `greaterThanEqual`
- `like`
- `likeIgnoreCase`
- `containsAllKeyWords`
- `containsAnyKeyWords`
- `containsSubstring`

- `containsSubstringIgnoringCase`

In the JDeveloper UI, you can choose any of these queries and the operation they can perform. There is a drop down list of comparison operations for each of these operators on the Advanced Operations screen. See Figure 9-81.

Figure 9-81 Database Adapter Configuration Wizard Advanced Operations Screen Showing Query By Example Comparison Operator lessThan Specified



Note that some combinations, such as `likeIgnoreCase` for Numeric values, do not make sense, but no restriction is imposed on your use.

You can pick any single parameter operator which can be found in `org.eclipse.persistence.expressions.Expression`, not just the ones listed. To set one of these operators you can edit the JCA file directly.

Choosing "equal" is the same as the default, so that value will not appear in the JCA file when you select finish. Older projects which do not have these properties will have them added automatically.

Combining Query by Example with a Regular Query

You can combine your use of the JDev expression builder with Query by Example. For example, if your query performs a `WHERE some_column IS NOT NULL` operation you can model this first operation with a regular `Select`, and also choose the Query by Example operation. In your JCA file, copy the `QueryName` property from the Select operation to the Query by Example operation, that is:

```
<property name="IsQueryByExample" value="true"/>
<property name="QueryName" value="dbReferenceSelect"/>
```

Note you do no need to copy operations to the activation file, unlike with a Query by Example operation

Constraints on Use

If your example record has the following:

```
<someOtherColumn>someOtherValue</someOtherColumn>
```

the SQL produced will be the following:

```
WHERE some_column IS NOT NULL AND some_other_column = 'someOtherValue'
```

You can only use this if there are no input parameters to the regular Select statement.

There is currently no support for a QueryByExample policy where certain columns in the example object can have nulls which are significant.

Thus null attributes are ignored, but EclipseLink support allows generation of IS NULL and IS NOT NULL through a combination of two other APIs. Neither of these APIs are exposed, but you can use these statements by combining them with a regular query. There is also no API to always exclude certain columns, even if they have significant (that is, not null) values. It is better to create an explicit query.

Modifying the or-mappings.xml File for UTF16 Character Data Insertions

When the Database Adapter is used on a UTF16 to insert data onto a database using non AL16UTF16 character set, for example WE8DE, automatic character conversion causes wrong data to be inserted into the database. The solution is to modify the JDeveloper Adapter Configuration Wizard generated -or-mappings.xml file. One restriction is that the Oracle database version you are using has to be Oracle 9 database or above.

To accomplish the indicated solution:

1. Locate the -or-mappings.xml file in the project's file system and open it with an editor (by default you cannot alter the file using JDeveloper.)
2. Locate the `<attribute-name>field name</attribute-name>` for the database field which will contain UTF16 characters.
3. Replace the attribute

   ```
   <attribute-classification>java.lang.String</attribute-classification>
   ```

   with

   ```
   <attribute-classification>org.eclipse.persistence.platform.database.oracle.
   NString</attribute-classification>
   ```

4. Save and redeploy the project.

Because the -or-mappings.xml file s generated automatically by the Database Adapter Configuration Wizard, you must repeat these steps when you run the Database Adapter Configuration Wizard.

10

Oracle JCA Adapter for MQ Series

This chapter describes how to use the Oracle JCA Adapter for MQ Series (Oracle MQ Series Adapter), which works with Oracle BPEL Process Manager (Oracle BPEL PM) and Oracle Mediator (Mediator) as an external service. The chapter describes JCA Adapter for MQ Series concepts, features, configuration and use cases.

This chapter includes the following sections:

MQ Series Message Queuing Concepts

Message queuing is a technique for asynchronous program-to-program communication. It enables application integration by allowing independent applications on a distributed system to communicate with each other. One application sends messages to a queue owned by a queue manager, and another application retrieves the messages from the queue. The communication between applications is maintained even if the applications run at different times or are temporarily unavailable.

The basic concepts of message queuing are described in the following list:

- Messaging

  Messaging is the mechanism that allows two entities to communicate by sending and receiving messages. Messaging can be of two types, synchronous and asynchronous. In *synchronous* messaging, the sender of the message places a message on a message queue and then waits for a reply to its message before resuming its own processing. In *asynchronous* messaging, the sender of the message proceeds with its own processing without waiting for a reply.

- Message

  Messages are structured data sent by one program and intended for another program.

- Message Queue

Message queues are objects that store messages in an application. Applications can put messages to the queues and get messages from the queues. A queue is managed by a queue manager.

- Queue Manager

  A queue manager provides messaging and queuing services to applications through an application programming interface. It provides you with access to the queues and also transfers messages to other queue managers through message channels.

- Message Channel

  A message channel provides a communication path between two queue managers. It connects queue managers. A message channel can transmit messages in one direction only.

- Transmission Queue

  A transmission queue is used to temporarily store messages that are destined for a remote queue manager.

- Message Segment

  If a message is very large, then it can be divided into multiple small messages, called segments. Each segment has a group ID and an offset. All segments of a message have the same group ID. The last segment of the message is marked with a flag.

- Message Group

  A message group consists of a set of related messages with the same group ID. Each message in a message group has a message sequence number. The last message in a message group is marked with a flag.

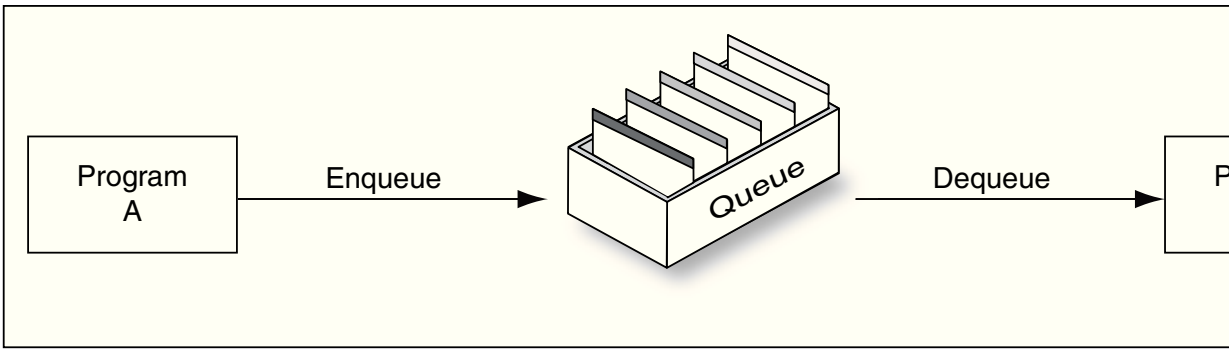
- Cluster

  A cluster is a group of queue managers that are logically associated.

- Enqueue/Dequeue

  To enqueue is to put a message in a queue whereas to dequeue is to get a message from a queue, as shown in Figure 10-1.

Figure 10-1 Enqueue/Dequeue

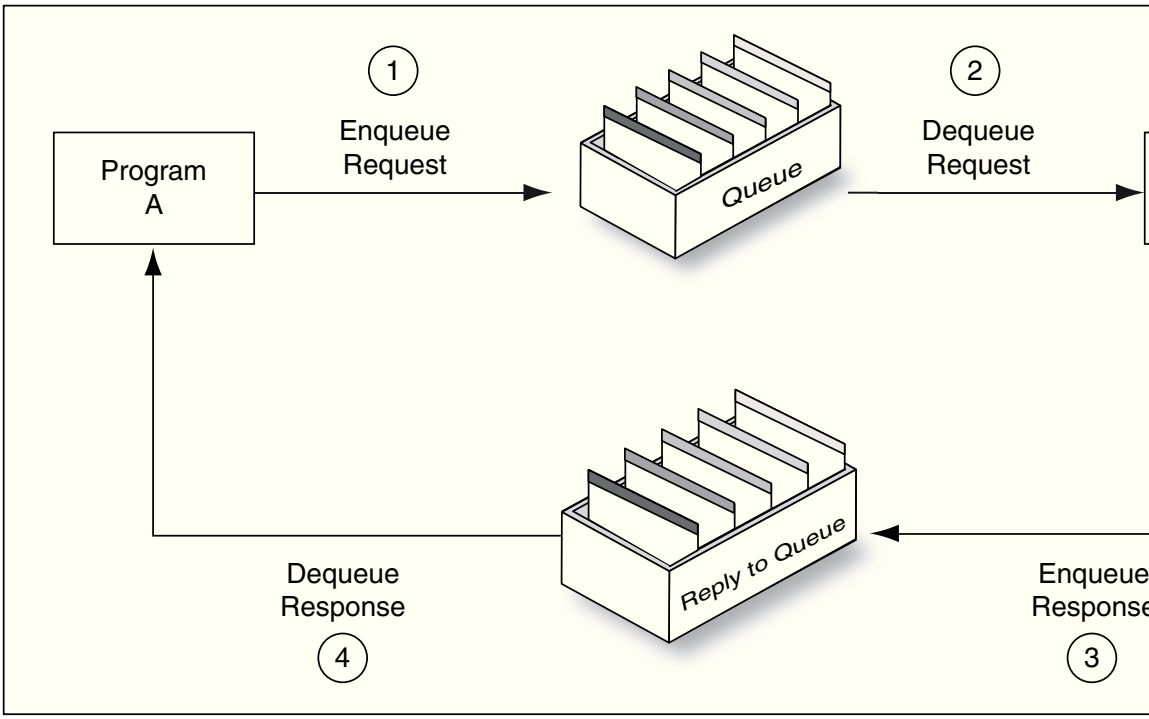


- Request/Response

  In a request/response interaction, a program sends a message to another program asking for a reply. The request message contains information about where the reply

should be sent. The receiving program sends a reply message in response to the request message. The request/response interaction is shown in Figure 10-2.

Figure 10-2 Request/Response Interaction



For more information about the interaction scenarios supported by the Oracle MQ Series Adapter, see Dequeue Message.

MQ Series Concepts

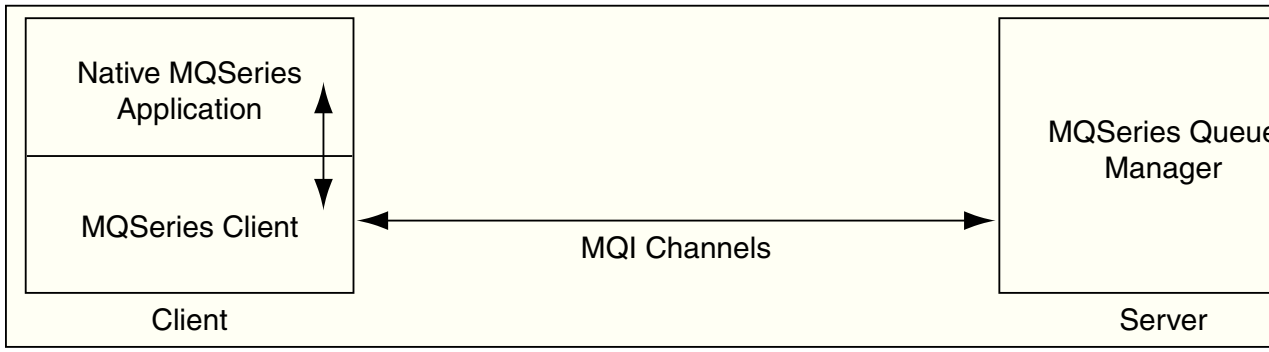
Messaging and Queuing Series (MQ Series) is a set of products and standards developed by IBM. MQ Series provides a queuing infrastructure that provides guaranteed message delivery, security, and priority-based messaging.

Note:

The Oracle MQ Series Adapter is certified on IBM MQ V7.5.

The communication process between an MQ Series application and an MQ Series server is shown in Figure 10-3. An MQ Series client enables an application to connect to a queue manager on a remote computer.

Figure 10-3 The MQ Series Communication Process



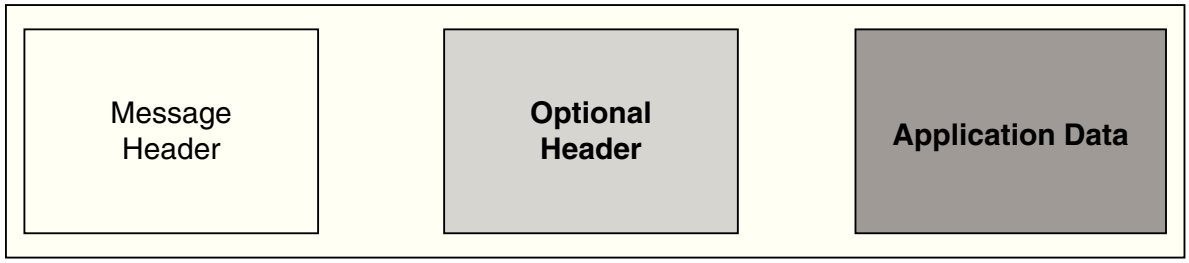
Every queue in MQ Series belongs to a queue manager. A queue manager has a unique name and provides messaging and queuing services to applications through a Message Queue Interface (MQI) channel. A queue manager also provides access to the queues created on it and transfers messages to other queue managers through message channels.

In MQ Series, data is sent in the form of messages. The sending application constructs a message and sends it to a queue by using API calls. The message remains in the queue until the receiving application is ready to receive it. The receiving application gets the messages from the queue by using API calls.

For sending messages to a remote queue, the remote queue definition must be defined locally. The remote queue definition consists of the destination queue name and the transmission queue name.

Figure 10-4 displays the message structure of an MQ Series message.

Figure 10-4 MQ Series Message



An MQ Series message consists of the following parts, as shown in Figure 10-4:

- Message Header

  The message header contains information such as unique message ID, message type, message priority, and routing information. Every MQ Series message must have a message header.

- Optional Header

  The optional header is required for communication with specific applications, such as the CICS application.

  For more information, see Integration with CICS.

- Application Data

  This contains the actual data, for example, a record from an indexed or flat file or a row or column from a DB2 table.

Note:

Important troubleshooting: Inbound MQ processes must not use the same queue unless mutually exclusive MessageSelectors have been defined for each inbound process. If a MessageSelector is not defined, or the selection criteria for two or more inbound process overlap, issues might occur due to one process reading messages that are meant for another process.

Also, the following message found in the MQ Series Adapter log is not an error. It means that no message is available on the queue.

```
com.ibm.mq.MQException: MQJE001: Completion Code 2, Reason
2033 at com.ibm.mq.MQQueue.get(MQQueue.java:1033)
```

Introduction to Native Oracle MQ Series Adapter

Oracle BPEL Process Manager and Oracle Mediator and Mediator include the Oracle MQ Series Adapter. The Oracle MQ Series Adapter enables applications to connect to MQ Series queue managers and place MQ Series messages on queues or to remove MQ Series messages from queues.

This section contains the following topics:

- The Need for Oracle MQ Series Adapter
- Oracle MQ Series Adapter Integration with Oracle BPEL Process Manager and Oracle Mediator
- Oracle MQ Series Adapter Integration with Mediator

Note:

To prevent the MQ Series Adapter's test connection thread from being blocked forever, ensure that the `KeepAlive` option is enabled on the MQ Series queue manager. See IBM MQ documentation for more details on how to set the `KeepAlive` option.

The Need for Oracle MQ Series Adapter

The Oracle MQ Series Adapter provides all native MQ Series functionalities. Although you can configure the Oracle JCA Adapter for JMS (Oracle JMS Adapter) with MQ Series provider, it provides only the JMS functionalities provided by MQ Series and not the native MQ Series functionalities. The following list explains the advantages of Oracle MQ Series Adapter over the Oracle JMS Adapter:

- The Oracle MQ Series Adapter supports Positive Action Notification (PAN) and Negative Action Notification (NAN).
- The Oracle MQ Series Adapter supports report messages such as confirmation on delivery, confirmation on arrival, exception report, and expiry report.
- The Oracle MQ Series Adapter supports sending unwanted or corrupted messages to a dead-letter queue.
- The Oracle MQ Series Adapter provides advanced filter options, such as filtering message belonging to a group.

- The Oracle MQ Series Adapter is faster and easier to use.

Note:

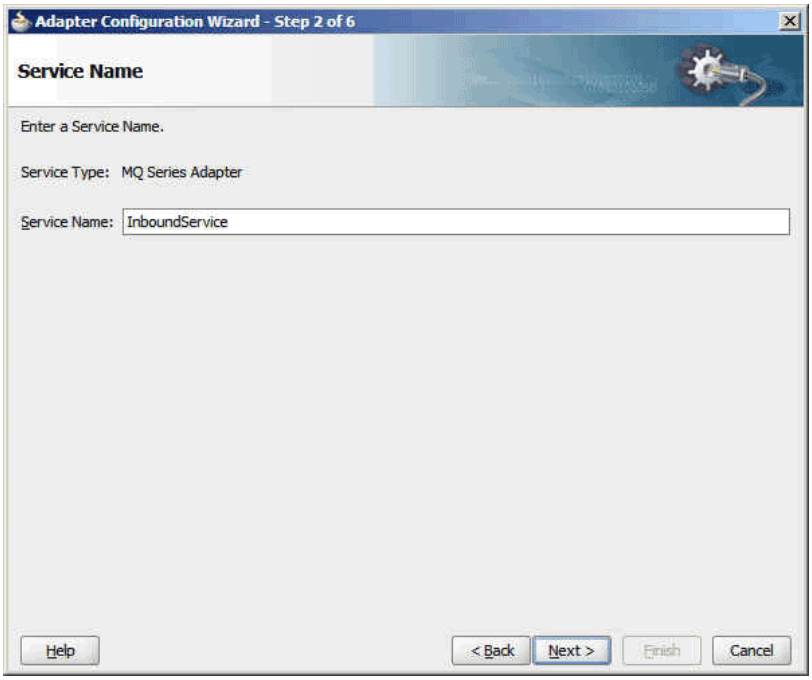
The MQ Series version that the Oracle MQ Series Adapter is certified is 7.5 version, both on Windows and Linux.

Oracle MQ Series Adapter Integration with Oracle BPEL Process Manager and Oracle Mediator

The Oracle MQ Series Adapter is automatically integrated with Oracle BPEL Process Manager and Oracle Mediator. When you create a partner link or an MQ adapter service in Oracle JDeveloper (JDeveloper), the Adapter Configuration Wizard is started.

This wizard enables you to select and configure the Oracle MQ Series Adapter or other Oracle JCA Adapters. The Adapter Configuration Wizard then prompts you to enter a service name, as shown in Figure 10-5.

Figure 10-5 The Service Name Page



When the configuration is complete, a WSDL file of the same name is created in the Application Navigator section of JDeveloper. This WSDL file contains the configuration information you specify with the Adapter Configuration Wizard.

The Operations page of the Adapter Configuration Wizard prompts you to select an operation to perform. Based on your selection, different Adapter Configuration Wizard pages appear and prompt you for configuration information.

Table 10-1 lists the available operations and provides references to sections that describe the information about these operations.

Table 10-1 Supported Operations for Oracle BPEL Process Manager

Operation	See Section...
Enqueue Message	Enqueue Message
Dequeue Message	Dequeue Message
Request-Response	Asynchronous Request-Response (As Client) Synchronous Request-Response (As Server) Asynchronous Request-Response (As Server) Synchronous Request-Response (As Server) Synchronous Request-Response (As Client) Synchronous Request-Response (Oracle Mediator as Client) Asynchronous Request-Response (Oracle Mediator As Client)
Outbound Dequeue	Outbound Dequeue Scenario

Oracle MQ Series Adapter Integration with Mediator

The Oracle MQ Series Adapter is automatically integrated with Mediator. When you create an MQ adapter service in JDeveloper Mediator Designer, the Adapter Configuration Wizard is started.

This wizard enables you to select and configure the Oracle MQ Series Adapter. When the configuration is complete, a WSDL file of the same name is created in the Application Navigator section of JDeveloper. This WSDL file contains the configuration information you specify in the Adapter Configuration Wizard.

The Operations page of the Adapter Configuration Wizard prompts you to select an operation to perform. Based on your selection, different Adapter Configuration Wizard pages appear and prompt you for configuration information. Table 10-2 lists the available operations and provides references to sections that describe the configuration information you must provide.

Table 10-2 Supported Operations for Oracle Mediator

Operation	See Section...
Enqueue Message	Enqueue Message
Dequeue Message	Dequeue Message
Request-Response	Synchronous Request-Response (As Server) Synchronous Request-Response (Oracle Mediator as Client) Asynchronous Request-Response (Oracle Mediator As Client)
Outbound Dequeue	Outbound Dequeue Scenario

Oracle MQ Series Adapter Features

This section explains the following features of the Oracle MQ Series Adapter:

RFH Version 2 (RFH2) Header

The RFH2 header is an extensible header. The RFH2 header enables you to add more header properties to the payload. The RFH2 header carries JMS-specific data that is associated with the message content and can also carry additional information that is not directly associated with JMS.

The RFH2 header consists of two parts, a fixed portion and a variable portion. There can be multiple RFH2 headers in the same message.

Fixed Portion

The fixed portion is modeled on the `standard` MQ header pattern and consists of the following fields:

StrucId (MQCHAR4)

Structure identifier.

Must be MQRFH_STRUC_ID (value: "RFH ") (initial value).

MQRFH_STRUC_ID_ARRAY (value: "R","F","H"," ") is also defined in the usual way.

Version (MQLONG)

Structure version number.

Must be MQRFH_VERSION_2 (value: 2) (initial value).

StrucLength (MQLONG)

Total length of MQRFH2, including the NameValueData fields.

The value set into StrucLength must be a multiple of 4 (the data in the NameValueData fields may be padded with space characters to achieve this).

Encoding (MQLONG)

Data encoding.

Encoding of any numeric data in the portion of the message following MQRFH2 (the next header, or the message data following this header).

CodedCharSetId (MQLONG)

Coded character set identifier.

Representation of any character data in the portion of the message following MQRFH2 (the next header, or the message data following this header).

Format (MQCHAR8)

Format name.

Format name for the portion of the message following MQRFH2.

Flags (MQLONG)

Flags.

MQRFH_NO_FLAGS =0. No flags set.

NameValueCCSID (MQLONG)

The coded character set identifier (CCSID) for the NameValueData character strings contained in this header. The NameValueData may be coded in a character set that differs from the other character strings that are contained in the header (StrucID and Format).

If the NameValueCCSID field is a 2-byte Unicode CCSID (1200, 13488, or 17584), then the byte order of the Unicode CCSID is the same as the byte ordering of the numeric fields in MQRFH2. (For example, Version, StrucLength, and NameValueCCSID itself.)

The NameValueCCSID field may take only values from Table 10-3:

Table 10-3 Possible Values for NameValueCCSID Field

Value	Meaning
1200	UCS2 open-ended
1208	UTF8
13488	UCS2 2.0 subset
17584	UCS2 2.1 subset (includes the Euro symbol)

Variable Portion

The variable portion follows the fixed portion. The variable portion contains a variable number of MQRFH2 folders. Each folder can occur multiple times in the same RFH2 header. Other folders such as `mqext`, `mq_usr`, `mqps` and others. can also be part of the RFH2 header. For more information, refer to the IBM documentation regarding MQ RFH2 headers.

The related properties are grouped together. The MQRFH2 header can contain the following message service folders:

The <mcd> folder

This contains properties that describe the shape or format of the message. For example, the Msd property identifies the message as being Text, Bytes, Stream. Map, Object, or Null. This folder is always present in JMS MQRFH2.

The <jms> folder

This is used to transport JMS header fields, and JMSX properties that cannot be fully expressed in the MQMD. This folder is always present in a JMS MQRFH2.

The <usr> folder

This is used to transport any application-defined properties associated with the message. This folder is only present if the application has set some application-defined properties.

The <psc> folder

This is used to convey publish/subscribe command messages to the broker. Only one psc folder is allowed in the NameValueData field.

The <pscr> folder

This is used to contain information from the broker, in response to publish/subscribe command messages. Only one pscr folder is present in a response message.

Table 10-4 shows a full list of property names.

Table 10-4 MQRFH2 Folders and Properties Used by JMS

JMS Field Name	Java Type	MQRFH2 Folder name	Property Name	Type/values
JMSDestination	Destination	jms	Dst	string
JMSExpiration	long	jms	Exp	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	String	jms	Cid	string
JMSReplyTo	Destination	jms	Rto	string
JMSTimestamp	long	jms	Tms	i8
JMSType	String	mcd	Type, Set, Fmt	string
JMSXGroupID	String	jms	Gid	string
JMSXGroupSeq	int	jms	Seq	i4
xxx (User Defined)	Any	usr	xxx	any
-	-	mcd	Msd	jms_none
-	-	mcd	Msd	jms_text
-	-	mcd	Msd	jms_bytes
-	-	mcd	Msd	jms_map
-	-	mcd	Msd	jms_stream
-	-	mcd	Msd	jms_object

The syntax used to express the properties in the variable portion is as follows:

NameValueLength (MQLONG)

Length, in bytes, of the NameValueData string that immediately follows this length field. It does not include its own length. The value set into NameValueLength is always a multiple of 4. The NameValueData field is padded with space characters to achieve this.

NameValueData (MQCHARn)

A single character string, whose length in bytes is given by the preceding NameValueLength field. It contains a folder holding a sequence of properties. Each property is a name/type/value triplet, contained within an XML element whose name is the folder name, as follows:

<foldername> triplet1 triplet2 ..... tripletn </foldername>

SSL Enabling

Secure Sockets Layer (SSL) is a protocol for transmitting encrypted data over the Internet or an internal network. SSL works by using public and private keys to encrypt data that is transferred over the SSL connection. Data that has been encrypted with a public key can be decrypted only with the corresponding private key. Conversely, data that has been encrypted with a private key can be decrypted only with the corresponding public key.

MQ Series supports secure communication, with MQ Series clients using SSL. As a part of this functionality, the adapter would provide support to put a message on queue using SSL. To enable Oracle MQ Series Adapter for SSL, the following properties must be provided:

- **SSLEnable**: The true/false value for this property means that the Oracle MQ Series Adapter is SSL enabled/disabled.
- **KeyStoreLocation**: This is the keystore where Oracle MQ Series Adapter has its private keys. This property is required as the adapter must authenticate itself to the MQ Series server.
- **KeyStorePassword**: This password is required to access keystore.
- **TrustStoreLocation**: This is the location where the adapter keeps its trusted certificates information. This information is required when an adapter must authenticate to the MQ Series server.
- **Protocol**: Key Management Algorithm.
- **KeyStoreProviderName**: The name of the keystore provider.
- **KeyStoreType**: Type of the key store.
- **KeyStoreAlgorithm**: Algorithm used by the key store.
- **CipherSuite**: Set CipherSuite to the name matching the CipherSpec set on the `SVRCONN` channel. If set to null (default), then no SSL encryption is performed.
- **SSLPeerName**: A distinguished name pattern. If CipherSuite is set, then you can use this variable to ensure that the correct queue manager is used. If set to null (default), then the DN of the queue manager is not checked. This variable is ignored if `sslCipherSuite` is null.

XA Transactions

Note: According to IBM, the use of WebSphere MQ Java Transaction API (JTA) by a non WebSphere MQ transaction manager is not supported. IBM recommends that the WebSphere MQ JMS API be used instead in XA transactions. This means that the user must use Oracle JCA Adapter for JMS when XA transactions are required to interact with IBM MQ instead of Oracle JCA Adapter for MQ Series. If the MQ adapter is used for XA transaction, the adapter might encounter an error when it needs to roll back an XA transaction. The user will then need to manually resolve any in-doubt transactions.

The Oracle MQ Series Adapter enables transaction support, which along with the inherent data processing, ensures that each modification has a clearly defined outcome, resulting in either success or failure, thus preventing potential corruption of data, executes independently from other changes, and, after completion, leaves underlying data in the same state until another transaction takes place.

The Oracle MQ Series Adapter supports both inbound and outbound XA transaction. You must set the `XATransaction` property in the Oracle WebLogic Server Administration Console to enable the XA transaction. To enable XA transaction, perform the following steps:

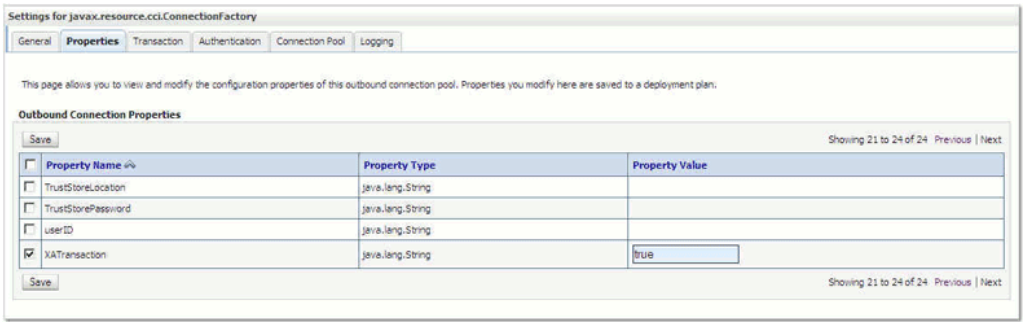
1. Log in to the Oracle WebLogic Server Administration Console using your password credentials.
2. Under Domain Structure, in the left pane, click **Deployments**. The Summary of Deployments page is displayed.
3. Click **MQSeriesAdapter**. The Settings of MQSeriesAdapter page is displayed.
4. Click the **Configuration** tab. The Configuration submenu options are displayed.
5. Click **Outbound Connection Pools**. The Outbound Connection Pool Configuration Table is displayed.
6. Click the + icon next to javax.resource.cci.ConnectionFactory and select **eis/MQ/MQAdapter**. The Outbound Connection Properties page is displayed.

   Note:

   Click **Lock & Edit** to enable the options in the console.

7. Select the **XATransaction** option and click the Property Value row at the end of the XATransaction.
8. Enter `true` in the text field, as shown in Figure 10-6, and click **Save**.

Figure 10-6 Outbound Connection Properties Page



9. Click the **Transaction** tab. The Settings for `javax.resource.cci.ConnectionFactory` page is displayed.
10. Select **XA Transaction** from the Transaction Support list.
11. Click **Save** to save your settings. The Save Deployment Plan Assistant page is displayed.
12. Click **OK**.

You have successfully enabled XA transaction for the Oracle MQ Series Adapter.

In order to use the XA transaction feature for MQ Series with BPEL for synchronous inbound request-reply scenario, you must set the `bpel.config.transaction` parameter to **required**. If this parameter is not set, then it causes the transaction to split at the BPEL boundary and MQ returns `MQRC_SYNCPOINT_NOT_AVAILABLE` error code.

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

XA Recovery

In a scenario involving fail over, such as when the `prepare phase` completes successfully before a middleware fails, messages must be recovered within the adapter without restarting the MQSeries server. You must manually resolve the in-doubt transactions.

To view all in-doubt transactions for a Queue Manager, you must execute the following command at the command prompt:

```
dspmqtrn -m[ourQueueManager]
```

To backout the messages, use the following command:

```
rsvmqtrn -m[ourQueueManager] -b [Transaction],[Number]
```

To commit the messages, use the following command :

```
rsvmqtrn -m[ourQueueManager] -c [Transaction],[Number]
```

Note:

You can use the`[Transaction]` and `[Number]` from the output of the `dspmqtrn` command.

XA Support Available for JMS Adapter to Communcate with ActiveMQ Series 5.8

Support for XA communication between the JMS Adapter with ActiveMQ Series 5.8 is available. To provide for this communication:

1. In the WebLogic Console, create a JNDI entry by specifying

   ```
   eis/activemq/XAQueue
   ```

   and specifying the ConnectionFactoryLocation as

   ```
   org.apache.activemq.ActiveMQXAConnectionFactory
   ```

2. Provide `FactoryProperties`

   ```
   BrokerURL=tcp://<YOUR_HOST>:<YOUR_PORT>;ThirdPartyJMSProvider=true
   ```

3. Set `IsTransacted : false`

High Availability

The Oracle MQ Series Adapter supports the high availability feature for the active-active topology with Oracle BPEL Process Manager (Oracle BPEL PM) and Mediator service engines. It supports this feature for both inbound and outbound operations.

Prerequisites for High Availability

Before you configure the Oracle MQ Series Adapter for high availability, you must ensure that the following prerequisites are met:

- Clustered processes must use the same queue.
- Retries should be configured either at the binding level or by using fault policies to ensure that any intermittent errors can be resolved automatically.

High Availability in Inbound/Outbound Operations

The Oracle MQ Series Adapter must ensure that it participates in the XA transaction. For more information about the XA transaction, see XA Transactions.

Scalability

The Oracle MQ Series Adapter supports the scalability feature for inbound operations only. Oracle MQ Series Adapter provides the parameter to control the number of threads that dequeue the messages from the inbound queue.You must specify the following property in the `.jca` file:

```
InboundThreadCount='N'
```

where, N is the number of threads to span to dequeue the messages from the inbound queue. The default setting is 2.

The Oracle MQ adapter creates the back-endconnections at deployment time, that is, at that time the adapter endpoint starts polling. You can have the application server prewarm the connection pool, which would provide a small marginal advantage, although connection creation does not otherwise delay the overall deployment task itself.

The example syntax for using InboundThreadCount in the .jca file is:

```
<adapter-config name="ExpressDeathEventListener" adapter="MQSeriesAdapter"
wsdlLocation="ExpressDeathEventListener.wsdl" <xmlns="http://
platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/MQ/MQAdapter" adapterRef=""/>
   <endpoint-activation portType="Dequeue_ptt" operation="Dequeue"
UITransmissionPrimitive="Dequeue">
   <activation-spec className="oracle.tip.adapter.mq.inbound.ActivationSpecImpl">
      <property name="QueueName" value="BPMPOC_EXPCLAIMQ"/>
      <property name="InboundThreadCount" value="10"/>
   </activation-spec>
  </endpoint-activation>
</adapter-config>
```

Securing Enterprise Information System Credentials

The Oracle MQ Series Adapter supports securing of the Enterprise Information System (EIS) credentials such as the user name and password, whenever it establishes an outbound connection with EIS. You can secure the user name and password for Oracle MQ Series Adapter by using Oracle WebLogic Server container-managed sign-on.

For more information, see Securing Enterprise Information System Credentials.

Fault Policy

A fault policy file defines fault conditions and their corresponding fault recovery actions. Each fault condition specifies a particular fault or group of faults, which it attempts to handle, and the corresponding action for it. A set of actions is identified by an ID in the fault policy file.

The Oracle MQ Series Adapter supports defining rejection handlers by using fault policies.

For more information about fault policies, see Configuring Rejection Handlers.

Inbound Rejection Handler

The Oracle MQ Series Adapter supports inbound message rejection handling. You can configure the message rejection handler to process translation errors, take corrective action.

For more information about rejection handlers, Configuring Rejection Handlers.

Retry Mechanism

The Oracle MQ Series Adapter supports the following two mechanisms for inbound retry:

- JCA Inbound Retry Mechanism
- Message Backout Queue

The JCA inbound retry mechanism is commonly used by all adapters, in general, whereas the message backout queue mechanism is used only by the Oracle MQ Series Adapter. If you specify the `BackoutQueueName` property in the .jca file, only then the Oracle MQ Series Adapter uses the message backout queue mechanism to retry. By default, the JCA inbound retry mechanism is used for retry.

Note:

Both these methods of retry in the Oracle MQ Series Adapter are mutually exclusive operations; the adapter uses one mechanism at a time. If you specify both options, then the Backout Queue option takes precedence.

JCA Inbound Retry Mechanism

The Oracle MQ Series Adapter supports a pull model for connecting to the back-end application for receiving events. Connection-related issues are considered recoverable and most inbound adapters keep retrying until the adapters are able to establish connection with the EIS.

In case of Oracle MQ Series Adapter, a message not being able to put to a queue is also retriable.

For more information about retry mechanism, see Error Handling.

Message Backout Queue

Backout Queue is a queue for putting rejected messages from an inbound queue. The inbound adapter checks for the backout count of the messages and if this count exceeds the `MaximumBcakoutCount` value, then the adapter puts the messages to the specified Backout Queue. This mechanism is used by the Oracle MQ Series Adapter to handle inbound retries for the rejected messages.

If you specify the `BackoutQueueName` property in the `.jca` file, then Oracle MQ Series Adapter uses the message backout count for retries. You can specify the maximum retries using the `MaximumBackoutCount` property. The default value for this property is infinite. If you do not specify the `MaximumBackoutCount` value along with the `BackoutQueueName`, then the adapter retries infinitely. The adapter does not consider `JCA` retries (specified in composite.xml) when the BackOut Queue properties are specified.

The `BackoutRetries` property must be set to specify the number of retries for delivering the message to the Backout Queue with retry interval set using the `BackoutRetryInterval` property. The default value for `BackoutRetries` is `3` and `BackoutInterval` is `5` sec.

If a message gets rejected even after the `MaximumBackoutCount` value is reached, then the adapter puts the message to Backout Queue. If Oracle MQ Series Adapter is cannot put the message to Backout Queue, then the adapter tries till the `BackoutRetries` count with the `BackoutInterval` time delay. If even after the BackoutRetries the adapter cannot put the message to Backout Queue, then the adapter deactivates the endpoint.

You must also specify the name of the Queue Manager of the Backout Queue in the `BackoutQueueManagerName` property. You must not use this property if the BackoutQueue resides on the inbound queue QueueManager.

Note:

When using the Backout Queue, consider the following:

- The Backout Queue options cannot be used for translation failures.
- In cases where both JCA and BackOut retries are specified, the BackOut retries takes precedence.

For more information about configuring Backout Queues, see Configuring a Backout Queue.

Performance Tuning

The Oracle MQ Series Adapter supports performance tuning options.

Oracle MQ Series Adapter Concepts

This section explains the following concepts of the Oracle MQ Series Adapter:

- Messaging Scenarios
- Message Properties
- Correlation Schemas
- Distribution List Support
- Report Messages
- Message Delivery Failure Options
- Message Segmentation
- Integration with CICS
- Using the MQ Series Client Channel Definition Table Feature

Messaging Scenarios

The Oracle MQ Series Adapter supports the following messaging scenarios:

- Enqueue Message
- Dequeue Message
- Asynchronous Request-Response (As Client)
- Synchronous Request-Response (As Server)
- Asynchronous Request-Response (As Server)
- Synchronous Request-Response (As Server)
- Synchronous Request-Response (As Client)
- Synchronous Request-Response (Oracle Mediator as Client)
- Asynchronous Request-Response (Oracle Mediator As Client)

- Outbound Dequeue Scenario

Enqueue Message

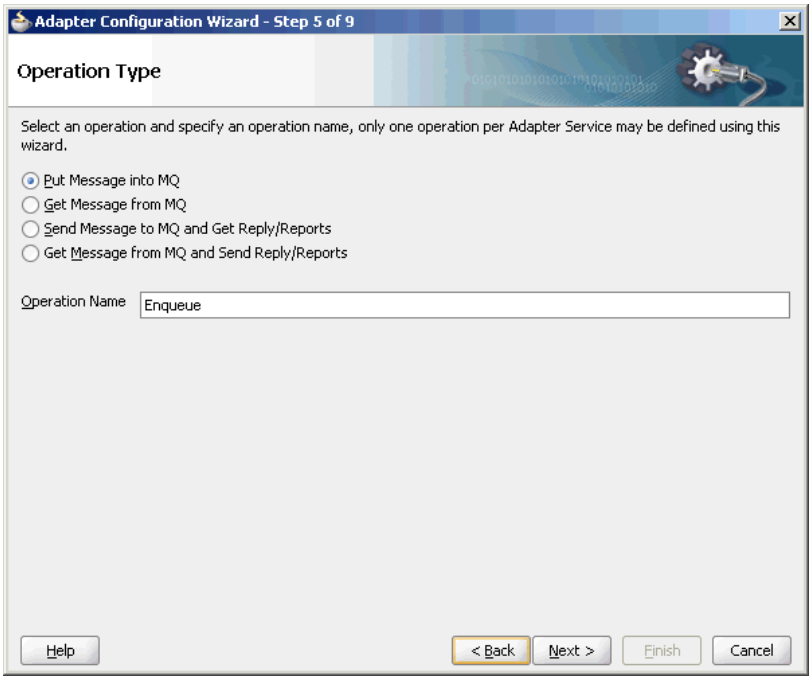
In this scenario, the Oracle MQ Series Adapter connects to a specific queue managed by a queue manager and then writes the message to the queue. For outbound messages sent from Oracle BPEL PM or Mediator, the Oracle MQ Series Adapter performs the following operations:

1. Receives message from Oracle BPEL PM or Mediator.
2. Formats the XML content as specified at design time.
3. Sets the properties of the message, such as priority, expiry, message type, and persistence. These properties are based on the selections that you made in the Adapter Configuration Wizard.

   For more information about message properties, see Messages Types.
4. Sends the message to the queue specified at design time in the Adapter Configuration Wizard.

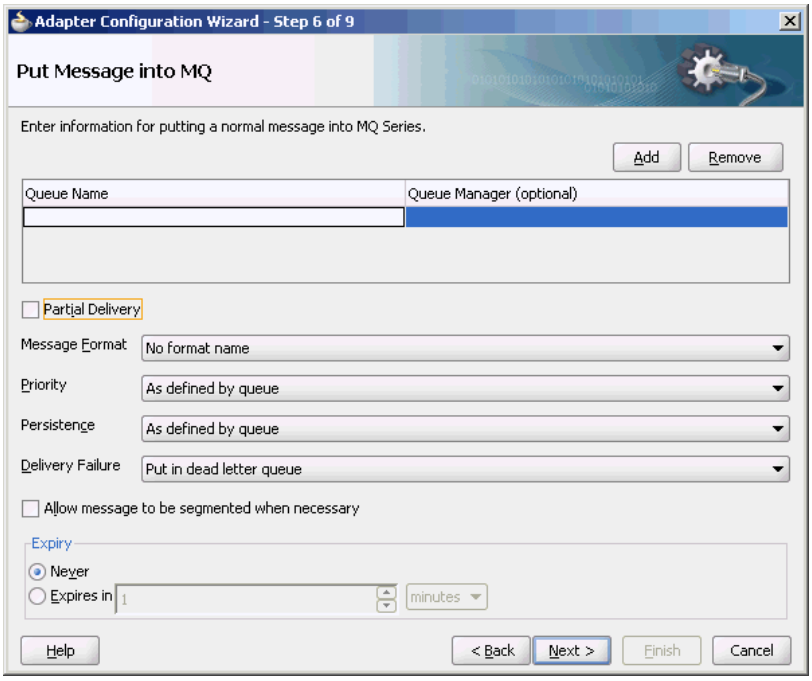
Figure 10-7 displays the operation type that you must select in the Adapter Configuration Wizard.

Figure 10-7 The Adapter Configuration Wizard: Produce Message Selection



The page that appears after selecting the Put Message into MQ operation type is shown in Figure 10-8.

Figure 10-8 Put Message Options



You can specify the following properties in this page:

- **Queue Name**: The name of the queue on which the Oracle MQ Series Adapter enqueues the message. This is a mandatory field.
- **Queue Manager**: The name of the queue manager to which the queue belongs. This field is optional and is necessary when enqueuing message to a remote queue.
- **Partial Delivery**: This is applicable only when you specify multiple queues for outbound operation, which is also known as the Distribution List scenario. Partial Delivery takes either `true` or `false`. If assigned `true`, then even if the delivery of message fails for some queues, it would still go and put the message to the rest of the queues specified in the distribution list. If assigned `false`, it means even if one message fails, then the message is not put to any queue.
- **Message Format**: The format of the message.

Note:

When enqueuing a message, ensure that the various mandatory values, required for a specific format, are specified correctly.

- **Priority**: The priority of the message, ranging from 0 (low) to 9 (high).
- **Persistence**: The persistence of the message. You can also specify the persistence of the message to be taken from the default persistence attribute, as defined by the destination queue.
- **Delivery Failure**: If the delivery of message fails, then either it can be put to a dead letter queue or it can be discarded.
- **Allow Messages to Be Segmented When Necessary**: This is applicable to a message that is big enough for the queue to accommodate. In that case, if you have

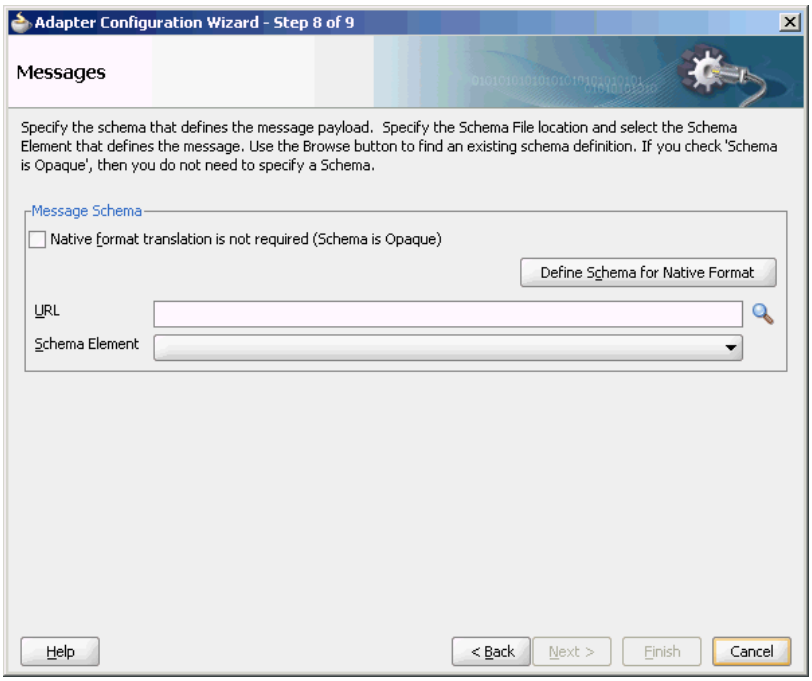
specified that it has to be segmented, then the single message can be broken into that many bytes the queue can take, which results in multiple messages.

- **Expiry**: The expiry time of the message. The message is discarded after the expiry time has elapsed.

For more information about these properties, see Message Properties.

The next Adapter Configuration Wizard page that appears is the Messages page, as shown in Figure 10-9. This page enables you to select the XML Schema Definition (XSD) file for translation.

Figure 10-9 Messages Page



If native format translation is not required (for example, a JPG or GIF image is being processed), then select the **Native format translation is not required** check box. The file is passed through in base-64 encoding.

XSD files are required for translation. To define a new schema or convert an existing data type description (DTD) or COBOL Copybook, select **Define Schema for Native Format**. This starts the Native Format Builder wizard. This wizard guides you through the creation of a native schema file from file formats, such as delimited by special characters, comma-delimited value (CSV), fixed-length, DTD, and COBOL Copybook. After the native schema file is created, you are returned to this Messages page with the **Schema File URL** and **Schema Element** fields filled in.

For more information, see Creating Native Schema Files with the Native Format Builder Wizard.

Note:

Ensure that the schema you specify includes a namespace. If your schema does not have a namespace, an error message appears.

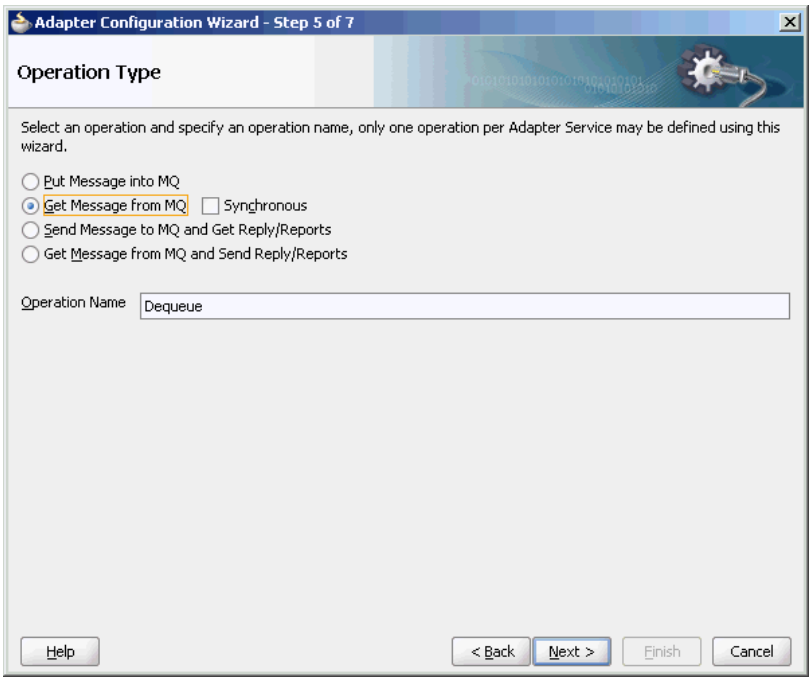
Dequeue Message

In this scenario, the Oracle MQ Series Adapter connects to a specific queue managed by a queue manager and then removes the message from the queue. For inbound messages sent to Oracle BPEL PM or Mediator, the Oracle MQ Series Adapter performs the following operations:

1. Connects to the queue specified at design time.
2. Dequeues the message from the queue when a message arrives.
3. Reads and translates the message based on the translation logic defined at design time.
4. Publishes the message as an XML message to Oracle BPEL PM or Mediator.

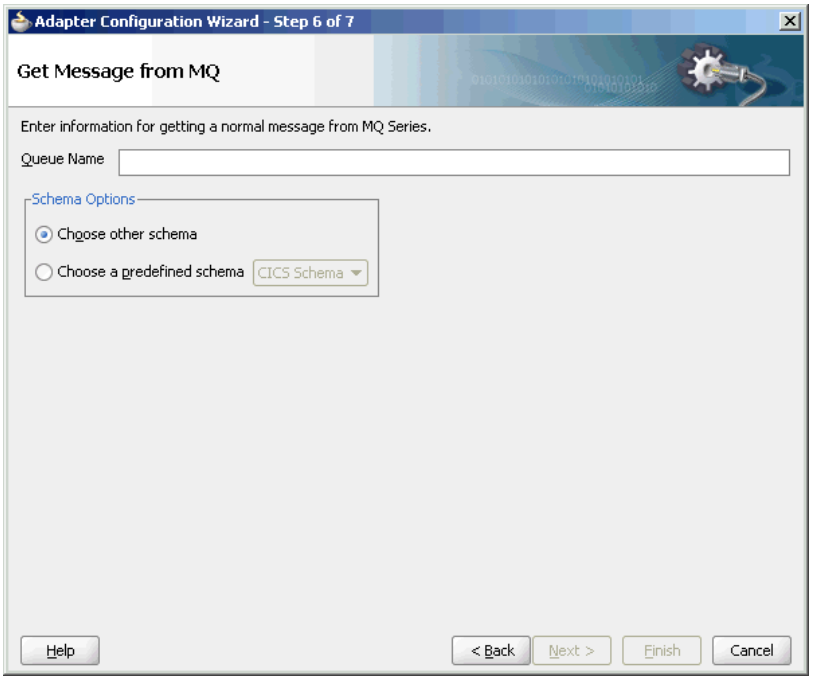
   Figure 10-10 displays the operation type that you must select in the Adapter Configuration Wizard.

Figure 10-10 The Adapter Configuration Wizard: Consume Message Selection



The page that appears after selecting the **Get Message from MQ** operation type is shown in Figure 10-11.

Figure 10-11 Get Message from MQ Page



You can specify the following properties in this page:

- **Queue Name**: The name of the queue from which the Oracle MQ Series Adapter dequeues the message. This is a mandatory field.
- **Schema Options**: This option enables you to specify the schema for the message to be dequeued.
  - **Choose Other Schema**: This option enables you to choose your schema for the message to be dequeued.
  - **Choose a Predefined Schema**: This option enables you to choose a readymade schema that the adapter provides.

The next Adapter Configuration Wizard that appears is the Messages page, as shown in Figure 10-9. This page enables you to select the XSD schema file for translation.

As with specifying the schema for the produce message operation, you can perform the following tasks in this page:

- Specify if native format translation is not required.
- Select the XSD schema file for translation.
- Start the Native Format Builder wizard to create an XSD file from file formats such as CSV, fixed-length, DTD, and COBOL Copybook.

For more information about the Messages page, see Enqueue Message.

Asynchronous Request-Response (Oracle BPEL PM As Client)

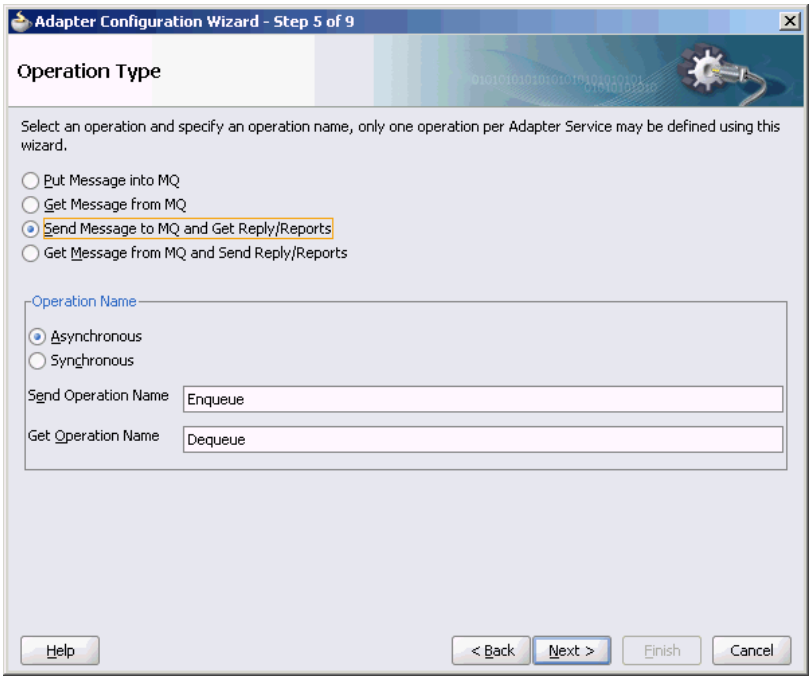
In this scenario, the Oracle BPEL PM sends a request message and receives the corresponding response using a non-initiating receive activity. The invoke activity

initiates the outbound invocation of the adapter to send the request. The Oracle MQ Series Adapter performs the following operations:

1. Receives message from Oracle BPEL PM.
2. Formats the XML content as specified at design time in the Adapter Configuration Wizard.
3. Sets properties and a correlation schema on the request message.
4. Sends the message to the queue specified at design time. The third-party application receives the message, processes it, generates the response, and then enqueues the response message to the `replyTo` queue specified in the request message. The Correlation ID and Message ID of the response message are generated based on the correlation schema specified in the request message.
5. The Oracle MQ Series Adapter dequeues the message from the `replyTo` queue.
6. Sends the response to the non-initiating receive activity of Mediator. To ensure that response is sent to the correct BPEL instance, correlation schemas are used.

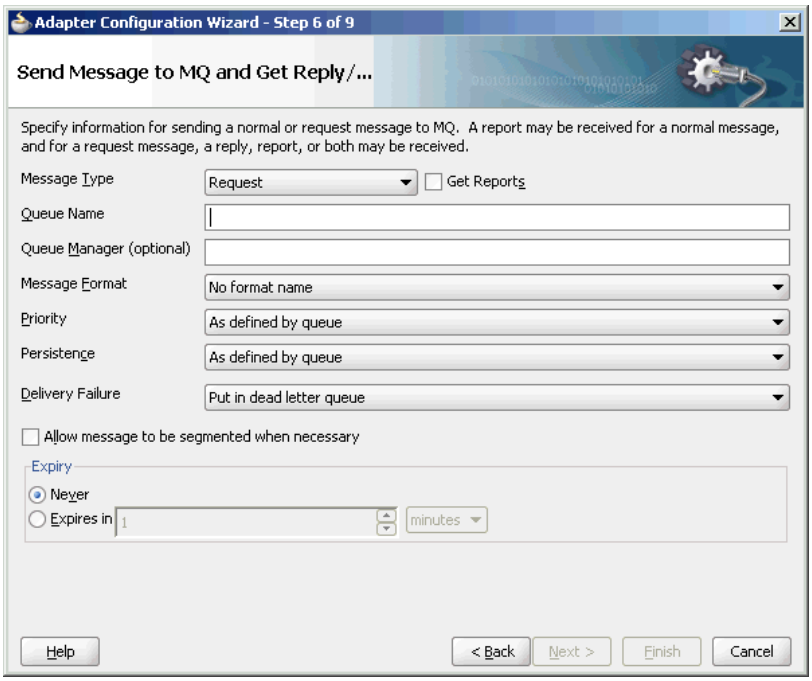
Figure 10-12 displays the operation type that you must select in the Adapter Configuration Wizard.

Figure 10-12 Selecting an Operation Type



The page that appears after selecting the **Send Message to MQ and Get Reply/ Reports** operation type is shown in Figure 10-13.

Figure 10-13 Send Message to MQ and Get Reply/Reports Page



You can specify the following properties in this page:

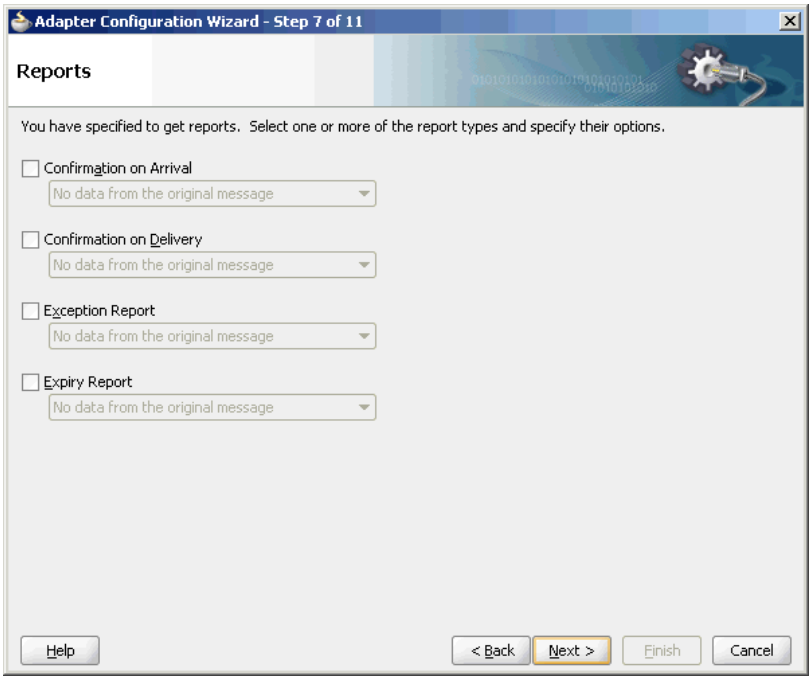
- **Message Type**: The type of the message. You can either send a normal message or a request message.
- **Get Reports**: Select this option if you want any kind of report. You can specify the type of report in the next page, as shown in Figure 10-14.
- **Queue Name**: The name of the queue to which the Oracle MQ Series Adapter enqueues the message. This is a mandatory field.
- **Queue Manager**: The name of the queue manager to which the queue belongs. This field is optional.
- **Message Format**: The format of the message.
- **Priority**: The priority of the message ranging from 0 (low) to 9 (high).
- **Persistence**: The persistence of the message. You can also specify the persistence of the message to be taken from the default persistence attribute, as defined by the destination queue.
- **Delivery Failure**: If the delivery of the message fails, then either it can be put to a dead letter queue or it can be discarded.
- **Allow Messages to Be Segmented When Necessary**: This is applicable to a message that is big enough for the queue to accommodate. In that case, if you have specified that it has to be segmented, then the single message can be broken into that many bytes the queue can take, which results in multiple messages.
- **Expiry**: The expiry time of the message. The message is discarded after the expiry time has elapsed.

For more information about these properties, see Message Properties and Report Messages.

The page that is displayed when you click **Next** in the Send Message to MQ and Get Reply/Reports page can be a Reports page (shown in Figure 10-14) or a Response page (shown in Figure 10-15).

The Reports page, shown in Figure 10-14, is displayed only if you have selected the Get Reports option in the Send Message to MQ and Get Reply/Reports page, as shown in Figure 10-13.

Figure 10-14 Reports Page



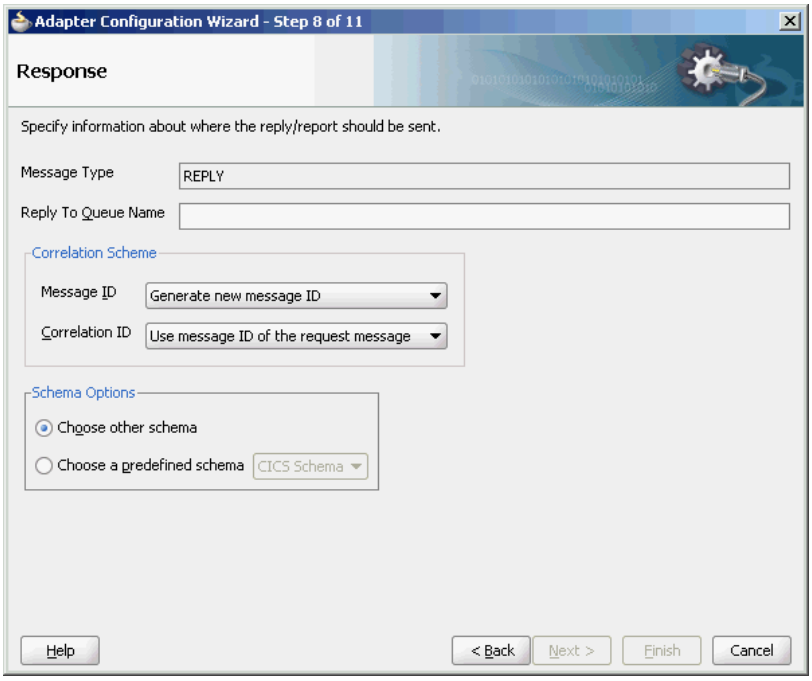
You can select the following types of reports in this page:

- Confirmation on Arrival
- Confirmation on Delivery
- Exception Report
- Expiry Report

For information about these report types, see Report Messages.

The Response page shown in Figure 10-15 is displayed when you click **Next** in the Reports page.

Figure 10-15 Response Page



You can specify the following properties in the Response page:

- **Reply to Queue Name**: The name of the reply queue name.
- **Correlation Scheme**: The correlation schema that is necessary for the Oracle MQ Series Adapter.

  For information about correlation schemas, see Correlation Schemas.
- **Schema Options**: This option enables you to specify the schema for the message to be dequeued.
  - **Choose Other Schema**: This option enables you to choose your schema for the message to be dequeued.
  - **Choose a Predefined Schema**: This option enables you to choose a readymade schema that the adapter provides.

Note:

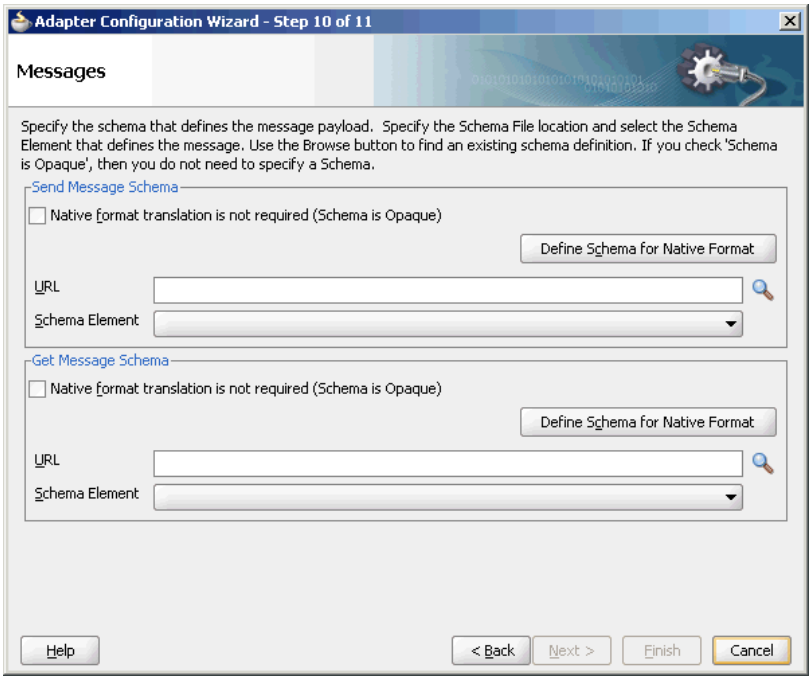
For Oracle MQ Series Adapter in an asynchronous outbound request/reply scenario (in this scenario a request message is put to a queue and the reply for the same is dequeued from the replyToQueue asynchronously), properties are differentiated by an `(Enqueue)` or `(Dequeue)` label in Oracle Enterprise Manager Console. . This differentiation is necessary to distinguish the property names for enqueue and dequeue. For example, `QueueName(Enqueue)` is used for putting a message and `QueueName(Dequeue)` is used for dequeuing the reply.

When using Oracle Enterprise Manager Console to edit Oracle MQ Series Adapter properties in this scenario, note the following:

- Because the replyToQueue for the request message is the same from the which the reply must be dequeued, if you change the `ReplyToQueueName(Enqueue)` property, you must also change the `QueueName(Dequeue)` property to the same value.
- Similarly, if you change the `MessageId(Dequeue)` property, you must also change the `MessageId(Enqueue)` property to the same value.
- Similarly, if you change the `CorrelationId(Dequeue)` property, you must also change the `CorrelationId(Enqueue)` property to the same value.

When you click **Next** in the Response page, a Messages page, shown in Figure 10-16, is displayed. This page enables you to select the XSD schema file for translation for request and as response message.

Figure 10-16 Messages Page



You can perform the following tasks in this page:

- Specify if native format translation is not required.
- Select the XSD schema file for translation.
- Start the Native Format Builder wizard to create an XSD file from file formats such as CSV, fixed-length, DTD, and COBOL Copybook.

For more information about the Messages page, see Enqueue Message.

In the solicit-request-response scenario, the reply message is expected in the reply queue specified with some correlation scheme that is provided through the request message. This reply queue, which is used by a particular process (BPEL/Mediator), should not be used by any other process.

If the same reply queue is used by some other application, then the message might be picked, irrespective of whether the reply message had the proper correlation or not, and eventually the message becomes lost.

Synchronous Request-Response (Oracle BPEL PM As Server)

In this scenario, the Oracle BPEL PM receives a request, processes it, and sends the response synchronously by using a reply activity. The Oracle MQ Series Adapter performs the following operations:

1. Dequeues the request message from the queue when the message arrives.
2. Reads and translates the message based on the translation logic defined at design time.
3. Publishes the message as an XML message to Oracle BPEL PM. The Oracle BPEL PM processes the request and sends the response to the Oracle MQ Series Adapter.
4. Receives the response message from the Oracle BPEL PM.
5. Formats the XML content as specified at design time.
6. Sets the properties of the message such as priority, expiry, message type, and persistence. These properties are based on the selections that you made in the Adapter Configuration Wizard.
7. Sends the message to the queue specified at design time in the Adapter Configuration Wizard.

Figure 10-17 shows a sample BPEL process for this scenario.

Figure 10-17 Synchronous Request-Response Oracle BPEL PM As Server Sample

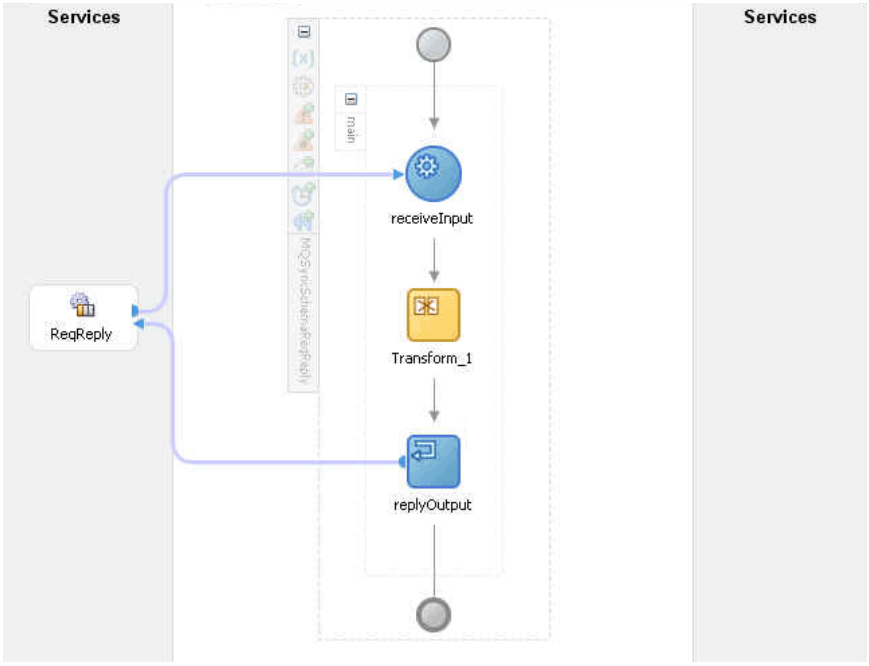
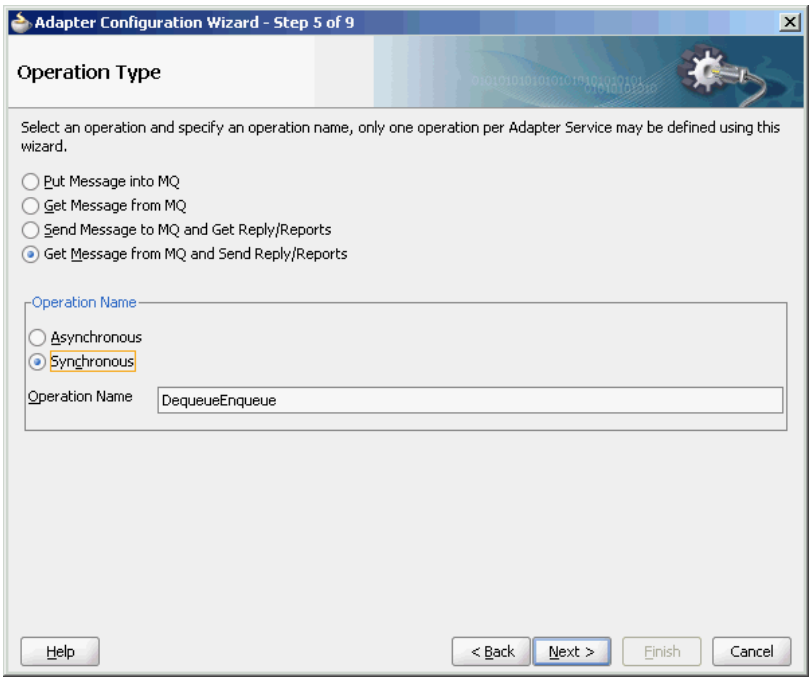


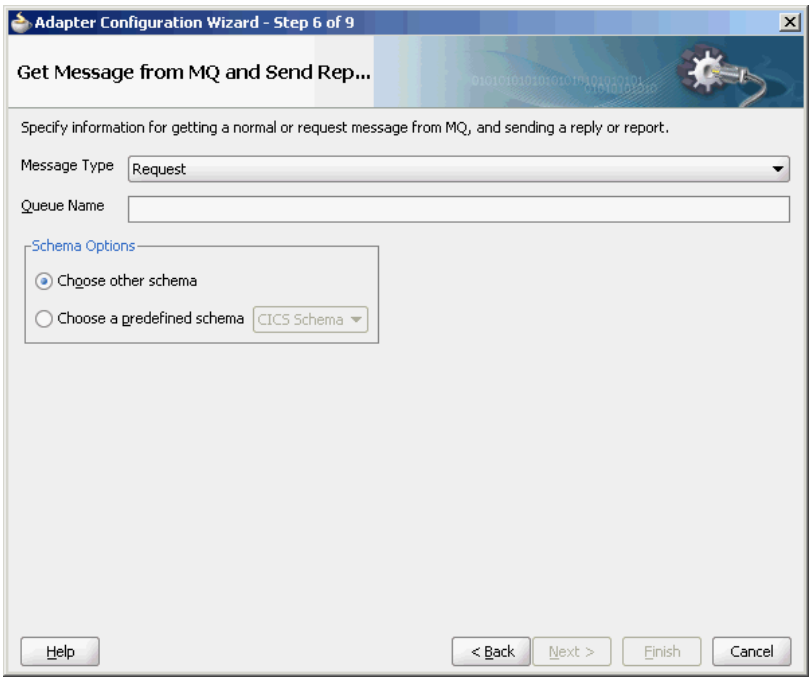
Figure 10-18 displays the operation type that you must select in the Adapter Configuration Wizard.

Figure 10-18 Operation Type Page Selection for Request-Response Synchronous Interaction



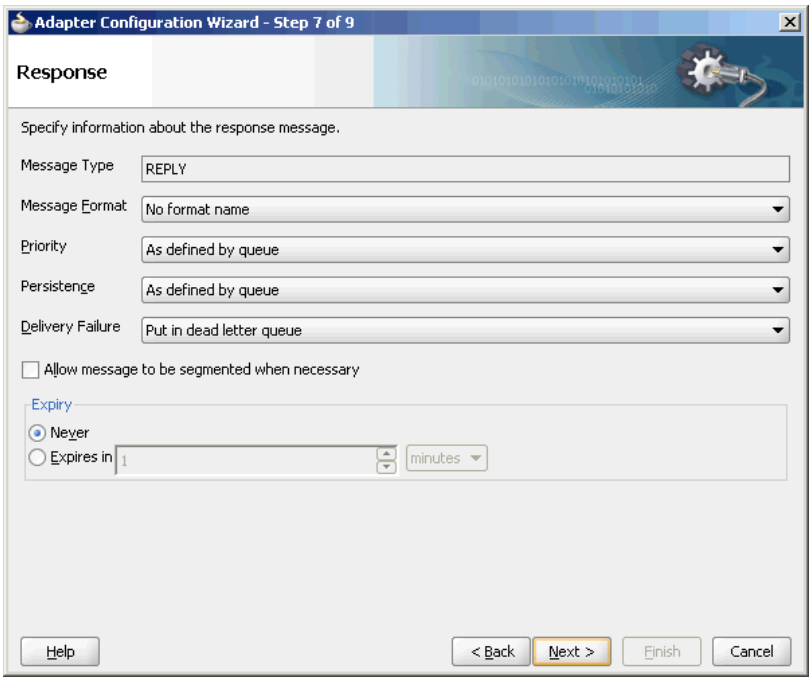
The page that appears after you select the **Get Message from MQ and Send Reply/ Reports** operation type is shown in Figure 10-19. Specify the queue name from which the Oracle MQ Series Adapter dequeues the message in this page.

Figure 10-19 Get Message from MQ and Send Reply/Reports Page



When you click **Next** in the Get Message from MQ and send Reply/Reports page, the Response page shown in Figure 10-20 is displayed.

Figure 10-20 Response Page for Synchronous Request-Response



You can specify the following properties in the Response page:

- **Message Type**: The message type of the message to be dequeued. This option affects the return message type.
- **Message Format**: The format of the message.

- **Priority**: The priority of the message.
- **Persistence**: The persistence of the message. You can also specify the persistence of the message to be taken from the default persistence attribute, as defined by the destination queue.
- **Delivery Failure**: If the delivery of the message fails, then either it can be put to a dead letter queue or it can be discarded.
- **Allow Messages to Be Segmented When Necessary**: This is applicable to a message that is big enough for the queue to accommodate. In that case, if you have specified that it has to be segmented, then the single message can be broken into that many bytes the queue can take, which results in multiple messages.
- **Expiry**: The expiry time of the message.

For more information about these properties, see Message Properties.

Click **Next** in the Response page, the Messages page is displayed, as shown in Figure 10-16. You can perform the following tasks in this page:

- Specify if native format translation is not required.
- Select the XSD schema file for translation.
- Start the Native Format Builder wizard to create an XSD file from file formats such as CSV, fixed-length, DTD, and COBOL Copybook.

For more information about the Messages page, see Enqueue Message.

Asynchronous Request-Response (Oracle BPEL PM As Server)

In Oracle BPEL PM initiated request-response interaction, a BPEL process receives a request as an inbound message, processes it, and then sends the response through an invoke activity. For asynchronous request-reply scenario, the Oracle MQ Series Adapter performs the following operations:

1. Dequeues the message from the queue when a message arrives.
2. Reads and translates the message based on the translation logic defined at design time.
3. Publishes the message as an XML message to Oracle BPEL PM. The Oracle BPEL PM processes the request and sends the response to the Oracle MQ Series Adapter.
4. Receives messages from Oracle BPEL PM.
5. Formats the XML content as specified at design time.
6. Sets the properties of the message, such as priority, expiry, message type, and persistence. These properties are based on the selections that you made in the Adapter Configuration Wizard.
7. Sends the message to the queue specified at design time in the Adapter Configuration Wizard.

Figure 10-21 shows a sample BPEL process for this scenario.

Figure 10-21 Asynchronous Request-Response Oracle BPEL PM As Server Sample

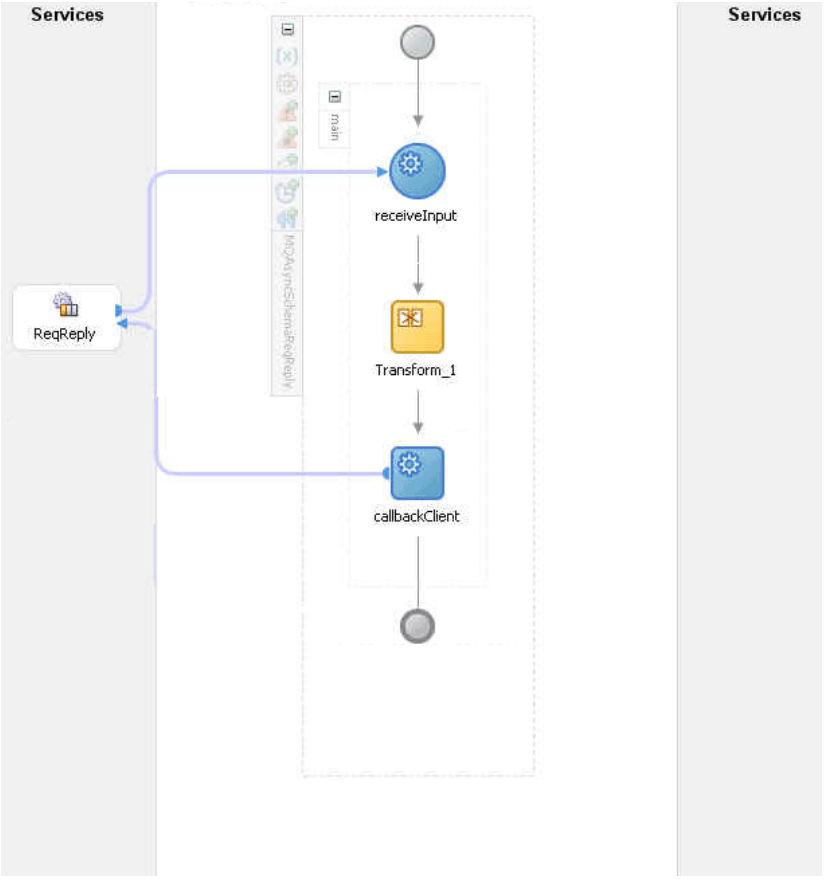
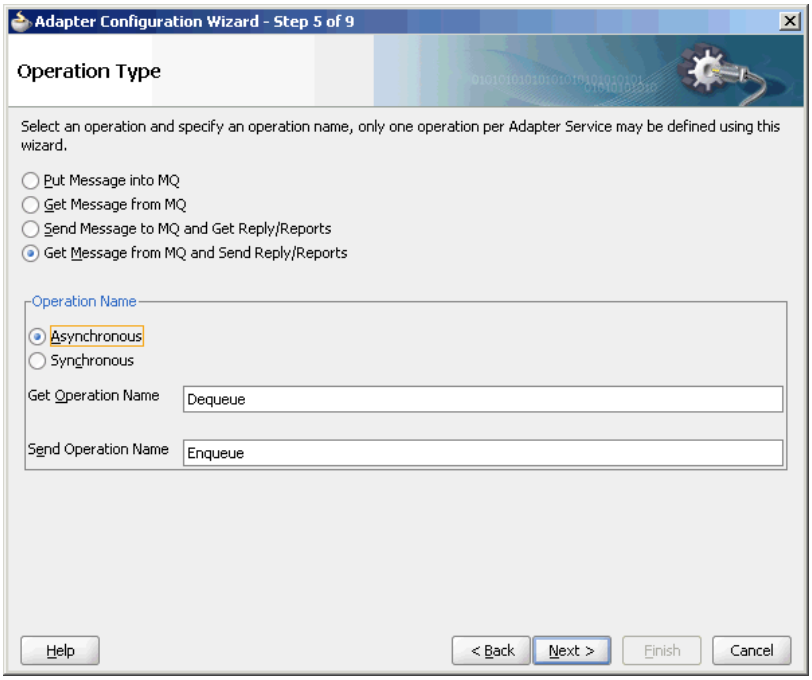


Figure 10-22 displays the operation type that you must select in the Adapter Configuration Wizard.

Figure 10-22 Operation Type Page Selection for Request-Response Asynchronous Interaction



The page that appears after selecting the Get Message from MQ and send Reply/Reports operation type is shown in Figure 10-19. Specify the queue name from which the Oracle MQ Series Adapter dequeues the message in this page.

When you click Next in the Get Message from MQ and send Reply/Reports page, the Response page shown in Figure 10-20 is displayed.

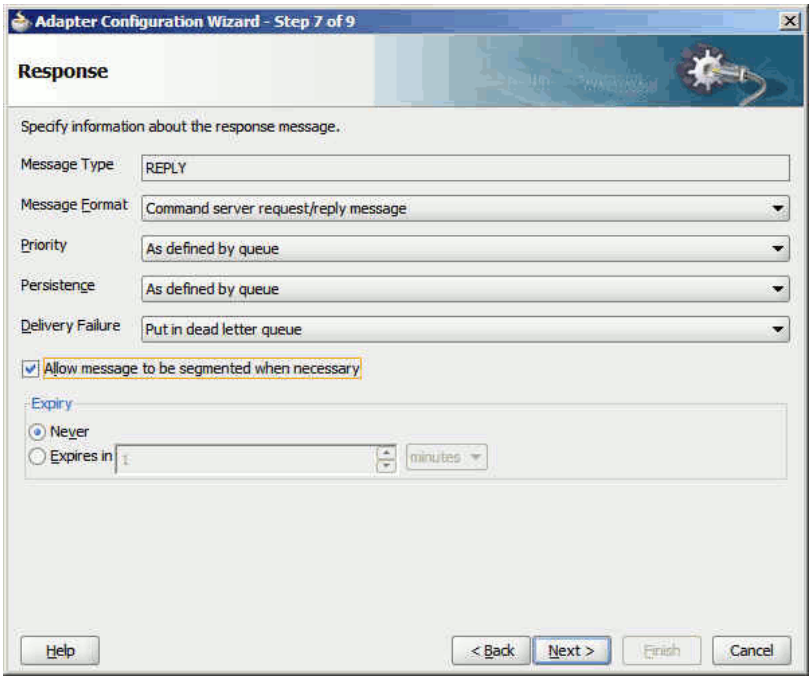
You can specify the following properties in the Response page:

- **Message Type**: The message type of the message to be dequeued. This option affects the return message type.
- **Message Format**: The format of the message.
- **Priority**: The priority of the message.
- **Persistence**: The persistence of the message. You can also specify the persistence of the message to be taken from the default persistence attribute, as defined by the destination queue.
- **Delivery Failure**: If the delivery of the message fails, then either it can be put to a dead letter queue or it can be discarded.
- **Allow Messages to Be Segmented When Necessary**: This is applicable to a message that is big enough for the queue to accommodate. In that case, if you have specified that it has to be segmented, then the single message can be broken into that many bytes the queue can take, which results in mulitple messages.
- **Expiry**: The expiry time of the message.

For more information about these properties, see Message Properties.

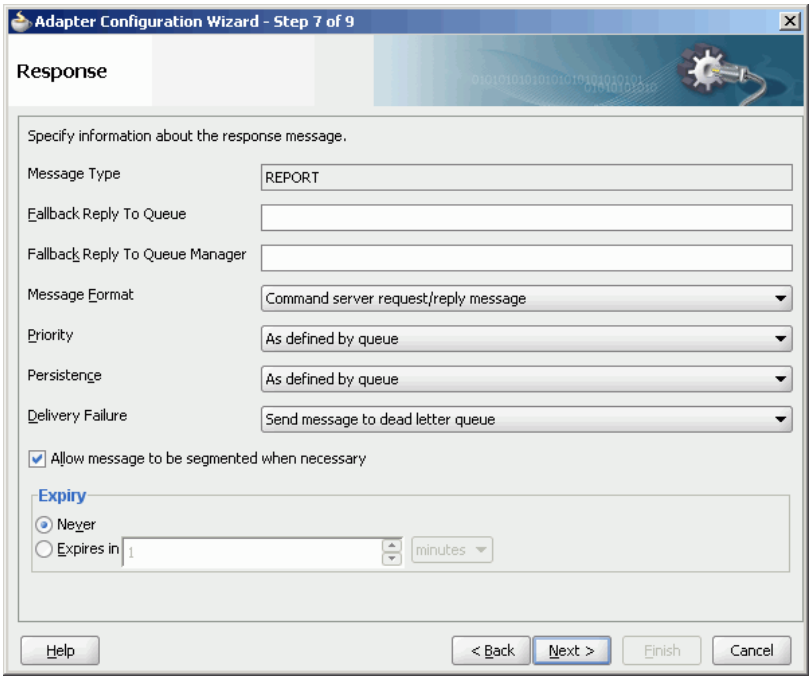
The page that is displayed when you click Next in the Get Message to MQ and Send Reply/Reports page is a Response page (shown in Figure 10-23 and Figure 10-24) but with two different set of options.

Figure 10-23 Response Page (Request Message Type Selected)



The Response page shown in Figure 10-24 is displayed only if you have selected the **Normal** option in Message Type field in the Get Message to MQ and Send Reply/ Reports page.

Figure 10-24 Response Page (Normal Message Type Selected)



You can specify the following properties in the Response page:

- (Optional) **Fallback Reply to Queue**: Enter a response fallback queue name. The response message is enqueued to the queue specified with the replyToQueue property of the request message. However, if the replyToQueue property is not set

on the request message, then entering a name here ensures that the process does not fail to enqueue the response.

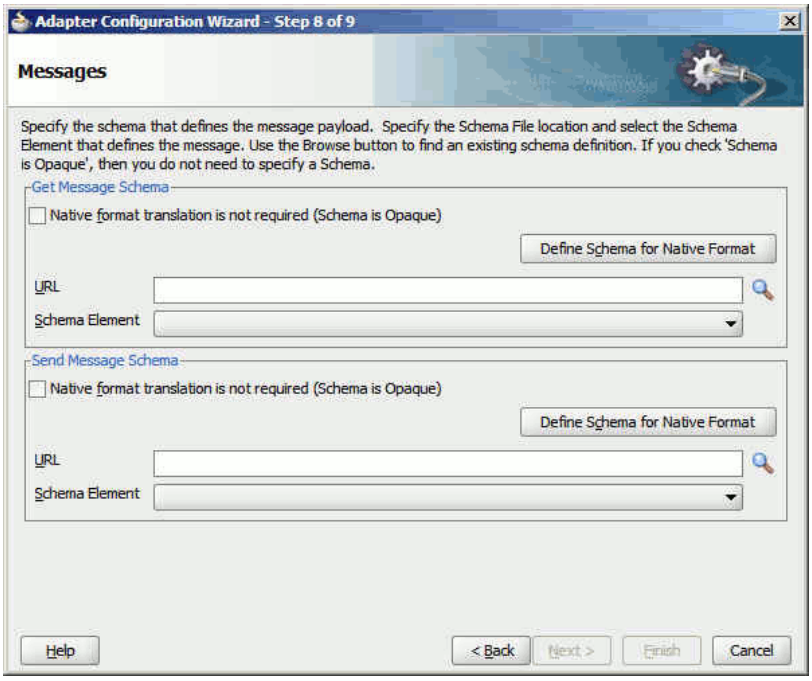
- (Optional) **Fallback Reply to Queue Manager**: Enter a secondary queue name. This name is used when the primary queue manager that was established when you specified the JNDI connection name cannot access the queue name entered in the Queue Name field. This is similar to the functionality described in the Fallback Reply to Queue field.

  To specify the other properties in this Response page, see properties mentioned for Figure 10-23.

When you click **Next** in the Response page, the Messages page shown in Figure 10-25 is displayed. You can perform the following tasks in this page:

- Specify if native format translation is not required.
- Select the XSD schema file for translation.
- Start the Native Format Builder wizard to create an XSD file from file formats such as CSV, fixed-length, DTD, and COBOL Copybook.

Figure 10-25 Messages Page



For more information about the Messages page, see Enqueue Message.

In asynchronous request-reply interaction, you must map the following properties from the inbound message header to the outbound message header:

- `MsgID`: Refers to the message ID.
- `CorrelID`: Refers to the correlation ID of a message.
- `CorrelationScheme`: Refers to a combination of both the msgid and the correlid of the request message.

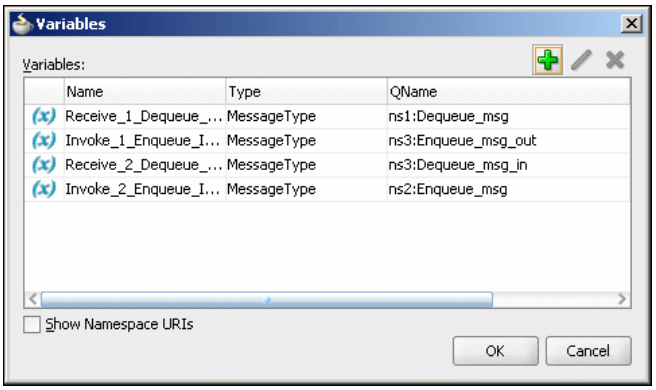
  For more information, see Correlation Schemas.

- `ReplyToQ` : Refers to the name of the response queue name.
- `ReplyToQueueManager`: Refers to the name of the response queue manager.

You can use the Assign activity to map these properties.

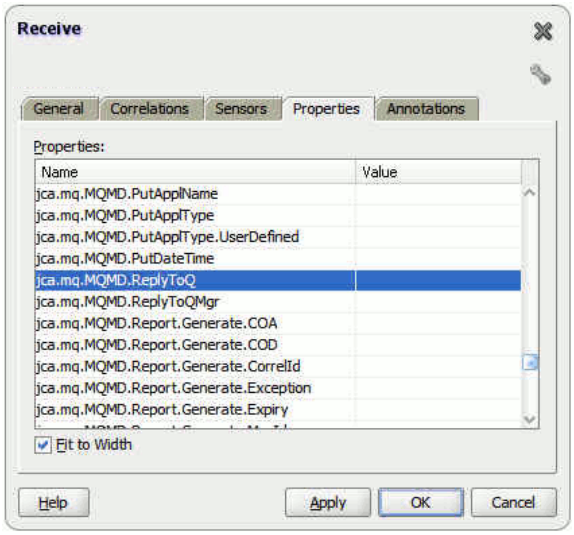
1. Create a BPEL process and double-click to open the BPEL Designer page.
2. In the vertical menu that appears, click the Variables icon that appears as (x) grayed out. The Variables dialog is displayed, as shown in Figure 10-26.

Figure 10-26 The Variables Dialog



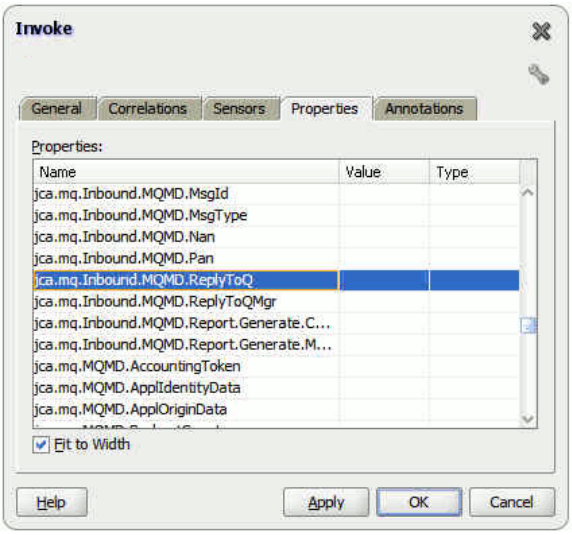
3. Capture the inbound header messages into these variables, as shown in Figure 10-27 and Figure 10-28.

Figure 10-27 The Receive Dialog



4. Assign the variables captured in Step 2 for the Outbound Reply message, as shown in Figure 10-28 and Figure 10-23.

Figure 10-28 The Invoke Dialog



Synchronous Request-Response (Mediator As Server)

In this scenario, the Mediator receives a request, processes it, and sends the response synchronously. The Oracle MQ Series Adapter performs the following operations:

1. Dequeues the request message from the queue when the message arrives.
2. Reads and translates the message based on the translation logic defined at design time.
3. Publishes the message as an XML message to Mediator. The Mediator processes the request and sends the response to the Oracle MQ Series Adapter.
4. Receives the response message from the Mediator.
5. Formats the XML content as specified at design time.
6. Sets the properties of the message such as priority, expiry, message type, and persistence. These properties are based on the selections that you made in the Adapter Configuration Wizard.
7. Sends the message to the queue specified at design time in the Adapter Configuration Wizard.

Figure 10-19 displays the operation type that you must select in the Adapter Configuration Wizard.

From this page onwards, all the pages are similar to the pages explained in Synchronous Request-Response (As Server).

Note:

The asynchronous request-response pattern is not supported for Mediator.

Synchronous Request-Response (Oracle BPEL PM As Client)

The Oracle MQ Series Adapter supports the outbound synchronous-solicit-request-response scenario. In this scenario, the adapter enqueues a normal/request message in

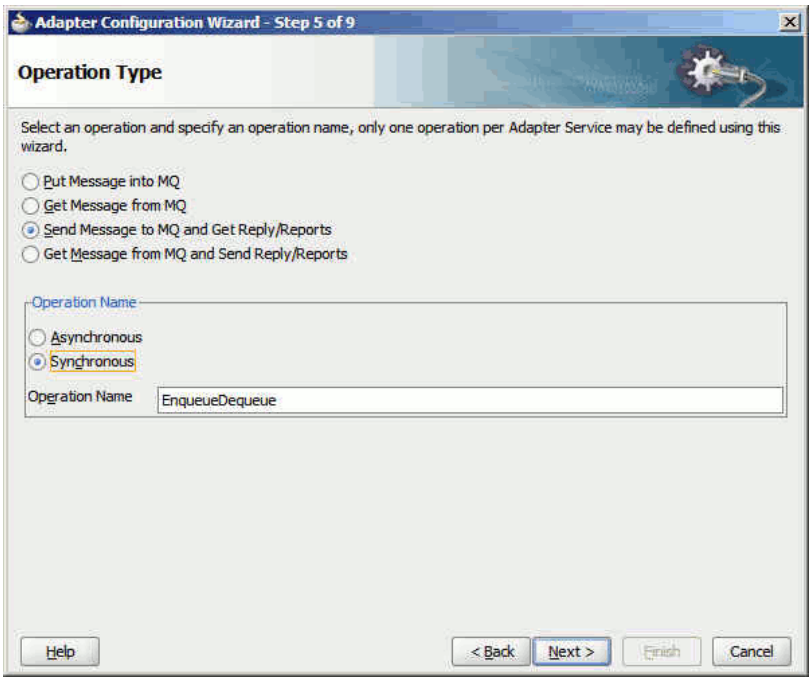
a queue and expects the report/reply synchronously. The report/reply message arrives in the `ReplyToQueueName` queue of the normal/request message.

Note:

Outbound synchronous-solicit-responses must be executed in non-XA modes as the request message does not get enqueued when it is participating in a global transaction.

Figure 10-29 displays the operation type that you must select in the Adapter Configuration Wizard.

Figure 10-29 The Operation Type Dialog



The page that appears after selecting the Send Message to MQ and Get Reply/Reports operation type is shown in Figure 10-13.

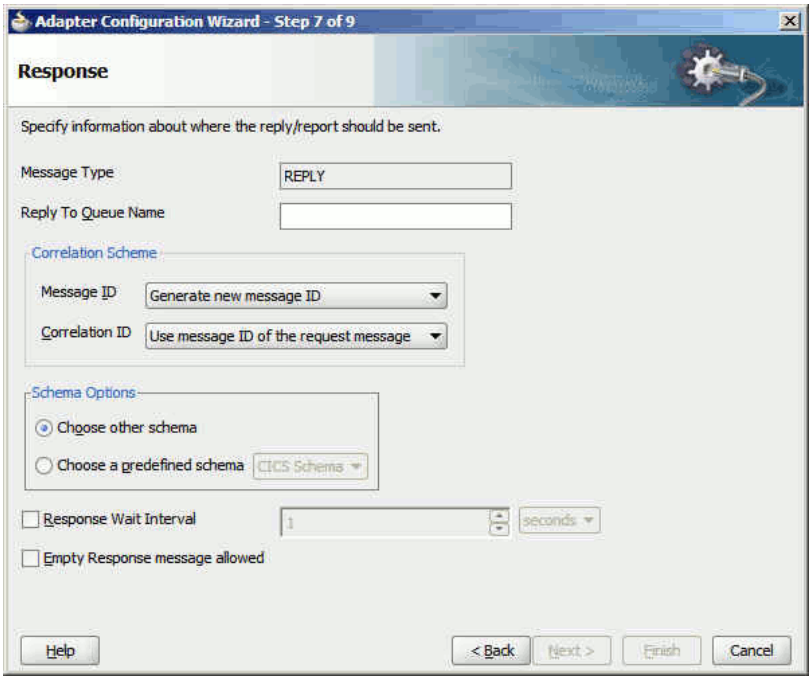
You can specify the following properties in this page:

- **Message Type**: The type of the message. You can either send a normal message or a request message.
- **Queue Name**: The name of the queue to which the Oracle MQ Series Adapter enqueues the message. This is a mandatory field.
- **Queue Manager**: The name of the queue manager to which the queue belongs. This field is optional and is necessary when enqueuing message to a remote queue.
- **Message Format**: The format of the message.
- **Priority**: The priority of the message ranging from 0 (low) to 9 (high).
- **Persistence**: The persistence of the message. You can also specify the persistence of the message to be taken from the default persistence attribute, as defined by the destination queue.

- **Delivery Failure**: If delivery of the message fails, then either it can be put to a dead letter queue or it can be discarded.
- **Allow Messages to Be Segmented When Necessary**: This is applicable to a message that is big enough for the queue to accommodate. In that case, if you have specified that it has to be segmented, then the single message can be broken into that many bytes the queue can take, which results in multiple messages.
- **Expiry**: The expiry time of the message. The message is discarded after the expiry time has elapsed.

Click Next in the Send Message to MQ and Get Reply/Reports page, the Response page, as shown in Figure 10-30, is displayed.

Figure 10-30 The Response Page



For the Synchronous Request-Response scenario, you must also edit the following properties in the Response page:

- **Reply to Queue Name**: The name of reply queue name.
- **Correlation Scheme**: The correlation schema that must be used by the Oracle MQ Series Adapter.

  For more information about correlation schemas, see Correlation Schemas.
- **Schema Options**: This option enables you to specify the schema for the message to be dequeued.
  - **Choose Other Schema**: This option enables you to choose your schema for the message to be dequeued.
  - **Choose a Predefined Schema**: This option enables you to choose a readymade schema that the adapter provides.
- **Response Wait Interval**: The permitted value for this property is any interval value (>= 0). This is the time in milliseconds during which the adapter waits for the

report/reply to arrive in `replyToQueueName`. By default, the value of this property is 0 milliseconds. You can change this value, but the value must be less than that of the timeout interval for the outbound activity. If the report/reply message does not arrive in the stipulated time, then the adapter throws an exception. This property is not mandatory.

Note:

The `ResponseWaitInterval` value must be less than the timeout interval for the outbound activity. If the `ResponseWaitInterval` value exceeds the outbound activity timeout, then the adapter can behave ambiguously.

Synchronous Request-Response (Oracle Mediator as Client)

The Oracle MQ Series Adapter also supports the outbound synchronous-solicit-request-response scenario. In this scenario, the adapter enqueues a normal/request message in a queue and expects the report/reply synchronously. The report/reply message arrives in the Reply to Queue Name queue of the normal/request message.

The Synchronous Request-Response scenario for Oracle Mediator as client is same as the Synchronous Request-Response for Oracle BPEL as client. For more information about the Synchronous Request-Response scenario, see Synchronous Request-Response (As Client).

Asynchronous Request-Response (Oracle Mediator As Client)

In this scenario, Oracle Mediator sends a request message and receives the corresponding response from the Mediator callback handler. Oracle Mediator sends an outbound invocation to send the request. The Oracle MQ Series Adapter performs the following operations:

1. Receives message from Oracle Mediator.
2. Formats the XML content as specified at design time in the Adapter Configuration Wizard.
3. Sets properties and a correlation schema on the request message.
4. Sends the message to the queue specified at design time. The third-party application receives the message, processes it, generates the response, and then enqueues the response message to the `replyTo` queue specified in the request message. The Correlation ID and Message ID of the response message is generated based on the correlation schema specified in the request message.
5. The Oracle MQ Series Adapter dequeues the message from the `replyTo` queue.
6. Sets the properties of the message such as priority, expiry, message type, and persistence. These properties are based on the selections that you made in the Adapter Configuration Wizard.
7. Sends the response to the non-initiating receive activity of the BPEL process. To ensure that response is sent to the correct BPEL instance, correlation schemas are used.

Figure 10-12 displays the operation type that you must select in the Adapter Configuration Wizard.

The page that appears after selecting the Send Message to MQ and Get Reply/Reports operation type is shown in Figure 10-13.

You can specify the following properties in this page:

- **Message Type**: The type of the message. You can either send a normal message or a request message.
- **Queue Name**: The name of the queue to which the Oracle MQ Series Adapter enqueues the message. This is a mandatory field.
- **Message Format**: The format of the message.
- **Queue Manager**: The name of the queue manager to which the queue belongs. This field is optional and is necessary when enqueuing message to a remote queue.
- **Priority**: The priority of the message ranging from 0 (low) to 9 (high).
- **Persistence**: The persistence of the message. You can also specify the persistence of the message to be taken from the default persistence attribute, as defined by the destination queue.
- **Delivery Failure**: If delivery of the message fails, then either it can be put to a dead letter queue or it can be discarded.
- **Allow Messages to be Segmented When Necessary**: This is applicable to a message that is big enough for the queue to accommodate. In that case, if you have specified that it has to be segmented, then the single message can be broken into that many bytes the queue can take, which results in mutliple messages.
- **Expiry**: The expiry time of the message. The message is discarded after the expiry time has elapsed.

For more information about these properties, see Message Properties and Report Messages.

The page that is displayed when you click Next in the Send Message to MQ and Get Reply/Reports page can be a Reports page (shown in Figure 10-14) or a Response page (shown in Figure 10-15).

The Reports page shown in Figure 10-14 is displayed only if you have selected the Get Reports option in the Send Message to MQ and Get Reply/Reports page shown in Figure 10-13.

The Response page shown in Figure 10-15 is displayed, irrespective of whether you select the Request or Normal option. The only difference is that if you select the Request option, then `REPLY` is displayed in the Message Type field of the Response page. On the other hand, if you select the Normal option, then `REPORTS` is displayed in the Message Type field of the Response page.

You can select the following types of reports in Figure 10-14:

- Confirmation on Arrival
- Confirmation on Delivery
- Exception Report
- Expiry Report

For information about these report types, see Report Messages.

The Response page, shown in Figure 10-15, is displayed when you click Next in the Reports page.

You can specify the following properties in the Response page:

- **Reply to Queue Name**: The name of reply queue name.
- **Correlation Scheme**: The correlation schema that is used by the Oracle MQ Series Adapter.

  For information about correlation schemas, see Correlation Schemas.
- **Schema Options**: This option enables you to specify the schema for the message to be dequeued.
  - **Choose Other Schema**: This option enables you to choose your schema for the message to be dequeued.
  - **Choose a Predefined Schema**: This option enables you to choose a readymade schema that the adapter provides.

Note:

For Oracle MQ Series Adapter in an asynchronous outbound request/reply scenario, properties are differentiated by an `(Enqueue)` or `(Dequeue)` label in Oracle Enterprise Manager Console. For example, `QueueName(Enqueue)` is used for putting a message and `QueueName(Dequeue)` is used for dequeuing the reply.

When using Oracle Enterprise Manager Console to edit Oracle MQ Series Adapter properties in this scenario, note the following:

- If you change the `ReplyToQueueName(Enqueue)` property, you must also change the `QueueName(Dequeue)` property to the same value.
- If you change the `MessageId(Dequeue)` property, you must also change the `MessageId(Enqueue)` property to the same value.
- If you change the `CorrelationId(Dequeue)` property, you must also change the `CorrelationId(Enqueue)` property to the same value.

When you click Next in the Response page, a Messages page shown in Figure 10-16 is displayed. This page enables you to select the XSD schema file for translation for request and as response message.

For more information about the Messages page, see Enqueue Message.

Outbound Dequeue Scenario

The outbound dequeue scenario dequeues a single message from a queue using the outbound Oracle MQ Series Adapter by using the Get Message from MQ option in the Operation Type page of the Adapter Configuration Wizard. To enable the outbound dequeue option, you must select the Synchronous option, as shown in Figure 10-29.

Click Next in the Send Message to MQ and Get Reply/Reports page, the Response page, as shown in Figure 10-30, is displayed. You must set the following properties in the Response page:

- **QueueName**: This is the name of the MQ Series queue from which the message is dequeued. This property is mandatory.
- **Response Wait Interval**: This is the time (in milliseconds) that the adapter waits if the message is not in the queue. The default value for this property is 0 milliseconds. This property is not mandatory. The permitted value for this property is any integer value (>=0). The value of this property must be less than that of the timeout for outbound activity.

Note:

The `ResponseWaitInterval` value must be less than the timeout interval for the outbound activity. If the `ResponseWaitInterval` value exceeds the outbound activity timeout, then the adapter can behave ambiguously.

Optionally, the following two header properties can be used to filter the dequeued message based on Message Id and Correlation Id.:

- `jca.mq.MQMD.MsgId`: This property sets the message filter option based on the `messageId`. The value provided for this property must be a hexadecimal-encoded value for some `messageId`.
- `jca.mq.MQMD.CorrelId`: This property sets the message filter option based on the `correlationId`. The value provided for this property must be a hexadecimal-encoded value for some `correlationId`.

Message Properties

The Oracle MQ Series Adapter supports the following message properties:

- Messages Types
- Message Format
- Message Expiry
- Message Priority
- Message Persistence

Messages Types

The Oracle MQ Series Adapter supports the following four types of messages:

- Normal Message

  A normal message is sent by one program to another program without expecting any response.

- Request Message

  A request message is sent by one program to another program requesting a response.

- Reply Message

  A reply message is sent by a program in response to a request message.

- Report Message

A report message is sent by a receiving program to a sending program as confirmation of successful or unsuccessful delivery of a message. A report message can be generated for any of the message types, normal message, request message, or reply message.

For more information about acknowledgment messages supported by the Oracle MQ Series Adapter, see Report Messages.

Message Format

You can specify the format for an outgoing message through the Adapter Configuration Wizard, as shown in Figure 10-8. The following message formats are supported:

- No format name (Default)
- Command server request/reply message
- Type 1 command reply message
- Type 2 command reply message
- Dead letter header
- Event message
- User-defined message in programmable command format
- Message consisting entirely of characters
- Trigger message
- Transmission queue header

Message Expiry

You can specify the expiry time for an outgoing message by using the Adapter Configuration Wizard, as shown in Figure 10-8. The queue manager discards the message after the expiry time of a message has elapsed.

If a message has expiration notification set, then a notification is generated when the message is discarded. The notification is sent to the queue specified in the `replyToQueue` parameter. By default, `NEVER` is set for the expiry field.

Message Priority

You can specify the priority of an outgoing message through the Adapter Configuration Wizard, as shown in Figure 10-8. A priority can be in the range of 0 (low) to 9 (high). You can also specify the priority of the message to be taken from the default priority attribute, as defined by the destination queue. By default, `AS_Q_DEF` is set as message priority.

Message Persistence

You can specify the persistence of an outgoing message through the Adapter Configuration Wizard, as shown in Figure 10-8. If message persistence is not set, then a message is lost when the queue manager restarts or there is a system failure. If you set persistence for a message to `true`, then it means that the message does not get lost even if there is system failure or the queue manager is restarted. You can also specify the persistence of the message to be taken from the default priority attribute, as

defined by the destination queue. The Adapter writes persistent messages to log files and queue data files. If a queue manager is restarted after a failure, it recovers these persistent messages from these files.

Note:

You can specify all these message properties at run time through message headers. You can use the assign activity to assign values to these properties.

Correlation Schemas

Mapping a response to a request in a request-reply interaction requires correlation. Each MQ Series request message contains a message ID and a correlation ID. When an application receives a request message from Oracle BPEL PM, it checks for the correlation schema defined for the response message. Based on the correlation schema, the application generates the message ID and correlation ID of the response message.

The response page of the Adapter Configuration Wizard shown in Figure 10-15 enables you to specify the correlation schema for the response message.

The Message ID box shown in Figure 10-15 provides the following options for the message ID of the response message:

- Generate a new message ID for the response message.
- Use the message ID of the request message.

Similarly, the Correlation ID box shown in Figure 10-15 provides the following options for the correlation ID of the response message:

- Use the message ID of the request message
- Use the correlation ID of the request message

Distribution List Support

The Oracle MQ Series Adapter enables you to enqueue a message to multiple queues.

When you select the Put Message Into MQ option in the Operation Type page and multiple queues, then the `DistributionList` parameter is automatically added to the JCA file.

Report Messages

The Oracle MQ Series Adapter enables you to set various types of acknowledgment messages on an outgoing message. These acknowledgment messages are known as report messages. A report message is generated, only if the criteria for generating that report message is met. When enqueuing a message on a queue, you can request for more than one type of report message. When you request for a report message, you must specify the queue name to which the report message is sent. This queue is known as `replyTo` queue. A report message can be generated by a queue manager, a message channel, or an application.

The Oracle MQ Series Adapter supports the following message reports:

- Confirmation on Arrival

The Confirmation on Arrival (COA) message indicates that the message has been delivered to the target queue manager. A COA message is generated by the queue manager. This message report can be selected in the Reports page of the Adapter Configuration page shown in Figure 10-14.

- Confirmation on Delivery

  A Confirmation on Delivery (COD) message indicates that the message has been retrieved by the receiving application. A COD message is generated by the queue manager. This message report can be selected in the Reports page shown in Figure 10-14.

- Exception Report

  An exception report is generated when a message cannot be delivered to the specified destination queue. Exception reports are generated by the message channel. This message report can be selected in the Reports page of the Adapter Configuration page shown in Figure 10-14.

- Expiry Report

  An expiry report indicates that the message was discarded because the expiry time specified for the message elapsed before the message was retrieved. An expiry report is generated by a queue manager. This message report can be selected in the Reports page of the Adapter Configuration page shown in Figure 10-14.

- Positive Action Notification

  A Positive Action Notification (PAN) indicates that a request has been successfully processed. It means that the action requested in the message has been performed successfully. This type of report is generated by the application.

- Negative Action Notification

  A Negative Action Notification (NAN) indicates that a request has not been successfully serviced. It means that the action requested in the message has not been performed successfully. This type of report is generated by the application.

You can specify whether all these report messages except PAN and NAN should contain the complete original message, a part of the original message, or no part of the original message. You can select any of the following options in the Adapter Configuration Wizard:

- No data from the original message
- The first 100 bytes of data in the original message
- The entire original message

Message Delivery Failure Options

The Message Delivery Failure options are supported only for remote queues and not for normal queues. The Oracle MQ Series Adapter enables you to specify the action that should be taken in case a message could not be delivered to the destination queue. You can specify any of:

- Place message on a dead letter queue

  This is the default action. A message is placed on a dead-letter queue if it cannot be delivered to the destination queue. A report message is generated if requested by the sender.

- Discard message

  This indicates that the message should be discarded if it cannot be delivered to the destination queue. A report message is generated if requested by the sender.

You can specify these options by selecting the Put Message To MQ option in the Adapter Configuration Wizard.

Message Segmentation

The Oracle MQ Series Adapter supports message segmentation for both inbound and outbound interactions. Segmentation is required when the size of a message is greater than the message size allowed for a queue. A physical message is divided into two or more logical messages. All logical messages have the same group ID and a sequence number, and an offset.

In the inbound interaction, the segmentation is inherently supported by the Oracle MQ Series Adapter. The Oracle MQ Series Adapter dequeues all logical messages in the order of sequence number and then publishes the single message as XML to Oracle BPEL PM or Mediator.

The Allow Messages to Be Segmented When Necessary option enables you to segment messages for outbound interactions. This option appears in the Response page of the Adapter Configuration Wizard.

The message is segmented based on whether the size of the message is larger than the maximum limit set on the queue.

Message Segmentation is not supported when using Distribution Lists. This is due to a limitation in the Java API of the IBM MQ series product that is used by the MQ Adapter.

Integration with CICS

The Oracle MQ Series Adapter provides support for sending and receiving messages from the CICS server. In the inbound direction, an inbound message from the CICS server is dequeued in the same way as a normal message. In the outbound direction, the message should be in the CICS format. A sample schema file for the outbound CICS message format is shown in the following XSD example. You should also refer to the list of supported Oracle MQ Series Adapter encodings in Oracle MQ Series Adapter Encodings.

```
<?xml version="1.0" ?><schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/cics_mqcih"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"

        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        nxsd:version="NXSD"

        nxsd:encoding="UTF8"
        nxsd:stream="bytes"
        nxsd:byteOrder="bigEndian"

        xmlns:nxsd_extn="http://xmlns.oracle.com/pcbpel/nxsd/extensions"

<element name="MSGForMQCICSBridge">
    <complexType>
      <sequence>
        <element name="MQCIH">
          <complexType>
```

```
<sequence>
  <!--
  MQCHAR4   StrucId;
  Structure identifier
  -->
  <element name="StrucId" type="string"
   nxsd:style="fixedLength" nxsd:length="4" nxsd:padStyle="tail"/>

  <!--
  MQLONG    Version;
  Structure version number 1 or 2
  -->
  <element name="Version" type="string"
   nxsd:style="integer" nxsd_extn:octet="4"
   nxsd_extn:align="0" nxsd_extn:sign="unticked" />
  <!--
  MQLONG    StrucLength;
  Length of MQCIH structure V1=164 V2=180
  -->
  <element name="StrucLength" type="string"
   nxsd:style="integer" nxsd_extn:octet="4"
   nxsd_extn:align="0" nxsd_extn:sign="unticked" />

  <!--
  MQLONG    Encoding;
  Reserved
  -->
  <element name="Encoding" type="string"
   nxsd:style="integer" nxsd_extn:octet="4"
   nxsd_extn:align="0" nxsd_extn:sign="unticked" />

  <!--
  MQLONG    CodedCharSetId;
  Reserved              -->
  <element name="CodedCharSetId" type="string"
   nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
   nxsd_extn:sign="unticked" />

  <!--
  MQCHAR8   Format;
  MQ Format name
  -->
  <element name="Format" type="string"
  nxsd:style="fixedLength" nxsd:length="8" />

  <!--
  MQLONG    Flags;
  Reserved
  -->
  <element name="Flags" type="string"
   nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"

   nxsd_extn:sign="unticked" />

  <!--
  MQLONG   ReturnCode;
  Return code from bridge
  -->
  <element name="ReturnCode" type="string"
   nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
   nxsd_extn:sign="unticked" />
```

```
<!--
MQLONG   CompCode;
MQ completion code or CICS EIBRESP
-->
<element name="CompCode" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />

<!--
MQLONG   Reason;
MQ reason or feedback code, or CICS EIBRESP2
-->
<element name="Reason" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />

<!--
MQLONG   UOWControl;
Unit-of-work control
-->
<element name="UOWControl" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />

<!--
MQLONG   GetWaitInterval;
Wait interval for MQGET call issued by bridge
-->
<element name="GetWaitInterval" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="ticked" />

<!--
MQLONG   LinkType;
Link type
-->
<element name="LinkType" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
nxsd_extn:sign="unticked" />

<!--
MQLONG   OutputDataLength;
Output commarea data length
-->
<element name="OutputDataLength" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="ticked" />

<!--
MQLONG   FacilityKeepTime;
Bridge facility release time
-->
<element name="FacilityKeepTime" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />

<!--
MQLONG   ADSDescriptor;
Send/receive ADS descriptor
-->
```

```
<element name="ADSDescriptor" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />

<!--
MQLONG   ConversationalTask;
Whether task can be conversational
-->
<element name="ConversationalTask" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />
            <!--
MQLONG   TaskEndStatus;
Status at end of task
-->
<element name="TaskEndStatus" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />


<!--
MQBYTE   Facility[8];
BVT token value. Initialise as required.
-->
<element name="Facility" type="string"
 nxsd:style="integer" nxsd_extn:octet="8" nxsd_extn:align="0"
 nxsd _extn:sign="unticked" />

<!--
MQCHAR4  Function;
MQ call name or CICS EIBFN function name
-->
<element name="Function" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  AbendCode;
Abend code
-->
<element name="AbendCode" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR8  Authenticator;
Password or passticket
-->
<element name="Authenticator" type="string"
nxsd:style="fixedLength" nxsd:length="8" />

<!--
MQCHAR8  Reserved1;
Reserved
-->
<element name="Reserved1" type="string"
 nxsd:style="fixedLength" nxsd:length="8" />

<!--
MQCHAR8  ReplyToFormat;
MQ format name of reply message
-->
<element name="ReplyToFormat" type="string"
```

```
 nxsd:style="fixedLength" nxsd:length="8" />

<!--
MQCHAR4  RemoteSysId;
Remote sysid to use
-->
<element name="RemoteSysId" type="string"
nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  RemoteTransId;
Remote transid to attach
-->
<element name="RemoteTransId" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  TransactionId;
Transaction to attach
-->
<element name="TransactionId" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  FacilityLike;
Terminal emulated attributes
-->
<element name="FacilityLike" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  AttentionId;
AID key
-->
<element name="AttentionId" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  StartCode;
Transaction start code
-->
<element name="StartCode" type="string"
nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  CancelCode;
Abend transaction code
-->
<element name="CancelCode" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  NextTransactionId;
Next transaction to attach
-->
<element name="NextTransactionId" type="string"
nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR8  Reserved2;
Reserved
```

```
                -->
                <element name="Reserved2" type="string"
                nxsd:style="fixedLength" nxsd:length="8" />
                <!--
                MQCHAR8  Reserved3;
                Reserved
                -->
                <element name="Reserved3" type="string"
                nxsd:style="fixedLength" nxsd:length="8" />

                <!--
                MQLONG   CursorPosition;
                Cursor position
                -->
                <element name="CursorPosition" type="string"
                nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
                nxsd_extn:sign="unticked" />

                <!--
                MQLONG   ErrorOffset;
                Error offset
                -->
                <element name="ErrorOffset" type="string"
                 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
                 nxsd_extn:sign="unticked" />

                <!--
                MQLONG   InputItem;
                Input item
                -->
                <element name="InputItem" type="string"
                 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
                 nxsd_extn:sign="unticked" />

                <!--
                MQLONG   Reserved4;
                Reserved
                -->
                <element name="Reserved4" type="string"
                nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
                nxsd_extn:sign="unticked" />
              </sequence>
            </complexType>
          </element>

          <!--
          Application data
          -->
          <element name="ApplicationData" type="string"
                                       fixed="Nothing" />

        </sequence>
      </complexType>
    </element>
  </schema>
```

Using the MQ Series Client Channel Definition Table Feature

The CCDT provides several advantages when defining properties while configuring the JNDI for your MQ Series Adapter.

It is useful to understand how the Client Channel Definition Table works.

To understand the configuration of the Client Channel Definition Table, you should understand the following basic terms and concepts:

- **Channel name and Connection name**: The channel name for each definition in the table should be exactly same as the server connection channel of the queue manager which has a listener running on the Connection name provided. For example, the first definition in the CCDT below, `channel.ccdt_qm1` is the name of the server connection channel of a queue manager which has its listener running on localhost(1414).

- **Queue manager name**: The queue manager name defined in the CCDT might not be the actual name of the queue manager with Channel name and Connection name defined. There can be the following possibilities:

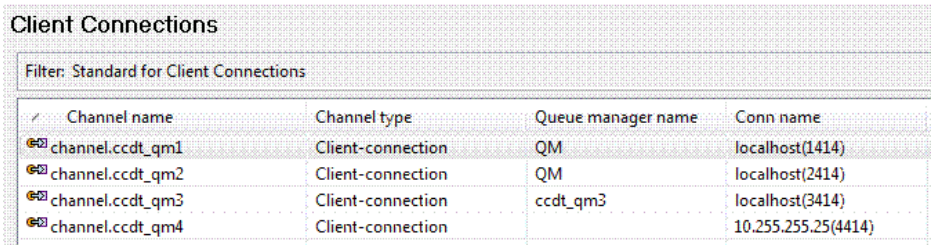
  – The Queue manager name defined in the client channel is the actual queue manager name as in the third definition in the table. Hence, `ccdt_qm3` is the actual name of the queue manager which has the `channel.ccdt_qm3` as its server connection channel and has a listener listening on localhost(3414).

  – No Queue manager name is defined for a client channel definition. In Figure 10-31, this situation can be seen for the fourth definition in the table which does not have a Queue manager name defined.

  – Two or more client channel entries in the CCDT have the same Queue manager name defined. This name can match with the actual queue manager name of any of the entries. The first and second definition in Figure 10-31 have the same Queue manager name defined as `QM` though they actually point to different queue managers.

    Figure 10-31shows a standard CCDT which has client channels defined for connections to four queue managers; that is, `ccdt_qm1` to `ccdt_qm4`.

    Figure 10-31 Sample CCDT

    Once defined, this CCDT resides, by default, at the following locations:

    - For Windows, `<C:\Program Files\IBM\ MQ\qmgrs\QM_NAME\@ipcc \AMQCLCHL.TAB>`
    - For Unix-based systems,`<var/mqm/qmgrs/QM_NAME/@ipcc/ AMQCLCHL.TAB>`

This feature simplifies what was formerly a task that involved defining the Hostname, PortNumber, ChannelName and QueueManagerName in ConnectionFactory properties while you were configuring the JNDI for your MQ Series Adapter.

- You can use the Client Channel Definition Table feature (CCDT), with only the URL pointing to a CCDT file and, additionally, a QueueManagerName, as part of the configuration. The MQ Series Adapter is able to read the rest of the connection

details from the CCDT to create the required connections for your MQ Series Adapter.

- You can configure the MQ Series Adapter to dynamically connect to the available queue manager from a list of queue managers, a few of which might be down.
- You do not have to change the JNDI used for the MQ Adapter in the composite process and then re-deploy the composite to change the queue manager on which the MQ adapter was required to connect.
- Only one ConnectionFactory JNDI entry is required for the queue managers and the MQ adapter will dynamically connect to the first available queue manager.
- CCDT can be used to define a list of queue managers with the same Queue Manager name property for the client channel definition. This list is used by the MQ Series Adapter to dynamically connect to the first available Queue Manager at runtime without requiring any configuration changes or redeployment.

You can gain a basic understanding of IBM MQ and MQ adapter usage and details about CCDT creation in the IBM MQ Series documentation.

IBM MQ version 6 is the minimum requirement for the CCDT feature to work.

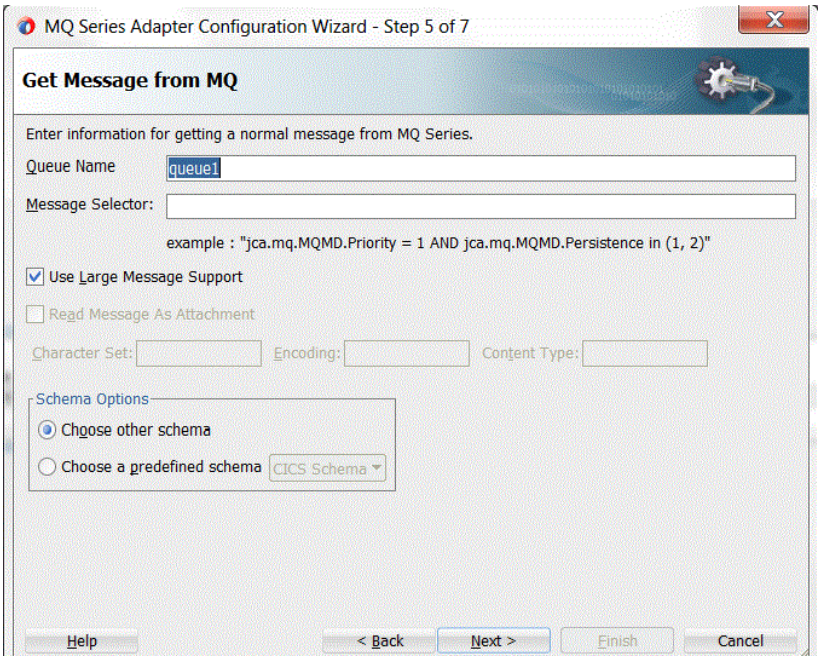
Large Payload Support

MQ Adapter now supports reading and writing large payloads as MQ messages. See the following sections for information about configuring the inbound and the outbound MQ Series Adapter for large payloads.

Configuring the Inbound MQ Adapter for Large Payloads

For inbound processing, you can configure the inbound MQ Adapter for large payloads by the selecting the **Use Large Message Support** checkbox while configuring the inbound adapter through the MQ Series Adapter Configuration Wizard. The following screenshot shows this.

Figure 10-32 Configuring Large Message Support at inbound in MQ Series Adapter



Configuring the Outbound MQ Adapter for Large Payloads

At outbound, there is no required configuration for large payload support. But two optional configurations are important for outbound processing:

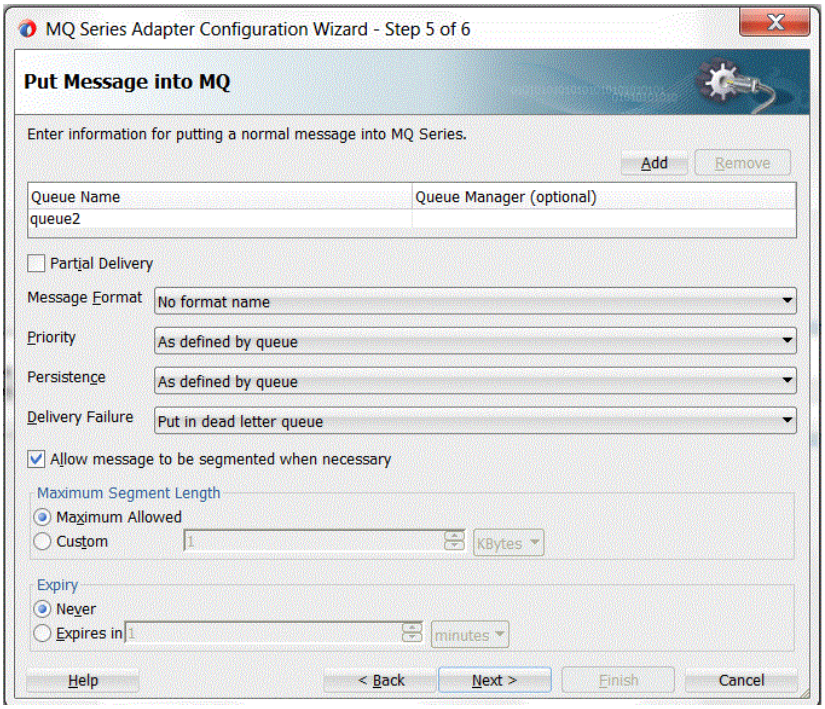
1. `SegmentIfRequired` property: By default, this property is set to TRUE; you can configure this by using the **Allow message to segmented when necessary** checkbox while configuring the outbound MQ adapter. You must set this boolean property to true if you want the MQ Adapter to break large messages into smaller segments. If this property is set to false, and the size of the message exceeds the maximum message size allowed by the queue, channel or queue manager, the MQ server does not permit putting the message into the queue and the MQ Adapter throws an exception.

2. `MaximumSegmentLength`: By default set to `Maximum Allowed`, you can configure this property by using the `Maximum Segment Length`, radio buttons while configuring the outbound MQ adapter through the wizard. By selecting the `Custom`, radio button, you can configure a user-defined message segment length and the adapter segments the message in that size. When `Maximum Allowed` radio button is selected, the MQ Adapter segments the messages into the maximum segment size that the outbound queue, channel and queue manager allow.

Note that if the custom segment value provided is higher than that allowed on the MQ server, then the value specified by the server will be honored.

The following figure shows the two configurations.

Figure 10-33 Configuring Large Message Support at Outbound in MQ Series Adapter



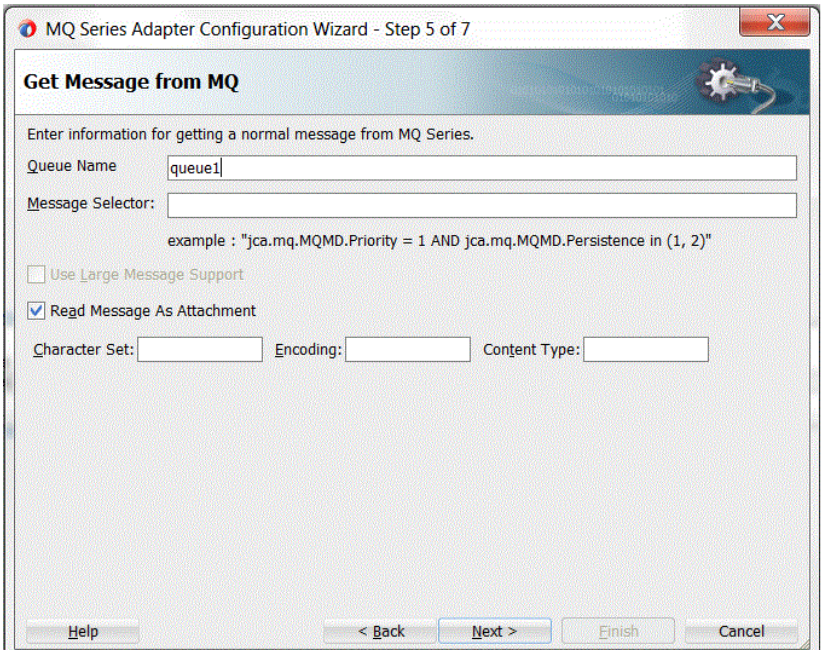
Attachment Support

The MQ Adapter supports processing messages as attachment. When processing inbound messages as attachment, the MQ Adapter ignores translating the payload and directly transfers the message to the next component. Similarly, at outbound, when

writing as an attachment, the MQ Adapter ignores translating the payload and opaquely writes the message to the queue. This feature is useful when opaquely transferring large messages for one queue to another.

You can configure the MQ Adapter to process a message as an attachment by selecting the **Read Message As Attachment** checkbox while configuring the inbound adapter using the configuration wizard. More details can be found in the use case section 10.6.9.

Figure 10-34 Configuring Inbound MQ Adapter to Read Message as Attachment



Configuring the Oracle MQ Series Adapter

The prerequisites for using the Oracle MQ Series Adapter are:

- IBM MQ server should be installed and running.
- A queue manager and a server connection channel should be created.

Note:

You must create queues based on the requirement of the application.

To configure the Oracle MQ Series Adapter, perform the following:

- Adding jar Files to the Classpath: MQ Series 6 and 7
- Adding JNDI Entry
- Enabling Binding Mode for Connections

Adding jar Files to the Oracle MQ Series Adapter Classpath: MQ Series 6 and 7

The steps in this section should be performed once, before using the Oracle MQ Series Adapter.

To add correct jar properties to the classpath for the Oracle MQ Series 6 Adapter, copy the following jar to *<DOMAIN_HOME>/lib* folder

- `com.ibm.mq.jar`

To add correct jar properties to the classpath for the Oracle MQ Series 7 Adapter, copy the following jars to *<DOMAIN_HOME>/lib* folder

- `com.ibm.mq.commonservices.jar`
- `com.ibm.mq.jar`
- `com.ibm.mq.pcf.jar`
- `com.ibm.mq.headers.jar`
- `com.ibm.mq.jmqi.jar`
- `com.ibm.mqetclient.jar` (for use with XA. Note that for any version prior to 7.5, you must have this `com.ibm.mqetclient.jar` for XA Transaction Scenarios. The jar is not provided with normal installation of IBM MQ Series. To procure this JAR, you must obtain extra licenses from IBM.)

In addition, if you are using the Oracle MQ Series 7 Adapter , the new Sharing Conversation property of the Server Connection Channel has to be to set to zero.

For version 7.5 of IBM MQ Series, you must have the following jars:

- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mqjms.jar
- dhbcore.jar

The `com.ibm.mqetclient.jar` is not required with the MQ Series 7.5 server. Running XA transactions will work without the presence of that jar if you are using MQ Series 7.5.

Adding JNDI Entry

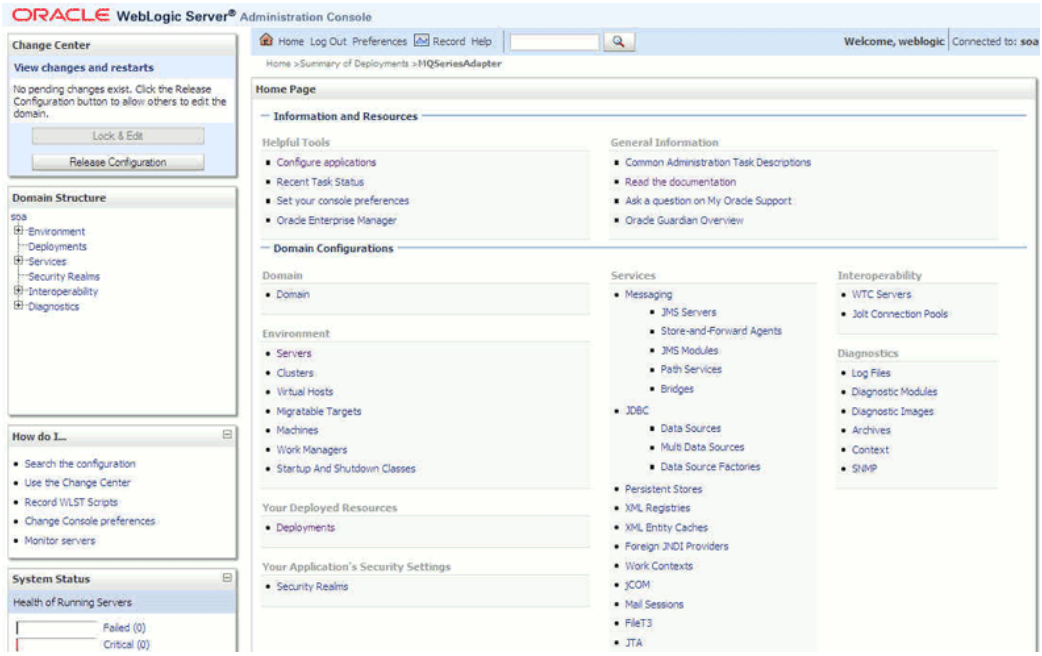
You can add a new JNDI entry in the Oracle WebLogic Server Administration Console by following these steps:

1. Log in to the following URL using the username/password to open the Oracle WebLogic Server Administration Console:

   http://<localhost>:port/console

   The Home page is displayed, as shown in Figure 10-35.

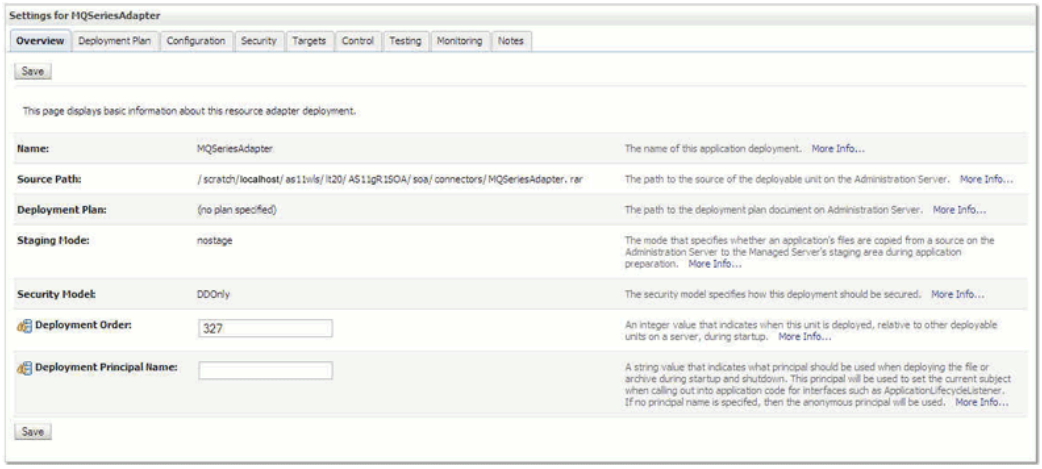
Figure 10-35 Oracle WebLogic Administration Console Home Page



2. Under Domain Structure, in the left pane, click **Deployments**. The Summary of Deployments page is displayed.

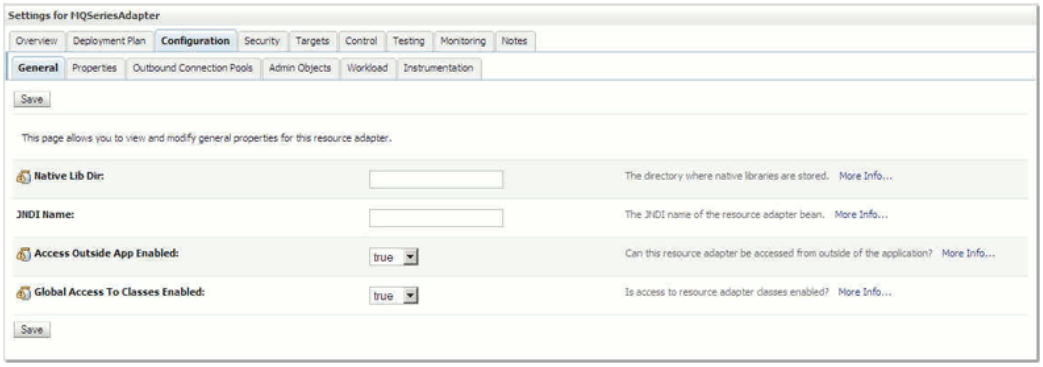
3. Click **MQSeriesAdapter**. The Settings of MQSeriesAdapter page is displayed, as shown in Figure 10-36.

Figure 10-36 Settings of MQSeriesAdapter Page



4. Click the **Configuration** tab. The Configuration submenu options are displayed, as shown in Figure 10-37.

Figure 10-37 Settings of MQSeriesAdapter Page - Configuration Submenu Options

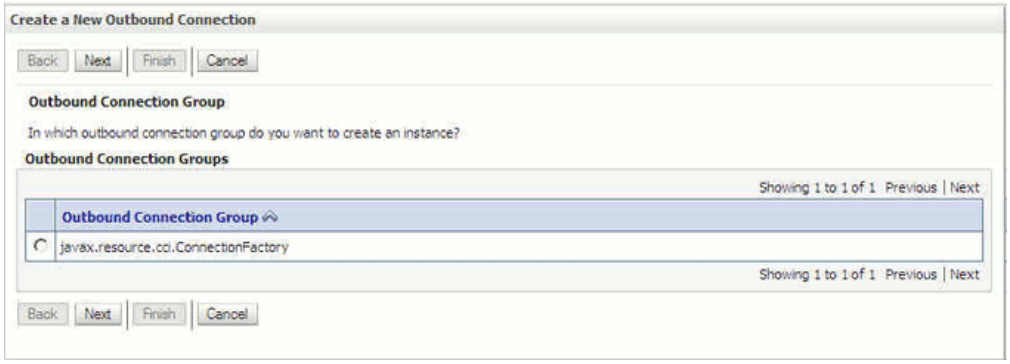


5. Click **Outbound Connection Pools**. The Outbound Connection Pool Configuration Table is displayed, as shown in Figure 10-38.

Figure 10-38 Outbound Connection Pool Configuration Table

6. Click **New**. The Create a New Outbound Connection page is displayed, as shown in Figure 10-39.

Figure 10-39 Create a New Outbound Connection Page



7. Select the **javax.resource.cci.ConnectionFactory** option, and click **Next**.

8. Enter a value in the JNDI Name field, for example `eis/MQ/MQAdapter`, as shown in Figure 10-40.

Figure 10-40 Create a New Outbound Connection Page - JNDI Name

9. Click **Finish**. The Save Deployment Plan Assistant page is displayed.
10. Click **OK**. You have successfully created a JNDI name.

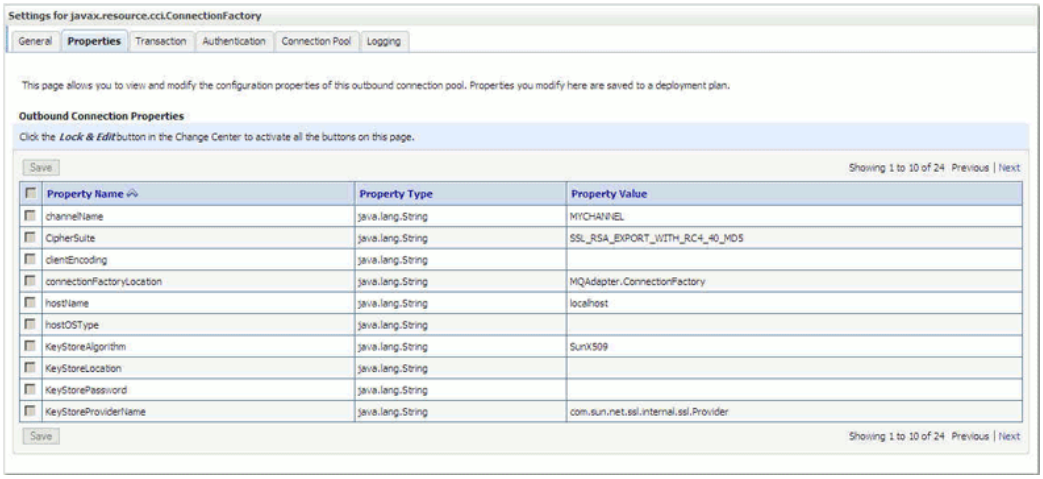
Enabling Binding Mode for Connections

You can enable binding mode for connections for the Oracle MQ Series Adapter by modifying a few properties in the Oracle WebLogic Server Administration Console:

To enable binding mode, perform the following steps:

1. Log in to the Oracle WebLogic Server Administration Console using your password credentials.
2. Under Domain Structure, in the left pane, click **Deployments**. The Summary of Deployments page is displayed.
3. Click **MQSeriesAdapter**. The Settings of MQSeriesAdapter page is displayed.
4. Click the **Configuration** tab. The Configuration submenu options are displayed.
5. Click **Outbound Connection Pools**. The Outbound Connection Pool Configuration Table is displayed.
6. Click the + icon next to javax.resource.cci.ConnectionFactory. A list of JNDIs are displayed.
7. Select, **eis/MQ/MQAdapter**, the JNDI that you created in the Adding JNDI Entry. The Outbound Connection Properties page is displayed, as shown in Figure 10-41, with a list of 24 properties.

Figure 10-41 Outbound Connection Properties Page



8. Set the following parameters:

   - **hostName**: This value should always be blank.
   - **portNumber**: This value should contain some unused port numbers. For example, 44888.
   - **channelName**: This value should always be blank.
   - **queueManagerName**: This value is a valid queue manager name.

You have enabled the binding mode for connections for the Oracle MQ Series Adapter.

Selective Dequeue of Messages Using Message Selectors

When two or more BPEL processes receive messages from the same queue, the messages are not assured of delivery to the correct BPEL process and can be associated with any of the BPEL processes listening on the queue.

However, you can set up processes so only one process listens for messages from any particular queue.

To accomplish this, you can configure the MQ adapter to dequeue only those messages that satisfy the message selection criteria and discard those which do not.

These messages can then be received by any other BPEL process listening for messages from the same queue, if the message satisfies its selection criteria.

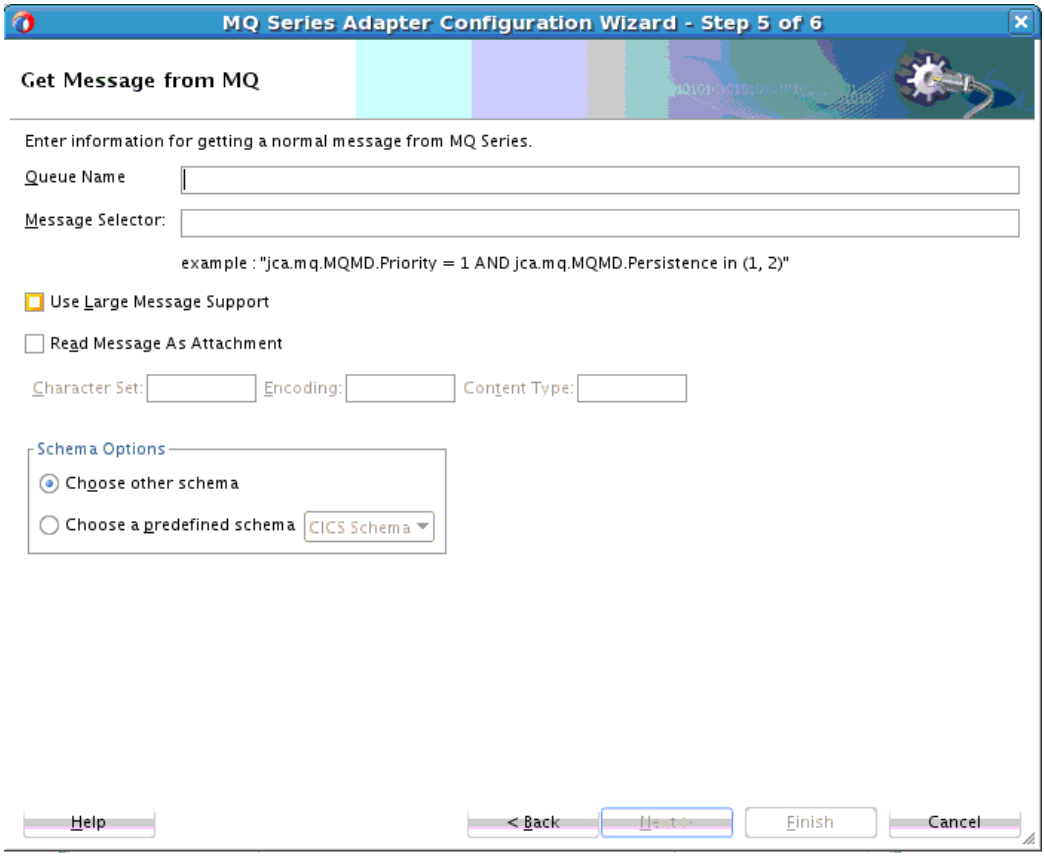
Before examining the syntax and rules associated with the Message Selector, it is useful to look at the Message Selector in the context of the MQ Adapter Configuration Wizard.

Note that the message selectors of the all the BPEL processes receiving messages from the same queue must be exclusive of each other.

Message Selector in the MQ Adapter Configuration Wizard

The Message Selector feature appears on the Get Messages from MQ Screen in the MQ Adapter Wizard. When you specify a Message Selector, the Wizard places the selector information you specify in the activation-spec. As you can see in Figure 10-42, the Message Selector field is directly below the Queue Name field.

Figure 10-42 Message Selector Field on the Get Message from MQ Screen in the MQ Adapter Configuration Wizard



Using Message Selectors with MQ

A message selector enables you to specify which messages the selector is interested in to dequeue from the MQ queue. Only those messages whose headers and properties match the selector are dequeued.

A message satisfies the selector if the selector evaluates to true when the message's header fields and property values are substituted for their corresponding identifiers in the selector.

A message selector is defined as an activation spec property when you use the MQ Configuration Wizard Adapter to configure the inbound MQ Adapter. A sample message selector follows:

Example - Sample Message Selector for Inbound MQ Adapter

```
<property name="MessageSelector" value="((jca.mq.MQMD.Priority BETWEEN 0 AND 3) AND
NOT(jca.mq.MQMD.Priority = 1)) AND jca.mq.MQMD.PutApplName LIKE 'WebSph_re MQ%' AND
jca.mq.MQMD.MsgType NOT IN (0, 1) AND (jca.mq.MQMD.ReplyToQ = 'test_q2' OR
jca.mq.MQMD.ReplyToQ IS NULL) AND name = 'Joe'"/>
```

If the MessageSelector is not defined, the MQ Adapter does not do any message selection and dequeues all the messages that arrive on the queue.

Message Selector Syntax: Literals

This section describes in detail the MessageSelector syntax. You must understand this syntax to fill in the selector information in the Selector box in the Get MQ Message screen that provides you the option to create a message selector.

The order of evaluation of the message selector is from left to right within the precedence level. Parenthesis can be used to change this order.

A selector can contain the following literals:

- **String**: A string literal should be enclosed in single quotes. Like Java String literals, these would use the Unicode character encoding.
- **Exact Numeric:** An exact numeric literal is a numeric value without a decimal point. Numbers in the range of Java Long will be supported.
- **Approximate Numeric**: An approximate numeric literal is a number with a decimal point. All numbers in the range of Java double will be supported.
- **Boolean**: True or False.

Message Selector Identifiers

Use the identifiers to identify the message header and the property values in the message selector. The message header properties are the MQMD properties and user defined properties and those defined in the USR folder of RFH2 header. Properties defined in other RFH2 folders are not supported for message selectors. Note that Identifiers

- Are case-sensitive
- Cannot be "AND", "OR", "NOT", "TRUE", "FALSE", "IN", "LIKE", "BETWEEN", "IS", "NULL", "ESCAPE" or "div".
- Have a naming convention. Any identifier name that does not begin with 'jca.mq.MQMD.' will be considered as a USR folder property identifier of the RFH2 header.

In detail, the following constitute the identifiers.

- MQMD header fields. See Table 10-5.
- RFH2 USR Folder. Any property name defined in the USR folder of the RFH2 header of the MQ message can be used as an identifier. The property name can be an unlimited length character sequence that must begin with a Java Identifier start character and all the following characters must be Java Identifier part characters (which can include '.', '_' and '$').

  Use of any other RFH2 Folder properties in message selector is not supported. Table 10-5 provides MQMD headers and Message Selector identifiers.

Table 10-5 MQMD Headers and Message Selector Identifiers

MQMD header field	Identifier
Accounting Token	jca.mq.MQMD.AccountingToken
Application Identity Data	jca.mq.MQMD.ApplIdentityData
Application Origin Data	jca.mq.MQMD.ApplOriginData

Table 10-5 (Cont.) MQMD Headers and Message Selector Identifiers

MQMD header field	Identifier
Backout Count	jca.mq.MQMD.BackoutCount
Coded Character Set Id	jca.mq.MQMD.CodedCharSetId
Correlation Id	jca.mq.MQMD.CorrelId
Encoding	jca.mq.MQMD.Encoding
Expiry	jca.mq.MQMD.Expiry
Feedback	jca.mq.MQMD.Feedback
Format	jca.mq.MQMD.Format
Group Id	jca.mq.MQMD.GroupId
Message Flags	jca.mq.MQMD.MsgFlags
Message Id	jca.mq.MQMD.MsgId
Message Sequence Number	jca.mq.MQMD.MsgSeqNumber
Message Type	jca.mq.MQMD.MsgType
Offset	jca.mq.MQMD.Offset
Original Length	jca.mq.MQMD.OriginalLength
Persistence	jca.mq.MQMD.Persistence
Priority	jca.mq.MQMD.Priority
Put Application Name	jca.mq.MQMD.PutApplName
Put Application Type	jca.mq.MQMD.PutApplType
Put Date Time	jca.mq.MQMD.PutDateTime
Reply to Queue Manager Name	jca.mq.MQMD.ReplyToQMgr
Reply to Queue Name	jca.mq.MQMD.ReplyToQ
Report	jca.mq.MQMD.Report
User Identifier	jca.mq.MQMD.UserIdentifier

Message Selector Expressions

Expressions within message selectors are of these types:

- **Arithmetic**: Composed of themselves, arithmetic operations and numeric literals.
- **Conditional**: Composed of themselves, comparison operations, logical operations and Boolean literals.

Message Selector Operators

Message Selector operators are of two types:

- **Bracketing**: Use standard brackets () for ordering expression evaluation.
- **Logical**: Logical operators in precedence order are NOT, AND, OR

Message Selector Comparison

Message Selector Comparisons include: : =, >, >=, <, <=, <> (not equal)

Note the following:

- Only like type values are allowed to be the subjects of comparison. One exception is that it is valid to compare exact numeric values and approximate numeric values.
- String and Boolean comparison is restricted to = and <>. Two strings will be equal if and only if they contain the same sequence of characters.

Message Selector Arithmetic

Arithmetic operators in precedence order are:

- +, - (unary)
- *, / (multiplication and division)
- +, - (addition and subtraction)

Message Selector Advanced Operators

See for a list of message selector advanced operators.

Table 10-6 Message Selector Advanced Operators

Operator	Syntax and Example
`[NOT] BETWEEN`	Syntax : `Identifier [NOT] BETWEEN arithmetic-expr1 AND arithmetic-expr2` • Example: `jca.mq.MQMD.Priority BETWEEN 0 AND 3` means the message priority should be >= 0 and <= 3. • Example: `jca.mq.MQMD.Priority NOT BETWEEN 2 AND 4` means the message priority should be less than 2 or greater then 4.
`[NOT] IN`	Syntax : `Identifier [NOT] IN (literal1, literal2, …)` • Example: `jca.mq.MQMD.MsgType IN (0, 1)`means the message type should be either 0 or 1. • Example: `jca.mq.MQMD.MsgType NOT IN (0, 1, 2)`means the message type should not be 0, 1 or 2.

Table 10-6 (Cont.) Message Selector Advanced Operators

Operator	Syntax and Example
`[NOT] LIKE`	Syntax : `Identifier [NOT] LIKE pattern-value` The pattern-value is a string literal where '_' stands for any single character, '%' stands for any sequence of characters, including the empty sequence, and all other characters stand for themselves. • `phone LIKE '12%3'` is true for '123' or '12993' and false for '1234' • `word LIKE 'l_se'` is true for 'lose' and false for 'loose' • `phone NOT LIKE '12%3'` is false for '123' and '12993' and true for '1234'
`IS [NOT] NULL`	Syntax : `Identifier IS [NOT] NULL` • `prop_name IS NULL` is true when prop_name is null and false when prop_name is not null • `prop_name IS NOT NULL` is true when prop_name is not null and false when prop_name is null

Message Selector Example

A Message Selector example with additional selectors follows. You can then see the results from applying this selector against a sample message.

Example - Message Selector with Additional Selectors

```
<property name="MessageSelector" value="((jca.mq.MQMD.Priority BETWEEN 0 AND 3) AND
NOT(jca.mq.MQMD.Priority = 1)) AND jca.mq.MQMD.PutApplName LIKE 'WebSph_re MQ%' AND
jca.mq.MQMD.MsgType NOT IN (0, 1) AND (jca.mq.MQMD.ReplyToQ = 'test_q2' OR
jca.mq.MQMD.ReplyToQ IS NULL) AND name = 'Joe'"/>
```

See Table 10-7, which lists sample messages and shows which messages would satisfy this selector and which do not.

Table 10-7 Sample Messages for Message Selector Example

Message Properties	Message 1	Message 2	Message 3	Message 4
Priority	2	2	3	1
Put Application Name	WebSphere MQ7	WebSphere MQ6	WebSphere MQ6	WebSphere MQ
Message Type	8	8	2	4
Reply to Queue Name		test_q2	test_q2	
USR RFH2 folder	name = Ankur age = 24 phone = 123456	name = Bob age = 42 phone = 654321	name = Ankur empId = 3321	color = Red theme = old
MessageSelector Pass?	True	False	False	False

Table 10-7 (Cont.) Sample Messages for Message Selector Example

Message Properties	Message 1	Message 2	Message 3	Message 4
Comments		The name property in USR Folder does not match the selector	The put application name property does not match	The priority and name property in USR folder does not match

Message Selector Use Case: One BPEL Process Receiving Selective Messages from a MQ Queue using Message Selector

This example will show you how to dequeue selective messages from an MQ Series Queue using the MQ series Adapter. For example, assuming you want to dequeue only those messages which satisfy all the following criteria:

- Message ID starts with 414D51206C6F63616C5F716D
- Priority > 4
- Message Type is not 2
- Reply to Queue Name is either 'queue12' or 'queue49'

  See Table 10-8 for an analysis that assists in creating a message selector for this set of criteria, where each set of criteria is matched with Message Selector text.

Table 10-8 Message Selector Table for One BPEL Process Use Case

Criteria	Message Selector text
Message ID starts with 414D51206C6F63616C5F716D	`jca.mq.MQMD.MsgId LIKE '414D51206C6F63616C5F716D%'`
Priority > 4	`jca.mq.MQMD.Priority > 4`
Message Type is not 2	`NOT (jca.mq.MQMD.MsgType = 2)`
Reply to Queue Name is either 'queue12' or 'queueu49'	`jca.mq.MQMD.ReplyToQ IN ('queue12', 'queue49')`

Thus, the complete message string you would enter in the MQ Series Adapter Configuration Wizard, Get Message screen would be:

```
jca.mq.MQMD.MsgId LIKE '414D51206C6F63616C5F716D%'
    AND jca.mq.MQMD.Priority >
      AND NOT (jca.mq.MQMD.MsgType = 2)
       AND jca.mq.MQMD.ReplyToQ IN ('queue12', 'queue49')
```

Thus, the `InboundAdapter.jca` JCA file the Wizard creates contains the following to indicate the message selector within the following tag as a child of the <activation-spec> tag:

```
<property name="MessageSelector" value=
     " jca.mq.MQMD.MsgId LIKE '414D51206C6F63616C5F716D%'
```

```
AND jca.mq.MQMD.Priority > 4 AND NOT
 (jca.mq.MQMD.MsgType = 2) AND jca.mq.MQMD.ReplyToQ
  IN ('queue12', 'queue49')"/>
```

Usage with Sample Messages

The MQ Adapter configured with the message selector only dequeues messages that satisfy the message selector that has been defined. Some examples can clarify the messages that it does dequeue and those it does not.

Table 10-9 lists sample messages by their number and Message ID.

Table 10-9 Sample Messages Used for One BPEL Process Use Case

Message Number	Message ID
Message 1	414D51206C6F63616C5F716D2020202023595B4D20002802
Message 2	414D51206C6F63616C5F716D2020202023595B4D20002D04
Message 3	414D51268A6C63618C5D516D2020202023595B4D20002A0D

Table 10-10 lists the results from using the message selector against the set of sample messages:

Table 10-10 Results from Using Message Selector against Sample Messages

Message Properties	Message 1	Message 2	Message 3
Priority	6	6	9
Message Type	8	2	8
Reply to Queue Name	queue12	queue49	queue12
Will be message be dequeued	Yes	No	No
Comments	The complete message selector return true and hence the message is dequeued		

Two BPEL Processes Receiving Messages from the Same MQ Queue. Both Processes Have Defined Mutually Exclusive Message Selectors

There are two BPEL processes, BPEL 1 and BPEL 2, both configured to receive MQ messages from the same MQ queue. You want to configure the inbound MQ Adapter in both these adapters in such a way that each BPEL process reliably receives messages meant for that process only.

For the first process, BPEL 1 you want to receive only those messages which satisfy the following criteria:

- The Correlation ID starts with 414D51206C6F63616C5F716D20202020
- Message Type is 2

For BPEL 2, you want wishes to receive only those messages which satisfy the following criteria:

- The Correlation ID starts with 414D5120514D5F616E6B7072616B615F
- Message Type is 2
- Priority > 5

Creating the Message Selectors for the Two BPEL Process Use Case

For the first process, BPEL 1 the MessageSelector string is:

```
jca.mq.MQMD.CorrelId LIKE
   '414D51206C6F63616C5F716D20202020%'
    AND jca.mq.MQMD.MsgType = 2
```

For the second process, BPEL 2 the MessageSelector string is:

```
jca.mq.MQMD.CorrelId LIKE
     '414D5120514D5F616E6B7072616B615F%' AND jca.mq.MQMD.MsgType
        = 2 AND jca.mq.MQMD.Priority > 5
```

The next step is to configure the inbound adapter. To do so for each of the BPEL process's inbound MQ adapter respectively, specify the Message Selector for each Adapter on the **Get Message from MQ** screen.

The MQ Series Adapter Wizard performs the following, based on what you enter in the Message Selector field.

For BPEL 1, the Wizard adds the following tag as a child of the <activation-spec> tag:

```
<property name="MessageSelector" value="
             jca.mq.MQMD.CorrelId LIKE
              '414D51206C6F63616C5F716D20202020%'
                 AND jca.mq.MQMD.MsgType = 2"/>
```

For BPEL 2, the Wizard adds the following tag as a child of the <activation-spec> tag:

```
<property name="MessageSelector" value="        jca.mq.MQMD.CorrelId LIKE
'414D5120514D5F616E6B7072616B615F %            ' AND jca.mq.MQMD.MsgType =
2               AND jca.mq.MQMD.Priority > 5"/>
```

Usage with Sample Messages for Two BPEL Process Use Case

When both the BPEL processes are deployed for listening on the same queue for messages simultaneously, the messages are selectively retrieved by each process.

Both processes receive only the message meant for them and if a message does not satisfy any of the BPEL process criteria then it remains on the queue. Table 10-11 lists the sample message used and Table 10-12 lists the results with certain sample messages:

Table 10-11 Sample Messages Used for Two BPEL Processes Use Case

Message Number	Message ID
Message 1	414D51206C6F63616C5F716D2020202023595B4D20004B06
Message 2	414D5120514D5F616E6B7072616B615F1E4B5A4D20094202

Table 10-11 (Cont.) Sample Messages Used for Two BPEL Processes Use Case

Message Number	Message ID
Message 3	414D5120514D325F616E6B7072616B61FD4A5A4D2006BE04

Table 10-12 Results from Using Message Selector against Sample Messages for Two BPEL Process Use Case

Message Properties	Message 1	Message 2	Message 3
Priority	1	5	8
Message Type	2	2	2
Where does the message go	BPEL1	BPEL2	Remains on Queue
Comments			The correlation id does not match the pattern-value of any of the BPEL processes.

Oracle MQ Series Adapter Use Cases

This section contains the following topics:

- Dequeue Enqueue
- Inbound Synchronous Request-Reply
- Inbound-Outbound Synchronous Request-Reply
- Asynchronous-Request-Reply
- Outbound Dequeue
- Configuring a Backout Queue
- CCDT Use Cases
- Reading Single or Multiple RFH2 Rules and Formatting Header Version 2 Headers

Dequeue Enqueue

This use case is the end-to-end demonstration of how MQ Adapter dequeues a message and enqueues the same message after transformation from the MQ Series queue. This section contains the following topics:

- Prerequisites
- Designing the SOA Composite
- Creating an Inbound Adapter Service
- Creating an Outbound Adapter Service

- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using Fusion Middleware Control

Prerequisites

To perform the dequeue enqueue use case, you must have the following files from the `artifacts.zip` file contained in the `Adapters-101MQAdapterDequeueEnqueue` sample:

- `artifacts/schemas/address-csv.xsd`
- `artifacts/schemas/address-fixedLength.xsd`
- `artifacts/input/data.txt`

You can obtain the `Adapters-101MQAdapterDequeueEnqueue` sample by accessing the Oracle SOA Sample Code site.

You must also create the following queues:

- `test_in`
- `test_out`

Designing the SOA Composite

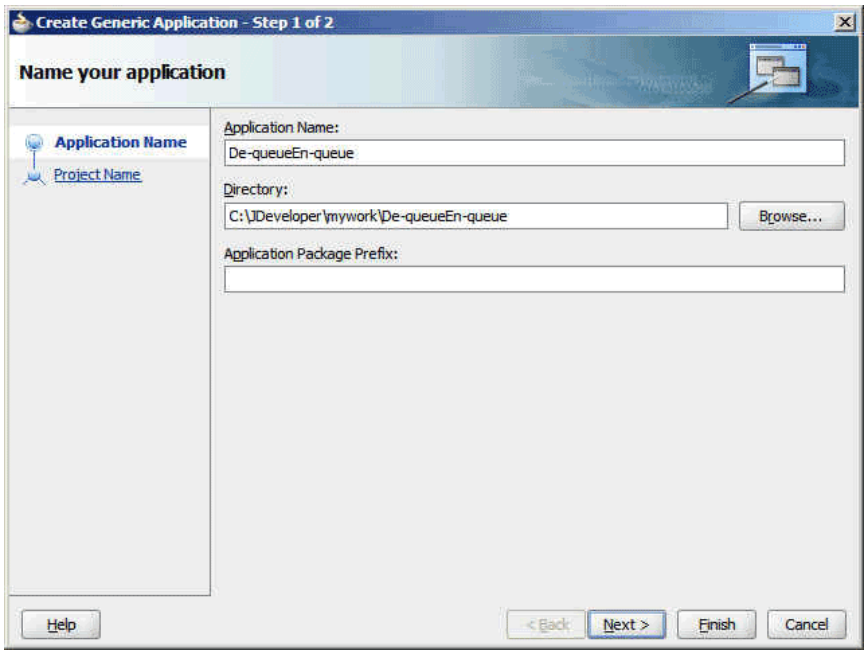
You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.

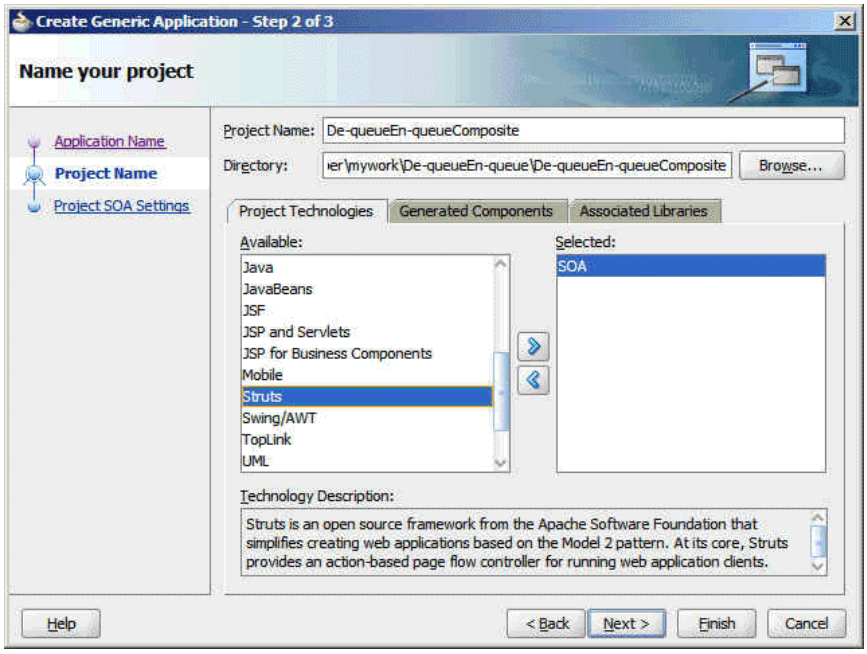
2. Expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.
4. In the **Name Your Application** screen, enter `De-queueEn-queue` in the **Application Name** field, as shown in Figure 10-43, and then click **Next**. The **Name Your Project** screen is displayed.

Figure 10-43 The Name Your Application Page



5. In the **Project Name** field, enter `De-queueEn-queueComposite` and from the **Available** list, select **SOA** and click the right-arrow button, as shown in Figure 10-44.

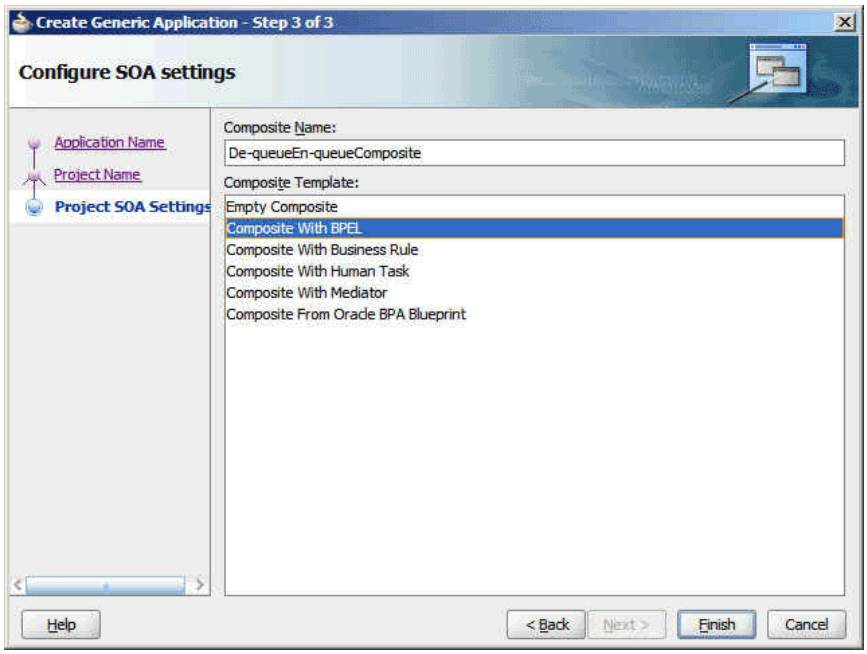
Figure 10-44 The Name Your Project Page



6. Click **Next**. The **Configure SOA Settings** screen is displayed.

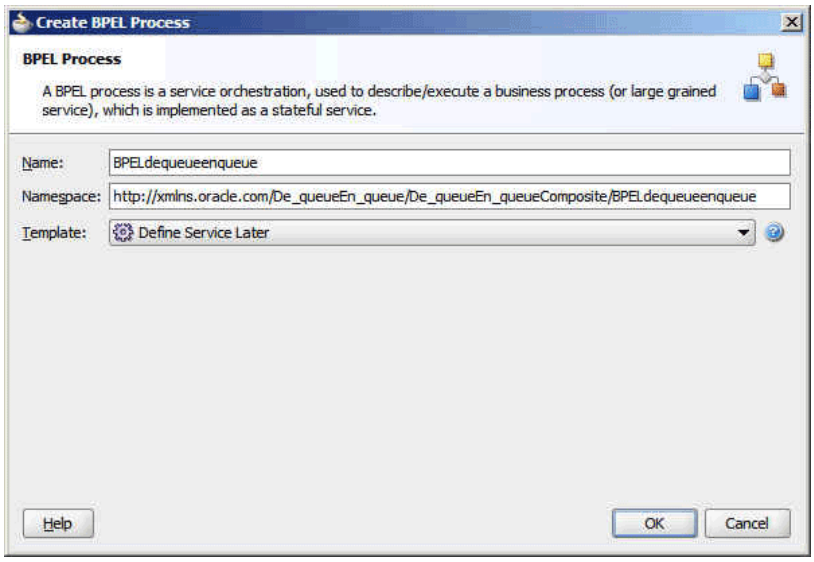
7. In the Composite Template list, select **Composite With BPEL**, as shown in Figure 10-45, and then click **Finish**. The Create BPEL Process dialog is displayed.

Figure 10-45 The Configure SOA Settings Page



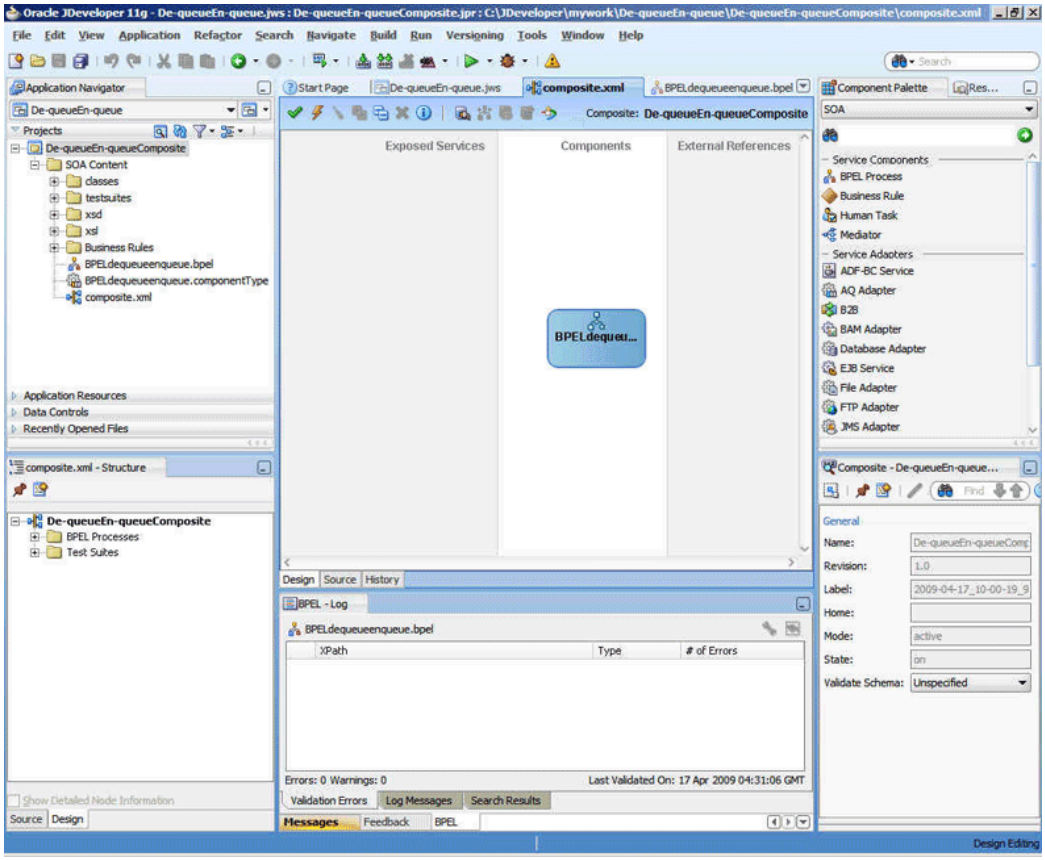
8. Enter `BPELdequeueenqueue` in the **Name** field, and select **Define Service Later** from the Template box, as shown in Figure 10-46.

Figure 10-46 The Create BPEL Process Dialog



9. Click **OK**. The De-queueEn-queue application and the De-queueEn-queue project appears in the design area, as shown in Figure 10-47.

Figure 10-47 The JDeveloper - Composite.xml



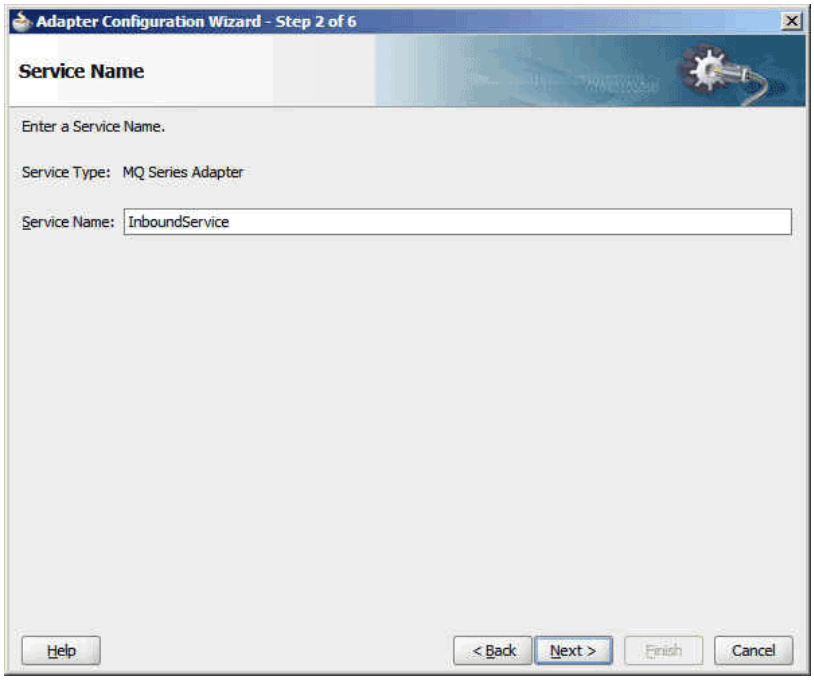
10. Copy the `address-csv.xsd` and `address-fixedLength.xsd` files to the xsd folder in your project (see Prerequisites for the location of these files).

Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message from a queue:

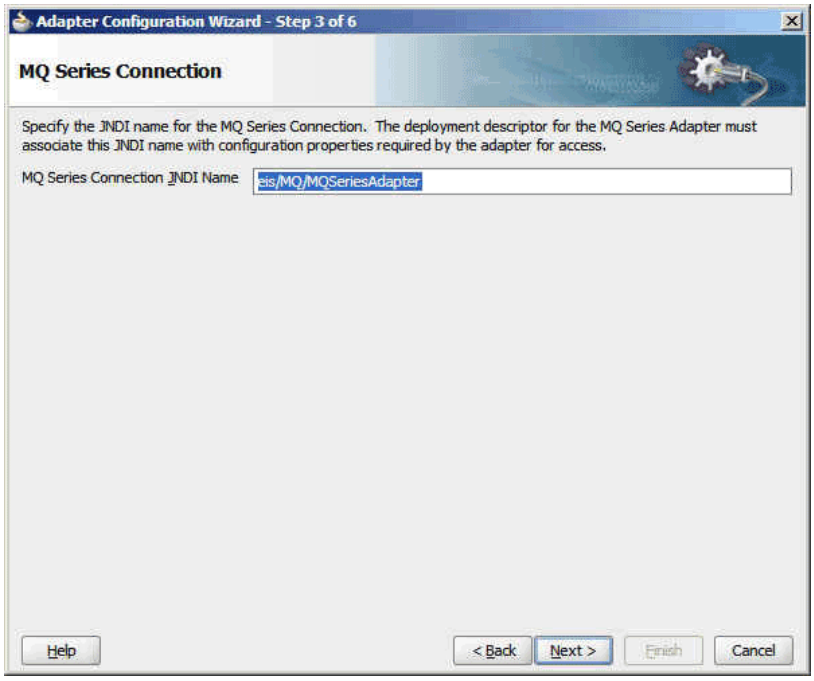
1. Drag and drop **MQ Adapter** from the Components window into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `InboundService` in the **Service Name** field, as shown in Figure 10-48, and click **OK**. The **MQ Series Connection** screen is displayed.

Figure 10-48 The Service Name Page



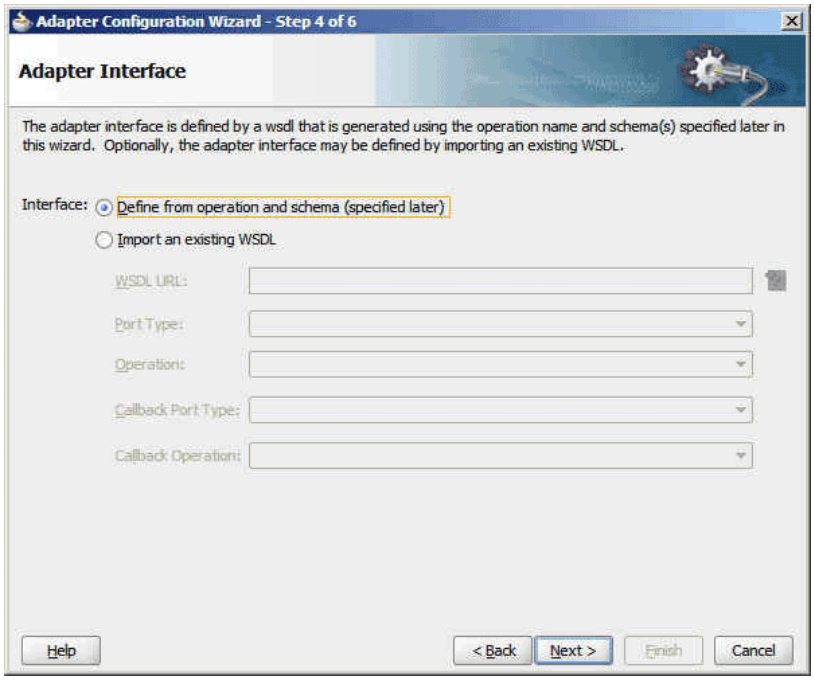
4. Accept the default JNDI name for the MQ Series connection, as shown in Figure 10-49, and click **Next**. The **Adapter Interface** page is displayed.

Figure 10-49 The MQ Series Connection Page



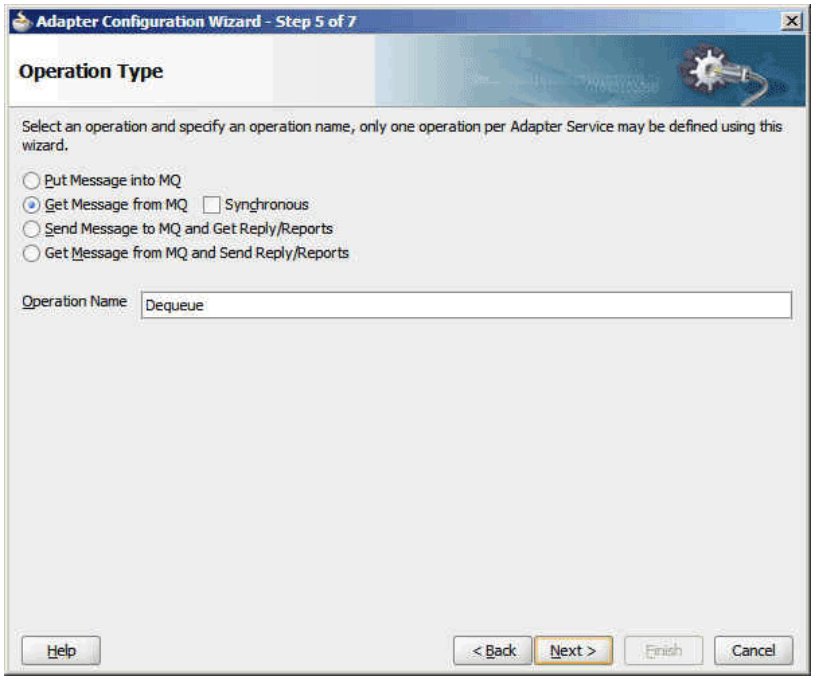
5. Select **Define from operation and schema (specified later)**, as shown in Figure 10-50, and click **Next**. The **Operation Type** page is displayed.

Figure 10-50 The Adapter Interface Page



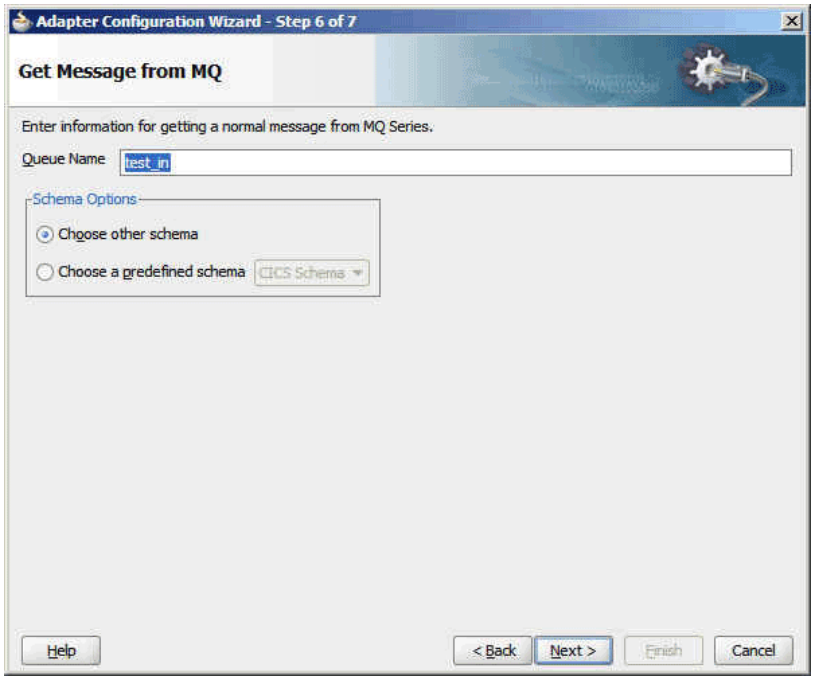
6. Select **Get Message from MQ**, as shown in Figure 10-51, and click **Next**. The **Get Message from MQ** page is displayed.

Figure 10-51 The Operation Type Page



7. Enter `test_in` in the **Queue Name** field, as shown in Figure 10-52, and click **Next**. The Messages page is displayed.

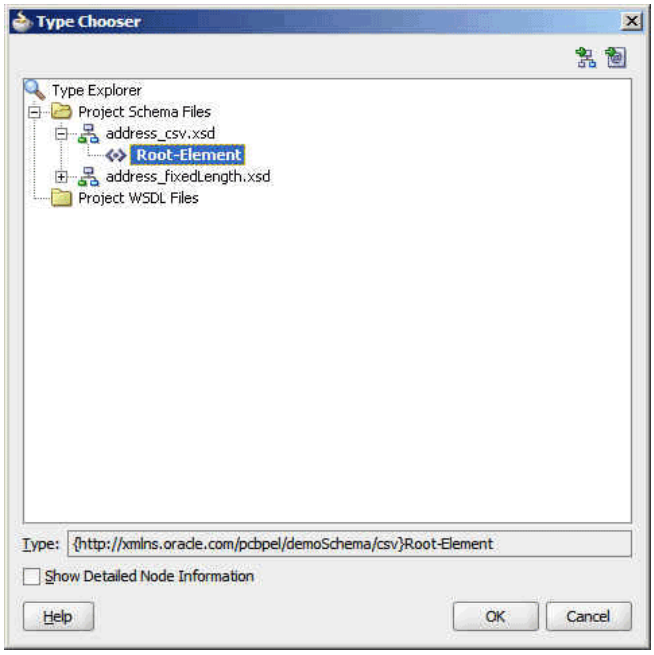
Figure 10-52 The Get Message From MQ Page



8. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.

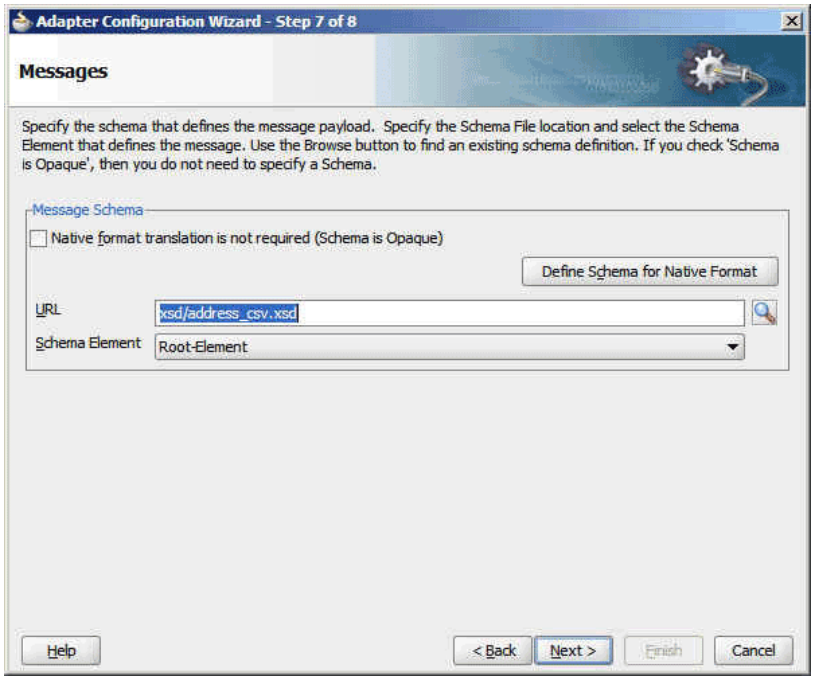
9. Select **Project Schema Files**, **address-csv.xsd**, and then **Root-Element**, as shown in Figure 10-53.

Figure 10-53 The Type Chooser Dialog



10. Click **OK**. The `address-csv.xsd` file appears in the URL field in the Messages page, as shown in Figure 10-54.

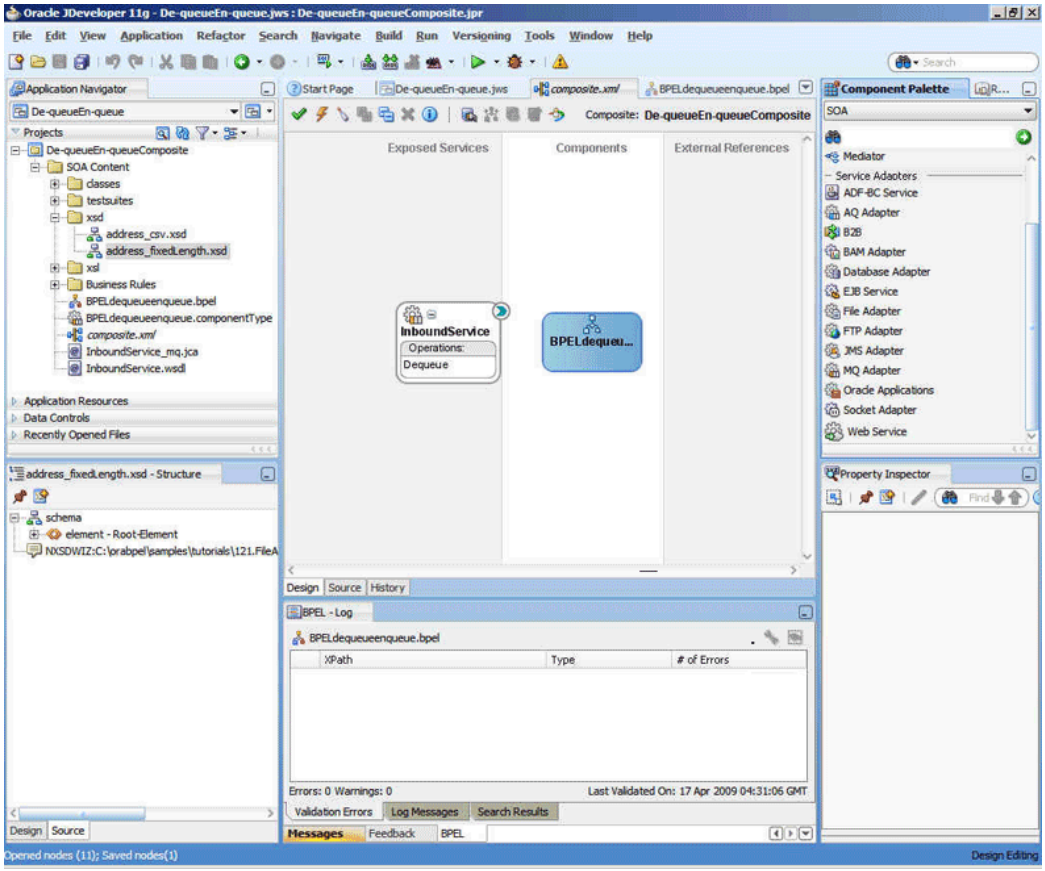
Figure 10-54 The Messages Page



11. Click **Next**. The **Finish** page is displayed.

12. Click **Finish**. You have now configured the inbound adapter service, and the composite.xml page is displayed, as shown in Figure 10-55.

Figure 10-55 The JDeveloper Page - Composite.xml Page

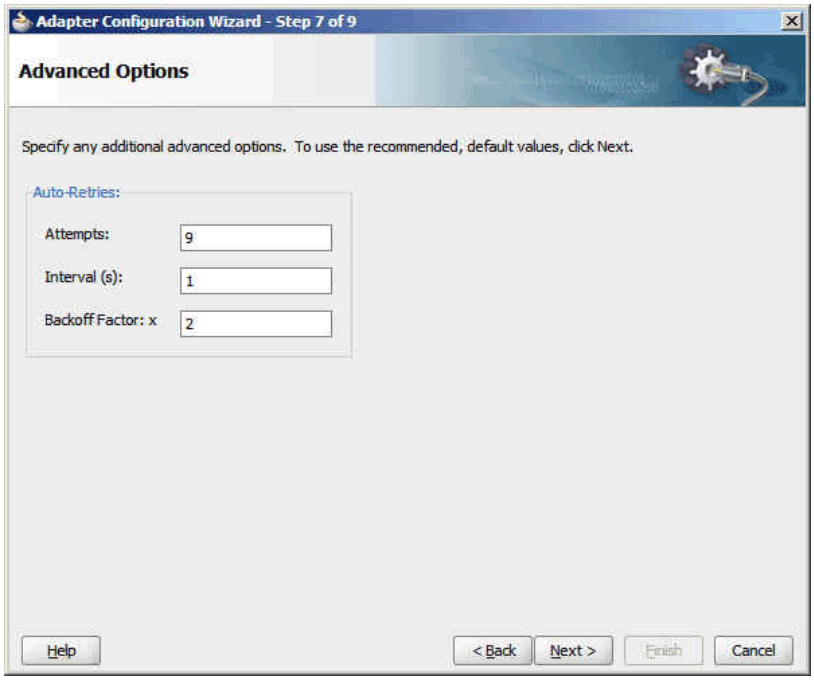


Creating an Outbound Adapter Service

Perform the following steps to create an adapter service that enqueues the messages.

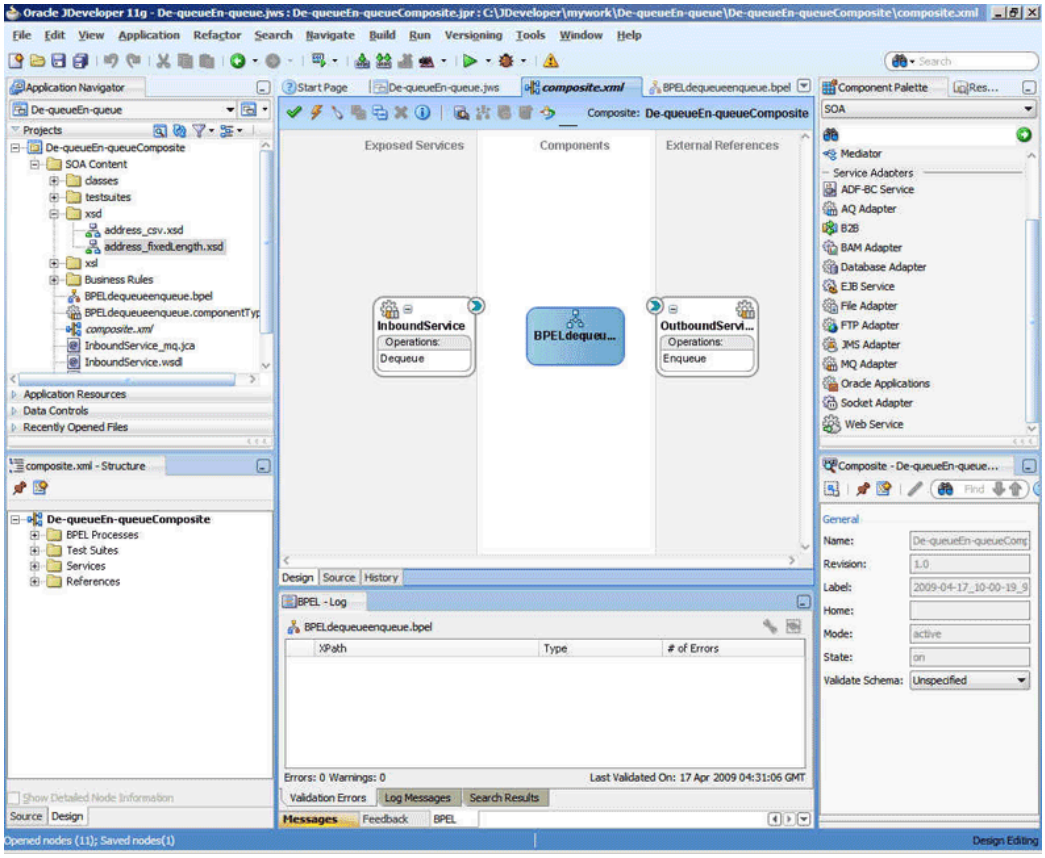
1. Drag and drop **MQ Adapter** from the Components window into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `OutboundService` in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, as shown in Figure 10-49, and click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, as shown in Figure 10-50, and click **Next**. The **Operation Type** page is displayed.
6. Select **Put Message into MQ**, and click **Next**. The Put Message into MQ page is displayed.
7. Enter `test_out` in the **Queue Name** field, and click **Next**. The Advanced Options page is displayed, as shown in Figure 10-56.

Figure 10-56 The Advanced Options Page



8. Accept the defaults and click **Next**. The Messages page is displayed.
9. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
10. Select **Project Schema Files**, **address-fixedLength.xsd**, and then **Root-Element**, and click **OK**. The address-fixedLength.xsd file appears in the URL field in the Messages page.
11. Click **Next**. The **Finish** page is displayed.
12. Click **Finish**. You have now configured the outbound adapter service, and the composite.xml page is displayed, as shown in Figure 10-57.

Figure 10-57 The JDeveloper Page - composite.xml Page



Wiring Services and Activities

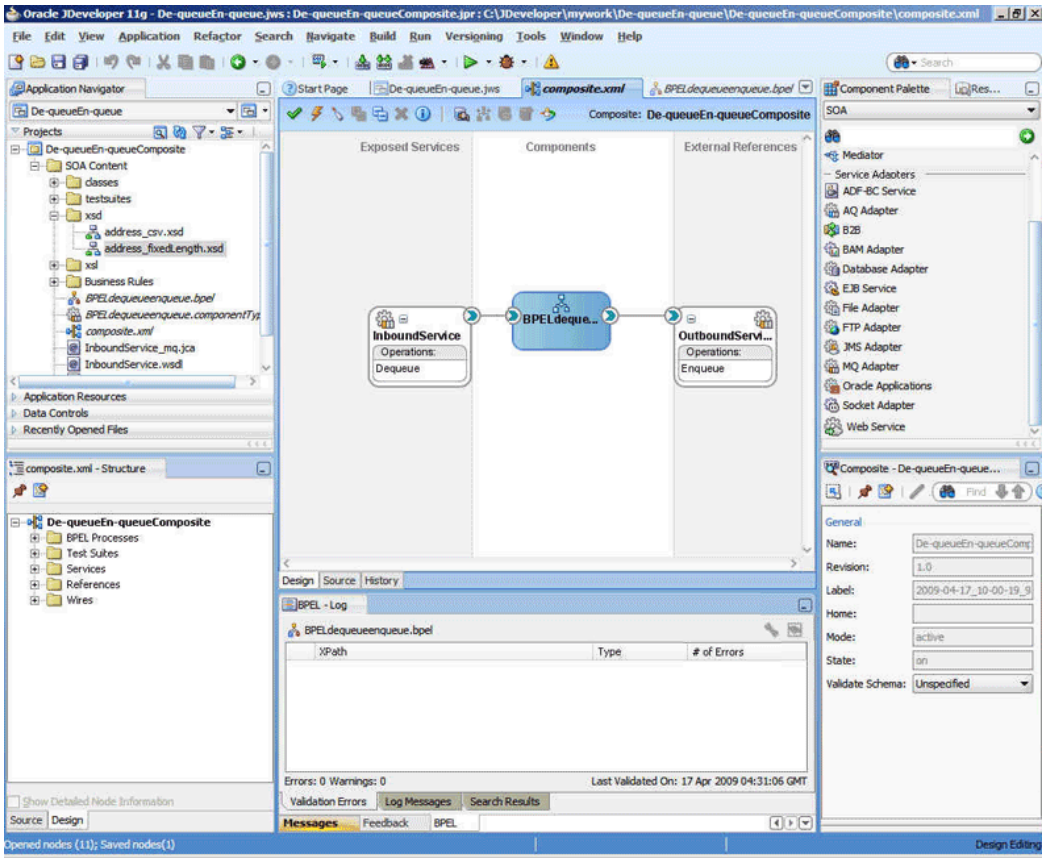
You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the **InboundService** in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.

2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in **OutboundService** in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 10-58.

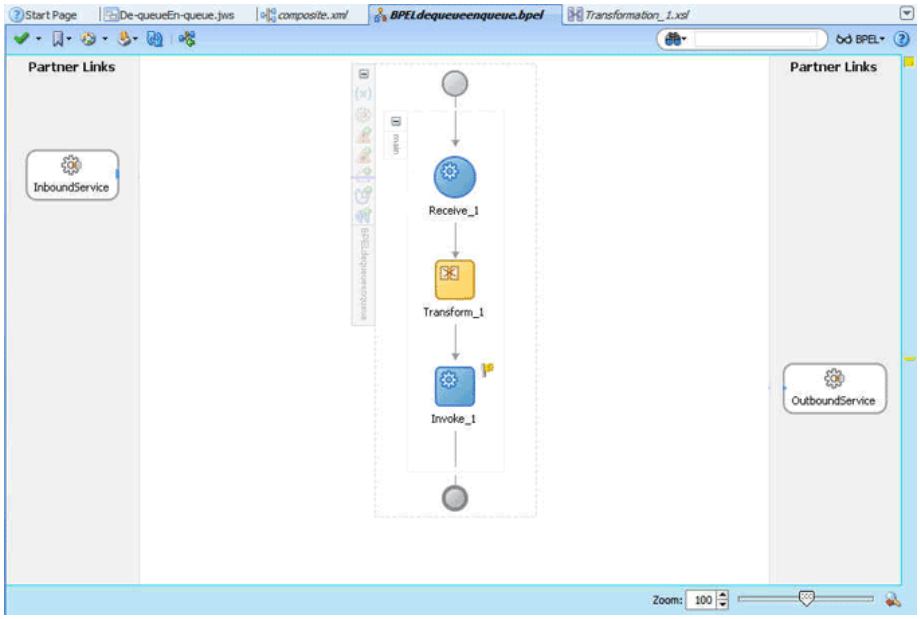
Figure 10-58 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.
4. Double-click **BPELdequeueenqueue**. The BPELdequeueenqueue.bpel page is displayed.
5. Drag and drop the **Receive**, **Transform**, and **Invoke** activities in the order mentioned from the Components window to the Components area.

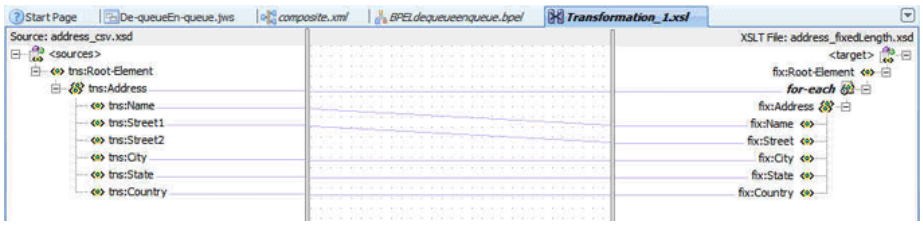
   The JDeveloper **BPELdequeueenqueue.bpel** page is displayed, as shown in Figure 10-59.

Figure 10-59 The BPELdequeueenqueue.bpel Page



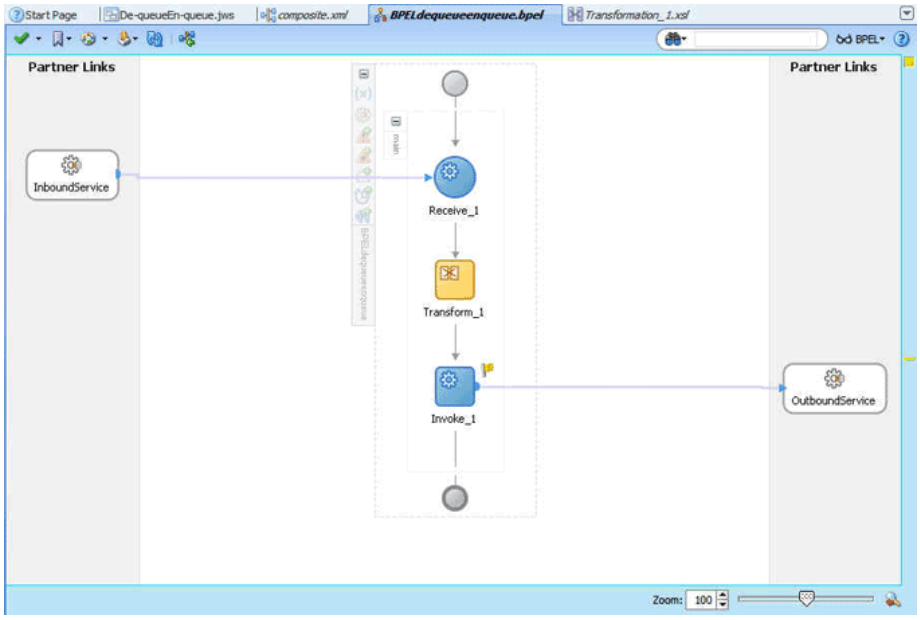
6. Drag and drop the **Receive** activity to InboundService. The Receive dialog is displayed.
7. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The Create Variable dialog is displayed.
8. Accept the defaults, and click **OK**.
9. Check the **Create Instance** box, and click **OK**.
10. Drag and drop the **Invoke** activity to OutboundService. The **Invoke** dialog is displayed.
11. Click the **Automatically Create Input Variable** icon that appears at the end of the Input Variable field.
12. Accept the defaults, and click **OK**. The **Invoke** dialog is displayed.
13. Click **OK**.
14. Double-click the **Transform** activity. The **Transform** dialog is displayed.
15. Click the **Create... (Alt+N)** icon. The **Source Variable** dialog is displayed.
16. Accept the defaults, and click **OK**.
17. Select the invoke variable as target, and click **OK**. The Transformation_xsl page is displayed.
18. Drag and drop **tns:Root-Element** in the Sources pane to the **fix:Root-Element** in the Target pane. The Auto Map Preferences dialog is displayed.
19. Click **OK**. The **Transformation_xsl** page is displayed, as shown in Figure 10-60.

Figure 10-60 The Transformation_xsl Page



The BPELdequeueenqueue.bpel page appears, as shown in Figure 10-61.

Figure 10-61 The BPELdequeueenqueue.bpel Page



Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Deploying Oracle JCA Adapter Applications from .

You must also create an application server connection. For more information about creating an application server connection, see Creating an Application Server Connection for Oracle JCA Adapters.

Monitoring Using Fusion Middleware Control

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Log in to *http://servername:portnumber/em* using your username/password. The Fusion Middleware Control page is displayed.
2. In the left pane, navigate to **SOA**, **soa-infra (soa_server1)**. A list of all the composites that are deployed appears.
3. Click **De-queueEn-queueComposite[1.0]**. The **De-queueEn-queueComposite[1.0]** page is displayed.

4. Copy the **data.txt** file and put it in the **test_in** queue.
5. Wait for some time and then refresh the Fusion Middleware Control Console. An instance appears on the console. This is the instance that was triggered because of the processing that occurred.
6. Click the **Instances** tab.
7. Click the instance associated with this deployment. The **Flow Trace** page is displayed.
8. Click the **BPELdequeueenqueue** component instance. The **Audit Trail** page is displayed.
9. Click the **Flow** tab to debug the instance. The BPEL process instance flow is displayed.
10. Click an activity to view the relevant payload details.

Inbound Synchronous Request-Reply

In this use case, the inbound Oracle MQ Series Adapter dequeues the request message from MQ Series inbound queue `test_in` and publishes it to the BPEL process. The Oracle MQ Series Adapter waits for the response from the BPEL process. When the Oracle MQ Series Adapter receives the response, it enqueues the response message to the MQ Series queue specified in the `replyToQueueName` queue of the request message. This use case consists of the following sections:

- Prerequisites
- Designing the SOA Composite
- Creating an Inbound Adapter Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using the Fusion Middleware Control Console

Prerequisites

This example assumes that you are familiar with basic BPEL constructs, such as activities and partner links, and the JDeveloper environment for creating and deploying BPEL Process.

The Oracle MQ Series Adapter must be configured as specified in Configuring the Oracle MQ Series Adapter and a queue `test_in` should be created.

To perform the inbound synchronous request-reply use case, you must have the following files from the `artifacts.zip` file contained in the `Adapters-101MQAdapterDequeueEnqueue` sample:

- `artifacts/schemas/address-csv.xsd`
- `artifacts/schemas/address-fixedLength.xsd`
- `artifacts/input/data.txt`

You can obtain the `Adapters-101MQAdapterDequeueEnqueue` sample by accessing the Oracle SOA Sample Code site.

You must also create the following queues:

- `test_in`
- `test_reply`

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.

2. Expand the **General** node, and select the **Applications** category.

3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.

4. In the **Name Your Application** screen, enter `SyncReqRes` in the **Application Name** field, and then click **Next**. The Name Your Project screen is displayed.

5. In the **Project Name** field, enter `Sync_ReqRes` and from the **Available** list, select SOA and click the right-arrow button.

6. Click **Next**. The Configure SOA Settings screen is displayed.

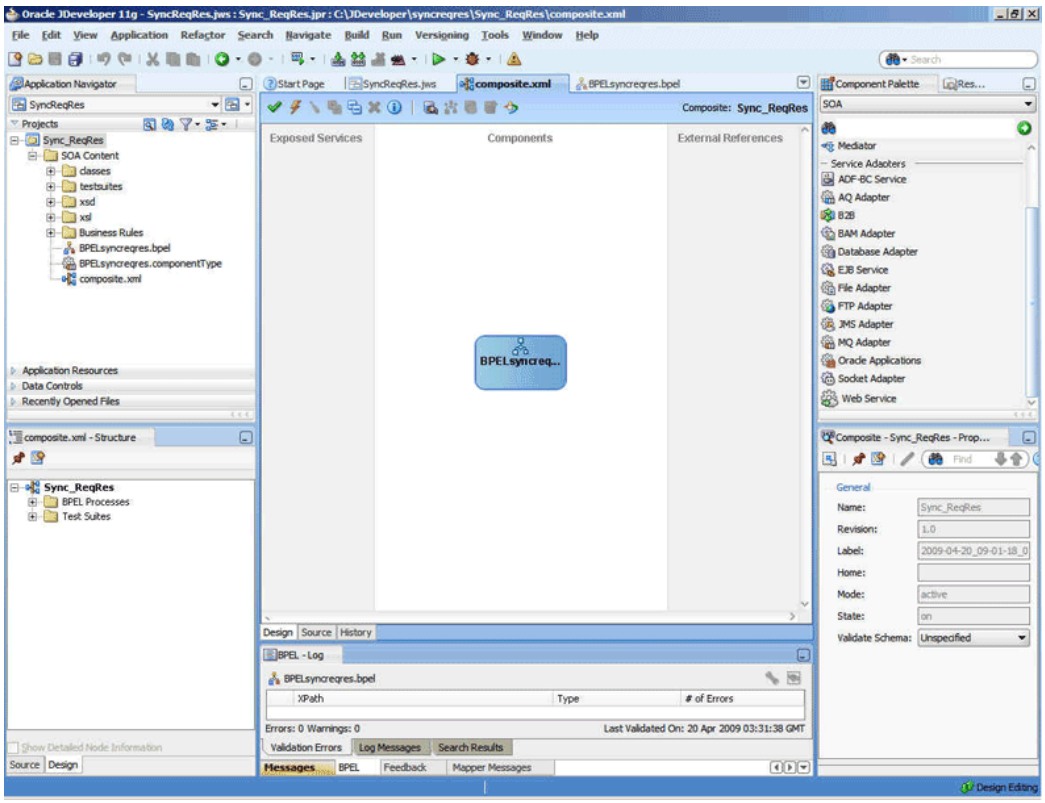
7. In the Composite Template list, select **Composite With BPEL** and then click **Finish**. The Create BPEL Process dialog is displayed.

   The Application Navigator of JDeveloper is updated with the new application and project and the Design tab contains, a blank palette.

8. Enter `BPELsyncreqres` in the **Name** field, select **Define Service Later** from the Template box.

9. Click **OK**. The SyncReqRes application and Sync_ReqRes project appears in the design area, as shown in Figure 10-62.

Figure 10-62 The JDeveloper - Composite.xml



Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message from a queue:

1. Drag and drop **MQ Adapter** from the Components window into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `inbound_reqres` in the **Service Name** field, and click **Next**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection JNDI name, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Get Message from MQ and Send Reply/Reports**, and select **Synchronous**, as shown in Figure 10-18, and click **Next**. The **Get Message from MQ and Send Reply/Reports page** is displayed.
7. Select **Normal** in the Message Type box, and enter `test_in` in the **Queue Name** field.
8. Click **Next**. The **Response** page is displayed.
9. Accept the defaults, and click **Next**. The **Messages** page is displayed.

10. Select **Project Schema Files**, **address-csv.xsd**, and then **Root-Element**, and click **OK**. The address-csv.xsd file appears in the URL field in the Messages page.

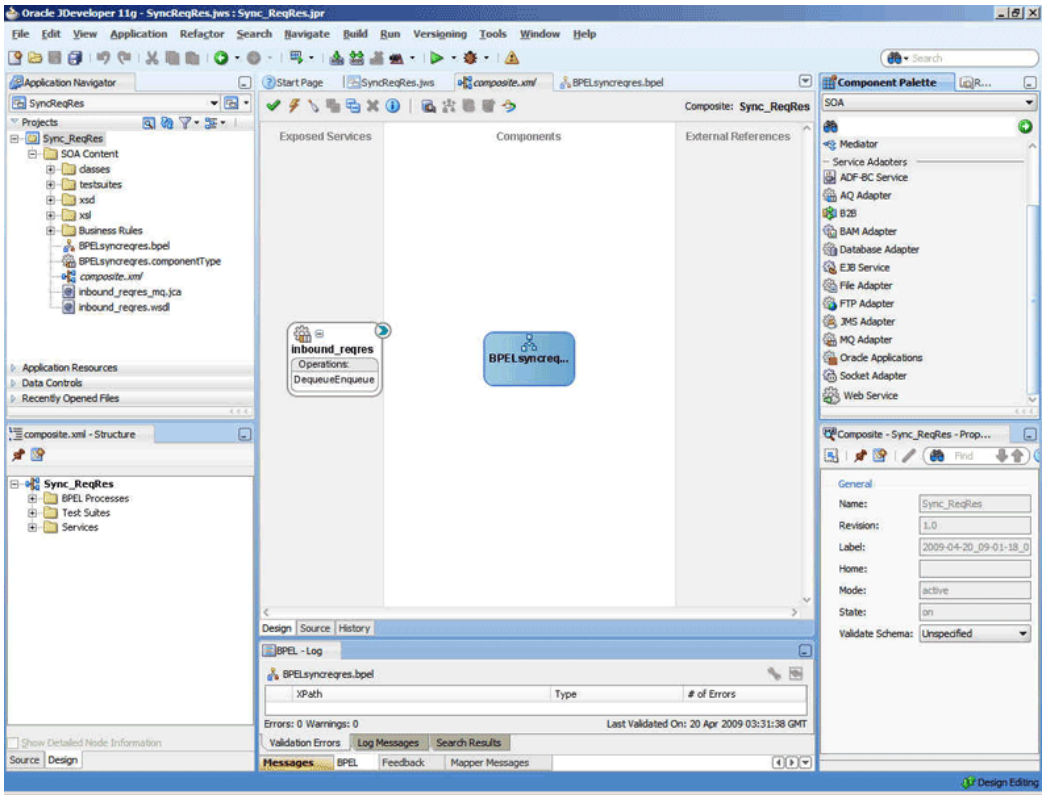
11. In the Send Message Schema group, click **Browse** at the end of the URL field. The **Type Chooser** dialog is displayed.

12. Select **Project Schema Files**, **address-fixedLength.xsd**, and then **Root-Element**, and click **OK**. The address-fixedLength.xsd file appears in the URL field in the Messages page.

13. Click **Next**. The **Finish** page is displayed.

14. Click **Finish**. You have now configured the inbound adapter service, and the **composite.xml** page is displayed, as shown in Figure 10-63.

Figure 10-63 The JDeveloper Page - Composite.xml Page



15. Click **File**, **Save All**.

Wiring Services and Activities

Perform the following steps to wire components:

1. Drag and drop the `inbound_reqres` adapter service to the `BPELsyncreqres` BPEL process.

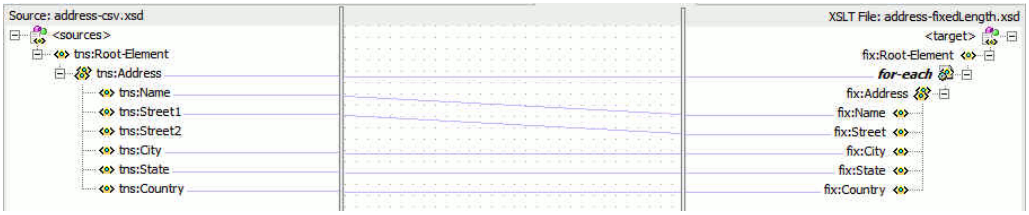
2. Double-click **BPELsyncreqres**. The **BPELsyncreqres.bpel** page is displayed.

3. Drag and drop the **Receive**, **Transform**, and **Reply** activities in the order mentioned from the Components window to the Components area.

4. Drag and drop the **Receive** activity to the `inbound_reqres` adapter service. The Receive dialog is displayed.

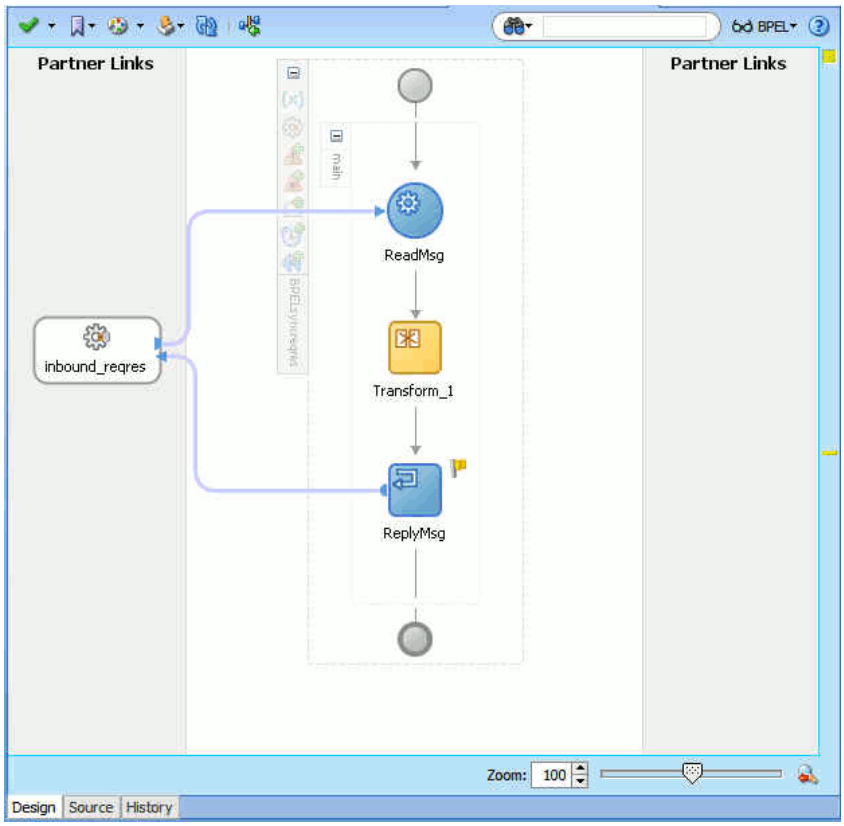
5. Enter `ReadMsg` in the **Name** field.
6. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The Create Variable dialog is displayed.
7. Accept the defaults, and click **OK**.
8. Check the **Create Instance** box in the Receive dialog, and click **OK**.
9. Drag and drop the **Reply** activity to the `inbound_reqres` adapter service. The Reply dialog is displayed.
10. Enter `ReplyMsg` in the **Name** field.
11. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The Create Variable dialog is displayed.
12. Accept the defaults, and click **OK**. The variable appears in the Reply dialog.
13. Click **OK**.
14. Double-click the **Transform** activity. The Transform dialog is displayed.
15. Click the plus icon. The Source Variable dialog is displayed.
16. From the Source Variable list, select **ReadMsg_DequeueEnqueue_InputVariable**, and click **OK**.
17. From the Target Variable list, select **ReplyMsg_DequeueEnqueue_OutputVariable**.
18. Click the **Create Mappings** icon. The Transformation.xsl page is displayed, as shown in Figure 10-64.

Figure 10-64 The Transformation.xsl Page



19. Drag the t**ns:Root-Element** from <sources> panel to the **fix:Root-Element** of the <target> panel. The **Auto Map Preferences** dialog is displayed.
20. Click **OK**. The JDeveloper **BPELsyncreqres.bpel** page is displayed, as shown in Figure 10-65.

Figure 10-65 The BPELsyncreqres.bpel Page



Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

To deploy the application profile using JDeveloper, see Deploying Oracle JCA Adapter Applications from .

You must also create an application server connection. For more information about creating an application server connection, see Creating an Application Server Connection for Oracle JCA Adapters.

Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Log in to *http://servername:portnumber/em* using your username/password. The Fusion Middleware Control page is displayed.
2. In the left pane, navigate to **SOA**, **soa-infra (soa_server1)**. A list of all the composites that are deployed appears.
3. Click **Sync_ReqRes[1.0]**. The Sync_ReqRes[1.0] page is displayed.
4. Create an MQ message with the contents of the `data.txt` file and set **replyToQueueName** to `test_reply`. Put this message in the `test_in` queue.

5. Wait for some time and then refresh the Fusion Middleware Control Console. An instance appears on the console. This is the instance that was triggered because of the processing that occurred.
6. Click the **Instances** tab.
7. Click the instance associated with this deployment. The Flow Trace page is displayed.
8. Click the **BPELsyncreqres** component instance. The Audit Trail page is displayed.
9. Click the **Flow** tab to debug the instance. The BPEL process instance flow is displayed.
10. Click an activity to view the relevant payload details.

Inbound-Outbound Synchronous Request-Reply

This use case is the end-to-end demonstration of the Synchronous Solicit Request-Reply scenario for MQ Adapter. In this use case, the composite dequeues the message from an inbound queue. Then, it enqueues a reply message to the **replyToQueue** queue as specified in the inbound message. This section contains the following topics:

- Prerequisites
- Designing the SOA Composite
- Creating an Inbound Adapter Service
- Creating an Outbound Adapter Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using the Fusion Middleware Control Console

Prerequisites

To perform the inbound synchronous request-reply use case, you require the following files from the `artifacts.zip` file contained in the `Adapters-101MQAdapterDequeueEnqueue` sample:

- `artifacts/schemas/address-csv.xsd`
- `artifacts/schemas/address-fixedLength.xsd`

You must also create queues named:

- `test_in`
- `test1`
- `ReplyQ`
- `test_reply`

You can obtain the `Adapters-101MQAdapterDequeueEnqueue` sample by accessing the Oracle SOA Sample Code site.

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.

2. Expand the **General** node, and select the **Applications** category.

3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.

4. In the **Name Your Application** screen, enter `Sync-Req-Rep` in the **Application Name** field, and then click **Next**. The Name Your Project screen is displayed.

5. In the **Project Name** field, enter `Sync-Req-RepComposite` and from the **Available** list, select SOA and click the right-arrow button.

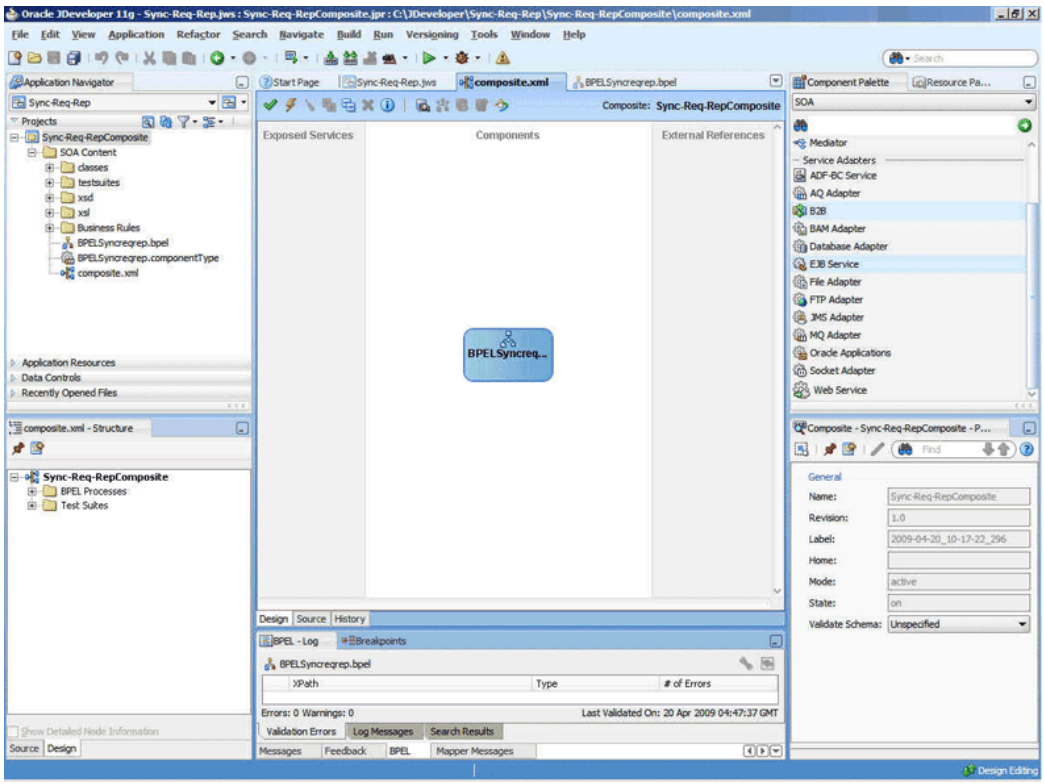
6. Click **Next**. The Configure SOA Settings screen is displayed.

7. In the Composite Template list, select **Composite With BPEL** and then click **Finish**. The Create BPEL Process dialog is displayed.

8. Enter `BPELSyncreqrep` in the **Name** field, select **Define Service Later** from the Template box.

9. Click **OK**. The Sync-Req-Rep application and Sync-Req-RepComposite project appears in the design area, as shown in Figure 10-66.

Figure 10-66 The JDeveloper - Composite.xml

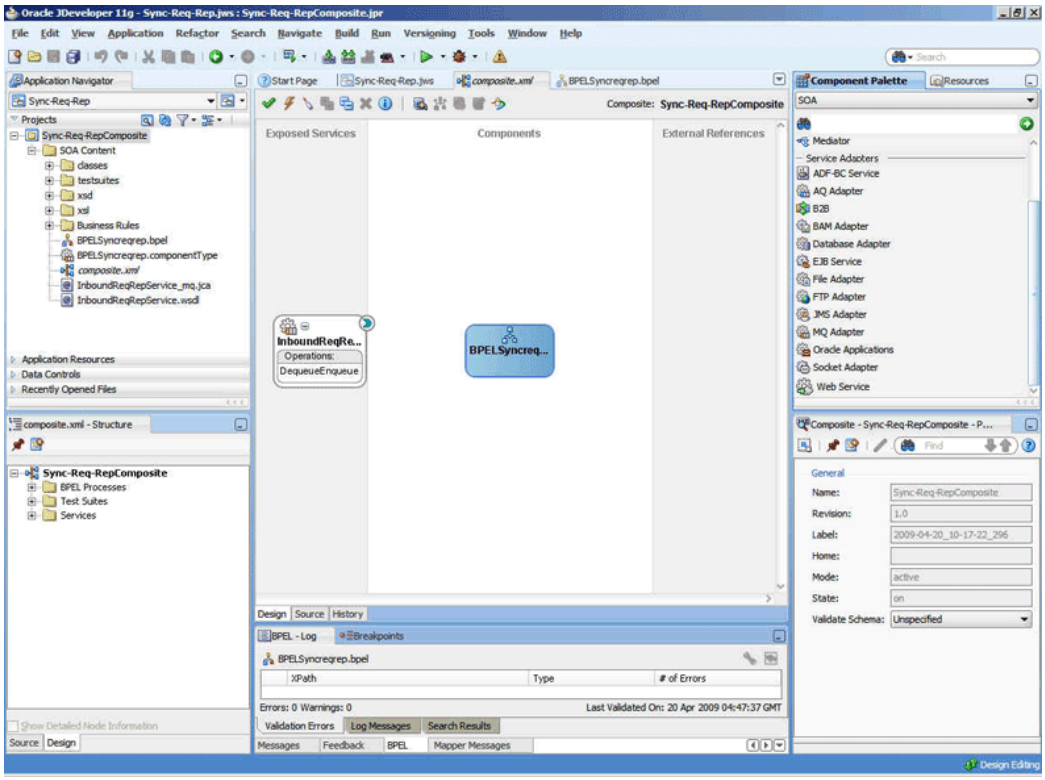


Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message from a queue:

1. Drag and drop **MQ Adapter** from the Components window into the Exposed Services swim lane. The Adapter Configuration Wizard **Welcome** page is displayed.
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `InboundReqRepService` in the **Service Name** field, and click **Next**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Get Message from MQ and Send Reply/Reports** and **Synchronous** in the **Operation Name** box, as shown in Figure 10-18, and click **Next**. The Get Message from MQ and Send Reply/Reports page is displayed.
7. Select **Normal** in the Message Type list and enter `test_in` in the **Queue Name** field and select **Choose Other Schema** in the **Schema Options** box, and click **Next**. The Response page is displayed.
8. Accept the defaults and click **Next**. The Message page is displayed.
9. Click **Browse** in the **Get Message Schema** box that appears at the end of the URL field. The **Type Chooser** dialog is displayed.
10. Select **Project Schema Files**, **address-csv.xsd**, and **Root-Element**, and then click **OK**. The Message page is populated with the address-csv.xsd file in the Get Message Schema box.
11. Click **Browse** in the **Send Message Schema** box that appears at the end of the URL field. The Type Chooser dialog is displayed.
12. Select **Project Schema Files**, **address-fixedLength.xsd**, and **Root-Element**, and then click **OK**. The **Message** page is populated with the address-fixedLength.xsd file in the **Send Message Schema** box.
13. Click **Next**. The Finish page is displayed.
14. Click **Finish**. You have configured the InboundReqRepService adapter service, and the composite.xml page is displayed, as shown in Figure 10-67.

Figure 10-67 The JDeveloper Page - Composite.xml Page



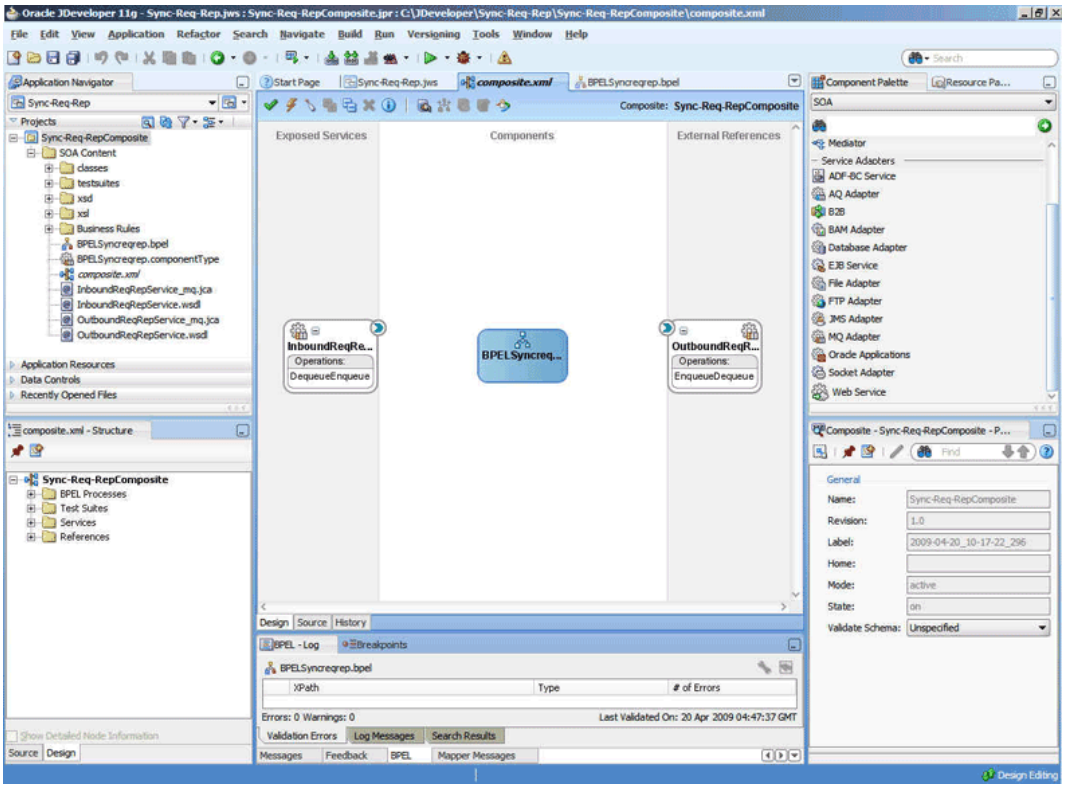
Creating an Outbound Adapter Service

Perform the following steps to create an adapter service that enqueues the request messages and dequeue the corresponding response messages (report) from a queue:

1. Drag and drop **MQ Adapter** from the Components window into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `OutboundReqRepService` in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.
4. Accept the defaults and click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Send Message to MQ and Get Reply/Reports**, select **Synchronous** in the Operation Name box, and click **Next**. The Send Message to MQ and Get Reply/Reports page is displayed.
7. Enter `test1` in the **Queue Name** field and click **Next**. The **Response** page is displayed.
8. Enter the name of the queue in the **Reply To Queue Name** field such as `ReplyQ`, select the **Response Wait Interval** option and enter a value, and select the **Empty Response Message Allowed** option.

9. Click **Next**. The **Advanced Options** page is displayed.
10. Accept the default values and click **Next**. The **Messages** page is displayed.
11. Click **Browse** in the Get Message Schema box that appears at the end of the URL field. The Type Chooser dialog is displayed.
12. Select **Project Schema Files**, **address-csv.xsd**, and **Root-Element**, and then click **OK**. The Message page is populated with address-csv.xsd file in the Get Message Schema box.
13. Click **Browse** in the Send Message Schema box that appears at the end of the URL field. The Type Chooser dialog is displayed.
14. Select **Project Schema Files**, **address-fixedLength.xsd**, and **Root-Element**, and then click **OK**. The **Message** page is populated with address-fixedLength.xsd file in the **Send Message Schema** box.
15. Click **Next**. The Finish page is displayed.
16. Click **Finish**. You have configured the OutboundReqRepService service, and the composite.xml page is displayed, as shown in Figure 10-68.

Figure 10-68 The JDeveloper Page - Composite.xml Page



Wiring Services and Activities

You have to assemble or wire the three components that you have created: InboundReqRepService, BPELSyncreqrep, and OutboundReqRepService. Perform the following steps to wire the components:

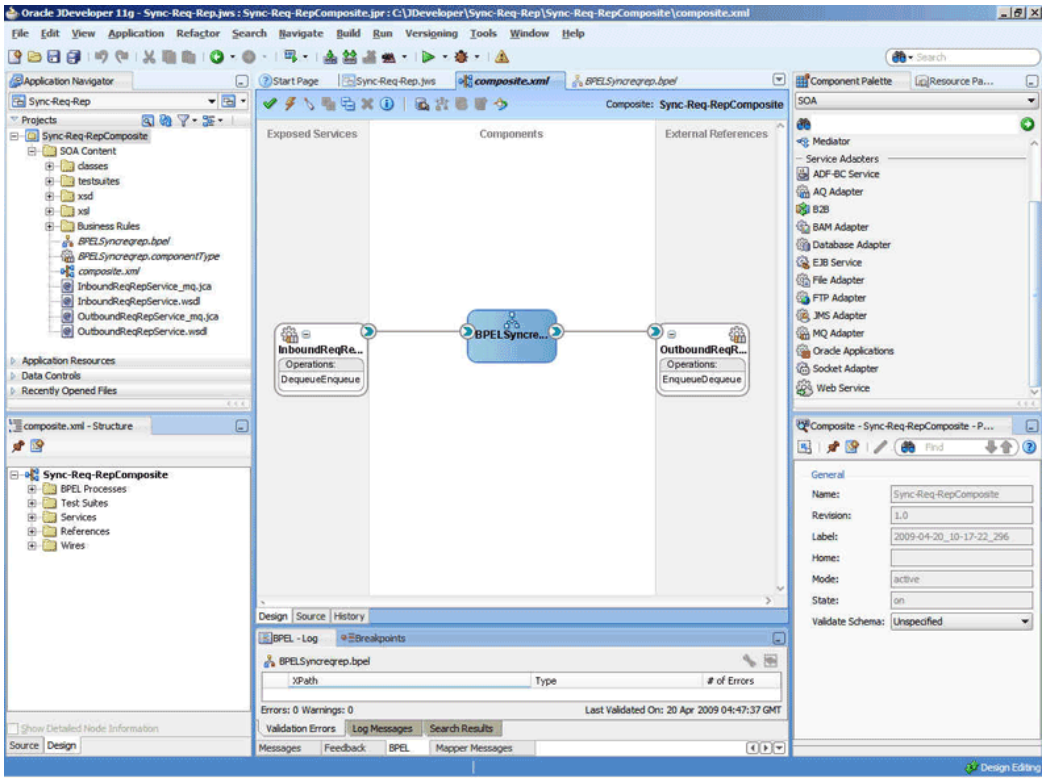
1. Drag the small triangle in the **InboundReqRepService** service in the Exposed Services area to the drop zone that appears as a green triangle in **BPELSyncreqrep** in the Components area.

2. Drag the small triangle in **BPELSyncreqrep** in the Components area to the drop zone that appears as a green triangle in **OutboundReqRepService** in the External References area.

3. Similarly, drag the small triangle in **BPELSyncreqrep** in the Components area to the drop zone in **OutboundReqRepService** in the External References area.

   The JDeveloper **composite.xml** page appears, as shown in Figure 10-69.

Figure 10-69 The JDeveloper - Composite.xml

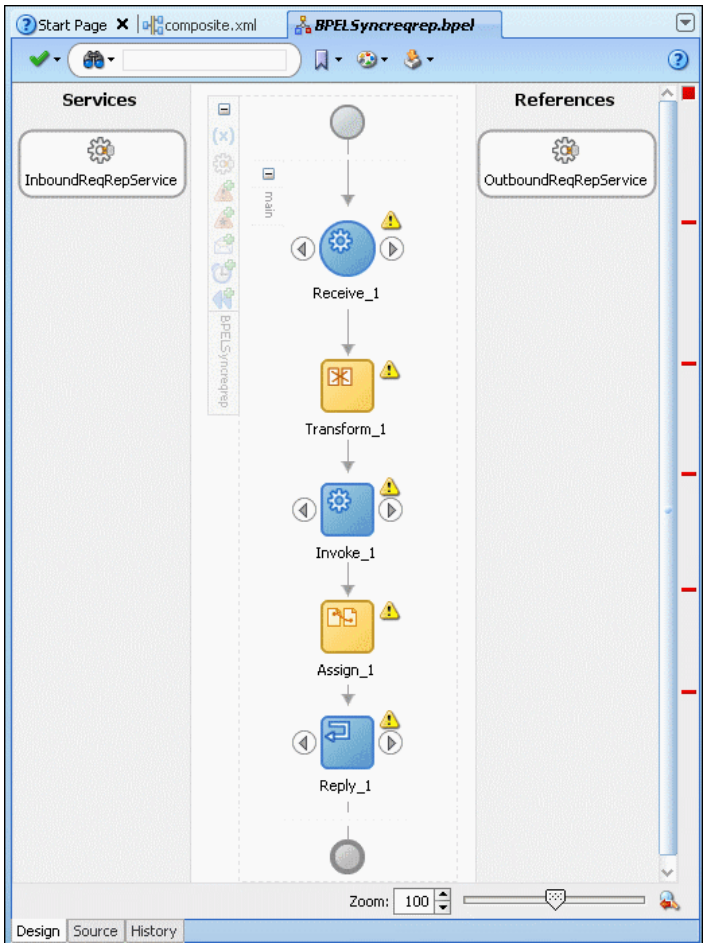


4. Click **File**, **Save All**.

5. Double-click **BPELSyncreqrep**. The **BPELSyncreqrep.bpel** page is displayed.

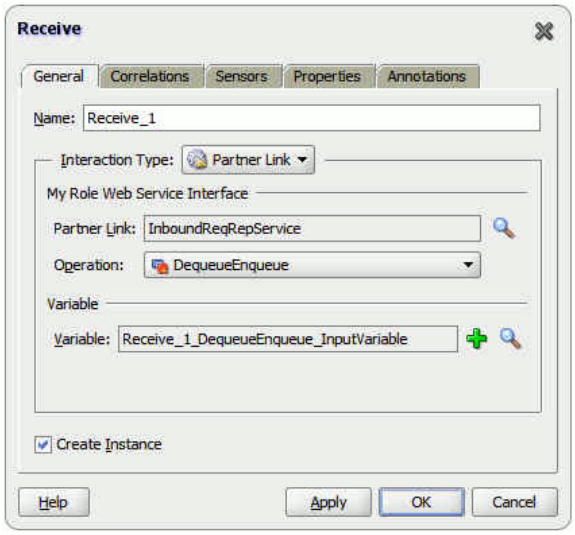
6. Drag and drop the **Receive**, **Transform**, **Invoke**, **Assign**, **Reply** activities in the order mentioned from the Components window to the Components area. The JDeveloper **BPELSyncreqrep.bpel** page is displayed, as shown in Figure 10-70.

Figure 10-70 The BPELSyncreqrep.bpel Page



7. Drag and drop the **Receive** activity to `InboundReqRepService`. The **Receive** dialog is displayed.
8. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The **Create Variable** dialog is displayed.
9. Accept the defaults, and click **OK**.
10. Check the **Create Instance** box, as shown in Figure 10-71, and click **OK**.

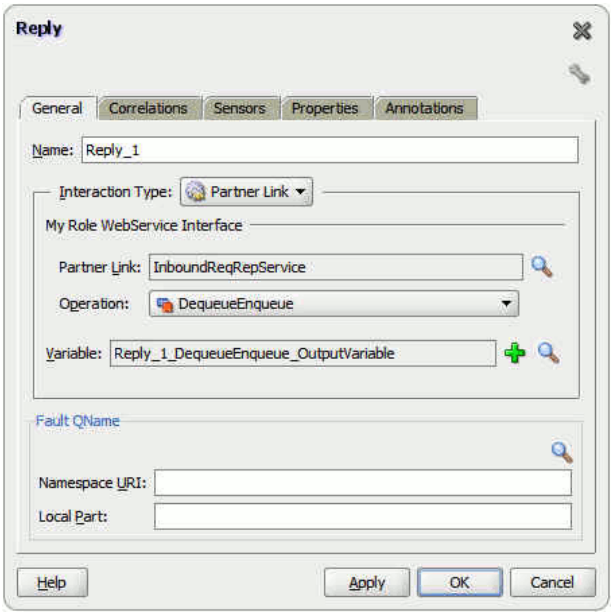
Figure 10-71 The Receive Dialog



11. Drag and drop the Reply activity to InboundReqRepService. The Reply dialog is displayed.

12. Click the **Auto Create Variable** icon to create the variable, and then click **OK**. The Reply dialog is displayed, as shown in Figure 10-72.

Figure 10-72 The Reply Dialog



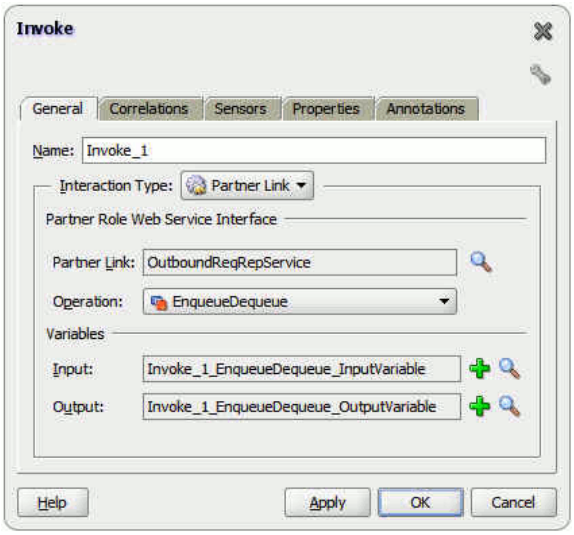
13. Drag and drop the **Invoke** activity to the OutboundReqRepService service. The Invoke dialog is displayed.

14. Click the **Automatically Create Input Variable** icon that appears at the end of the Input Variable field. The Create Variable dialog is displayed.

15. Click **OK**.

16. Similarly, create the output variable. Accept the defaults, and click **OK**. The Invoke dialog is displayed, as shown in Figure 10-73.

Figure 10-73 The Invoke Dialog

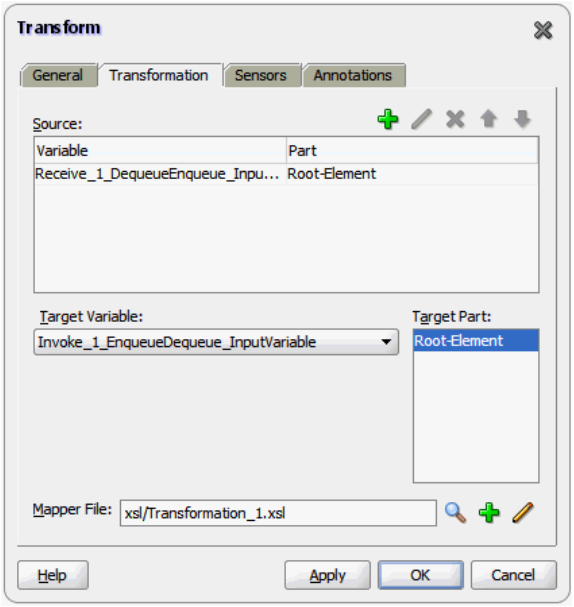


17. Click **OK**.

18. Double-click the **Transform** activity. The Transform dialog is displayed.

19. Click the plus icon, and select **Receive_1_DequeueEnqueue_InputVariable** as the source variable. Then, select **Invoke_1_EnqueueDequeue_InputVariable** for the target variable, as shown in Figure 10-74.

Figure 10-74 The Transform Dialog

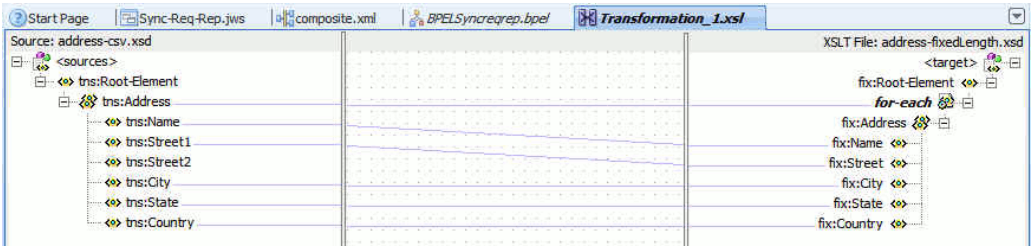


20. Click **Create Mapping**. The Transformation_1.xsl page is displayed.

21. Drag and drop the **tns:Root-Element** from the from <sources> panel to **fix:Root-Element** in the <target> panel. The Auto Map Preferences dialog is displayed.

22. Click **OK**. The mappings appear in the Transformation.xsl page, as shown in Figure 10-75.

Figure 10-75 The Transformation.xsl Page with Mappings



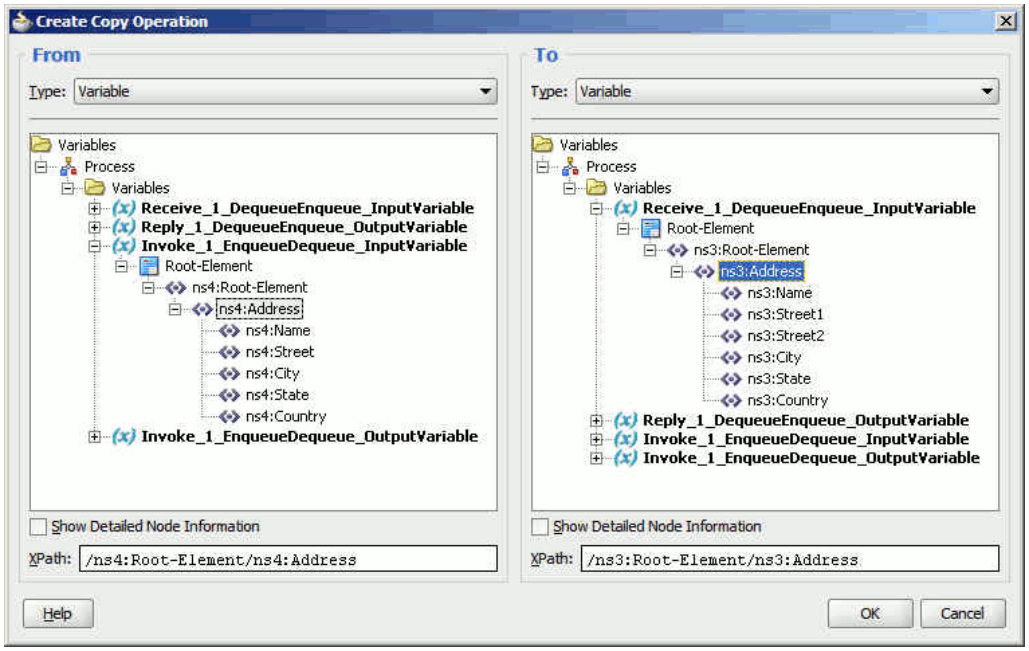
23. Click the **BPELSyncreqrep.bpel** tab.

24. Double-click the **Assign** activity. The Assign dialog is displayed.

25. Click the plus icon, and select **Copy Operation**. The Create Copy Operation dialog is displayed.

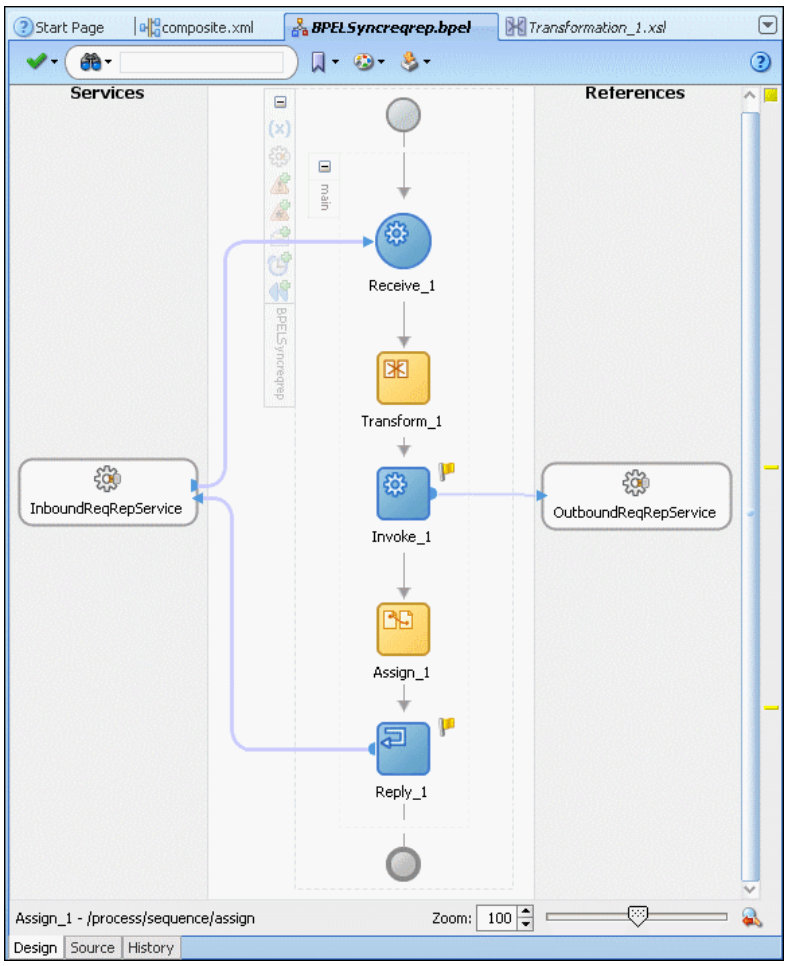
26. Select the variables, as shown in Figure 10-76, and click **OK**.

Figure 10-76 The Create Copy Operation Dialog



27. Click **OK** in the Assign dialog. The JDeveloper BPELSyncreqrep.bpel page is displayed, as shown in Figure 10-77.

Figure 10-77 The BPELSyncreqrep.bpel Page



Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Deploying Oracle JCA Adapter Applications from .

You must also create an application server connection. For more information about creating an application server connection, see Creating an Application Server Connection for Oracle JCA Adapters.

Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Log in to *http://servername:portnumber/em* using your username/password. The Oracle Fusion Middleware Control page is displayed.
2. In the left pane, navigate to **SOA**, **soa-infra (soa_server1)**. A list of all the composites that are deployed appears.
3. Click **Sync-Req-RepComposite[1.0]**. The Sync-Req-RepComposite[1.0] page is displayed.

4. Create an MQ message with the contents of the `data.txt` file and set **replyToQueueName** to `test_reply`. Put this message in the `test_in` queue.
5. Wait for some time and then refresh the Fusion Middleware Control Console. An instance appears on the console. This is the instance that was triggered because of the processing that occurred.
6. Click the **Instances** tab.
7. Click the instance associated with this deployment. The Flow Trace page is displayed.
8. Click the **BPELSyncreqrep** component instance. The Audit Trail page is displayed.
9. Click the **Flow** tab to debug the instance. The BPEL process instance flow is displayed.
10. Click an activity to view the relevant payload details.

Asynchronous-Request-Reply

This use case is the end-to-end demonstration of the Asynchronous-Request-Reply scenario. In this use case, first, the composite dequeues the message from an inbound queue. Then, it enqueues a request message and dequeues the reply message. Finally, the composite enqueues the reply message to the other queue. This section contains the following topics:

- Prerequisites
- Designing the SOA Composite
- Creating an Inbound Adapter Service
- Creating an Asynchronous Outbound Request Reply Adapter Service Outbound
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using the Fusion Middleware Control Console

Prerequisites

The Oracle MQ Series Adapter must be configured as specified in Configuring the Oracle MQ Series Adapter and create the following queues: test_in, test_out, and test_demo queues.

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.
2. Expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.

4. In the **Name Your Application** screen, enter `AsynchronousRequestReply` in the **Application Name** field, and then click **Next**. The Name Your Project screen is displayed.

5. In the **Project Name** field, enter `Async-Req-RepComposite` and from the **Available** list, select SOA and click the right-arrow button.

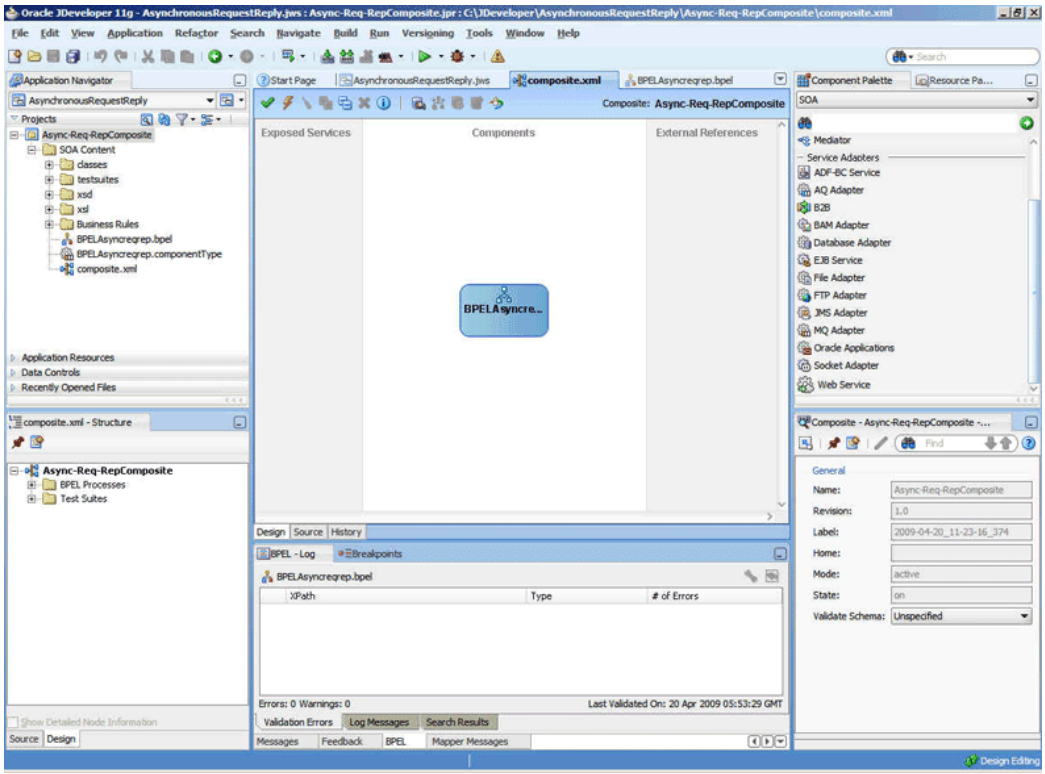
6. Click **Next**. The Configure SOA Settings screen is displayed.

7. In the Composite Template list, select **Composite With BPEL** and then click **Finish**. The Create BPEL Process dialog is displayed.

8. Enter `BPELAsyncreqrep` in the **Name** field, select **Define Service Later** from the Template box.

9. Click **OK**. The AsynchronousRequestReply application and the Async-Req-RepComposite project appear in the design area, as shown in Figure 10-78.

Figure 10-78 The JDeveloper - Composite.xml



Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message from a queue:

1. Drag and drop **MQ Adapter** from the Components window into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.

2. Click **Next**. The Service Name page is displayed.

3. Enter `InboundService` in the **Service Name** field, and click **Next**. The MQ Series Connection page is displayed.

4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.

5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.

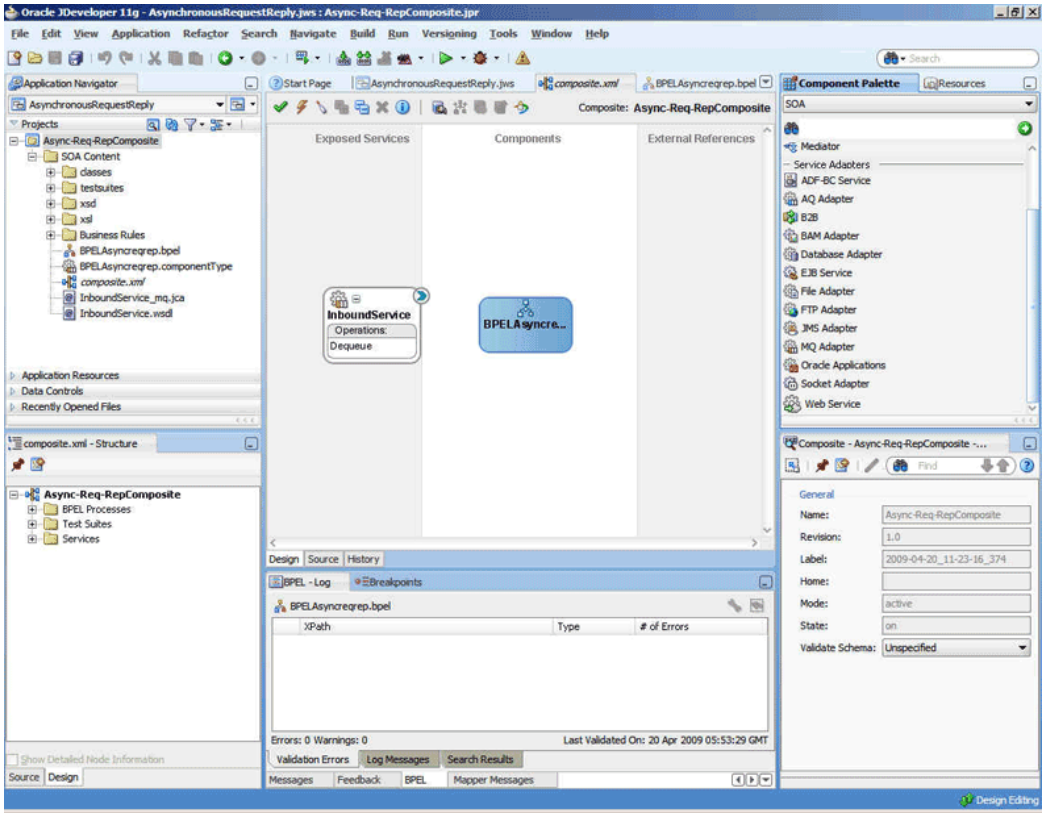
6. Select **Get Message from MQ** and click **Next**. The Get Message from MQ page is displayed.

7. Enter `test_in` in the **Queue Name** field and click **Next**. The Messages page is displayed.

8. Select **Native Format Translation is not required (Schema is Opaque)** and click **Next**. The Finish page is displayed.

9. Click **Finish**. You have configured the inbound adapter service, and the composite.xml page is displayed, as shown in Figure 10-79.

Figure 10-79 The JDeveloper Page - Composite.xml Page



Creating an Asynchronous Outbound Request Reply Adapter Service Outbound

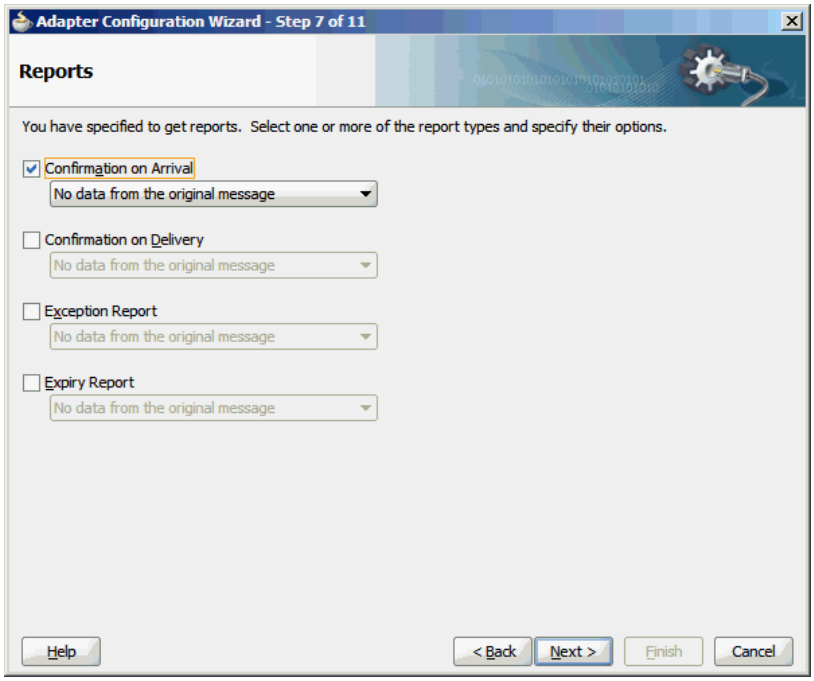
Perform the following steps to create an adapter service that enqueues the request messages and dequeue the corresponding response messages (report) from a queue:

1. Drag and drop **MQ Adapter** from the Components window into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.

2. Click **Next**. The Service Name page is displayed.

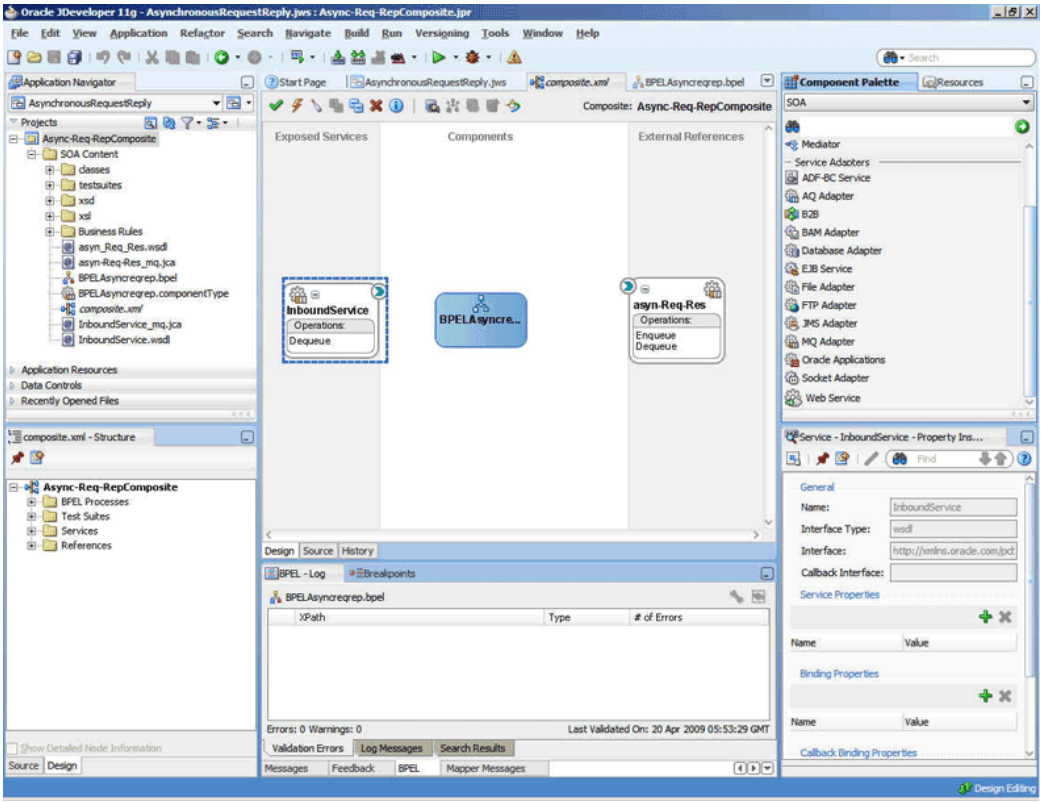
3. Enter `asyn-Req-Res` in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Send Message to MQ and Get Reply/Reports** and select **Asynchronous** in the Operation Name box, and then click **Next**. The Send Message to MQ and Get Reply/Reports page is displayed.
7. Select **Normal** in the Message Type box and enter `test_out` in the **Queue Name** field, and then select the **Get Reports** check box, and click **Next**. The Reports page is displayed.
8. Select **Confirmation on Arrival**, as shown in Figure 10-80, and click **Next**. The Response page is displayed.

Figure 10-80 The Adapter Configuration Wizard Reports Page



9. Enter `test_out` in the **Reply To Queue Name** field, and click **Next**. The Advanced Options page is displayed.
10. Accept the default values, and click **Next**. The Messages page is displayed.
11. Select **Native Format Translation is not Required(Schema is Opaque)** in both the Get Message Schema and Send Message Schema boxes, and click **Next**. The Finish page is displayed.
12. Click **Finish**. You have configured the **async-Req-Res** service, and the composite.xml page is displayed, as shown in Figure 10-81.

Figure 10-81 The JDeveloper Page - Composite.xml Page



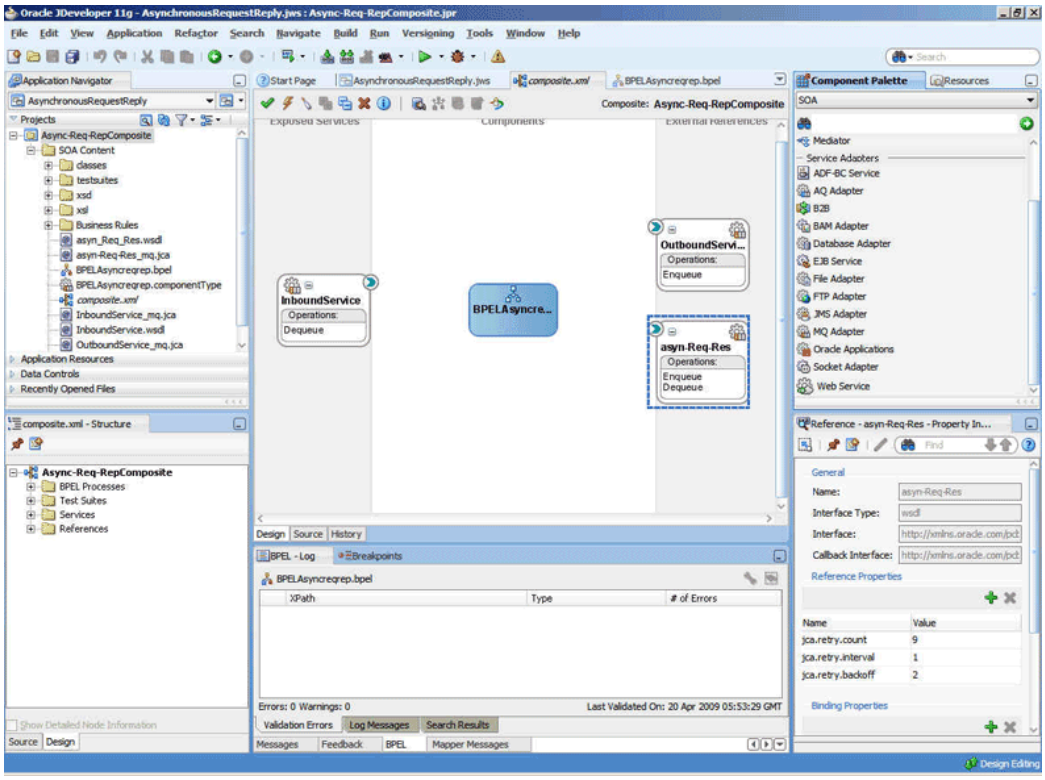
Creating Another Outbound Adapter Service

Perform the following steps to create an adapter service that enqueues the response (report) messages.

1. Drag and drop **MQ Adapter** from the Components window into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `OutboundService` in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Put Message into MQ**, and click **Next**. The Put Message into MQ page is displayed.
7. Enter `test_demo` in the **Queue Name** field, and click **Next**. The Advanced Options page is displayed.
8. Accept the default values, and click **Next**. The Messages page is displayed.
9. Select **Native Format Translation is not required(Schema is Opaque)**, and click **Next**. The Finish page is displayed.

10. Click **Finish**. You have configured the **OutboundService** service, and the composite.xml page is displayed, as shown in Figure 10-82.

Figure 10-82 The JDeveloper Page - Composite.xml Page



Wiring Services and Activities

You have to assemble or wire the four components that you have created: Inbound adapter service, BPEL process, async-Req-Res, and Outbound adapter reference. Perform the following steps to wire the components:

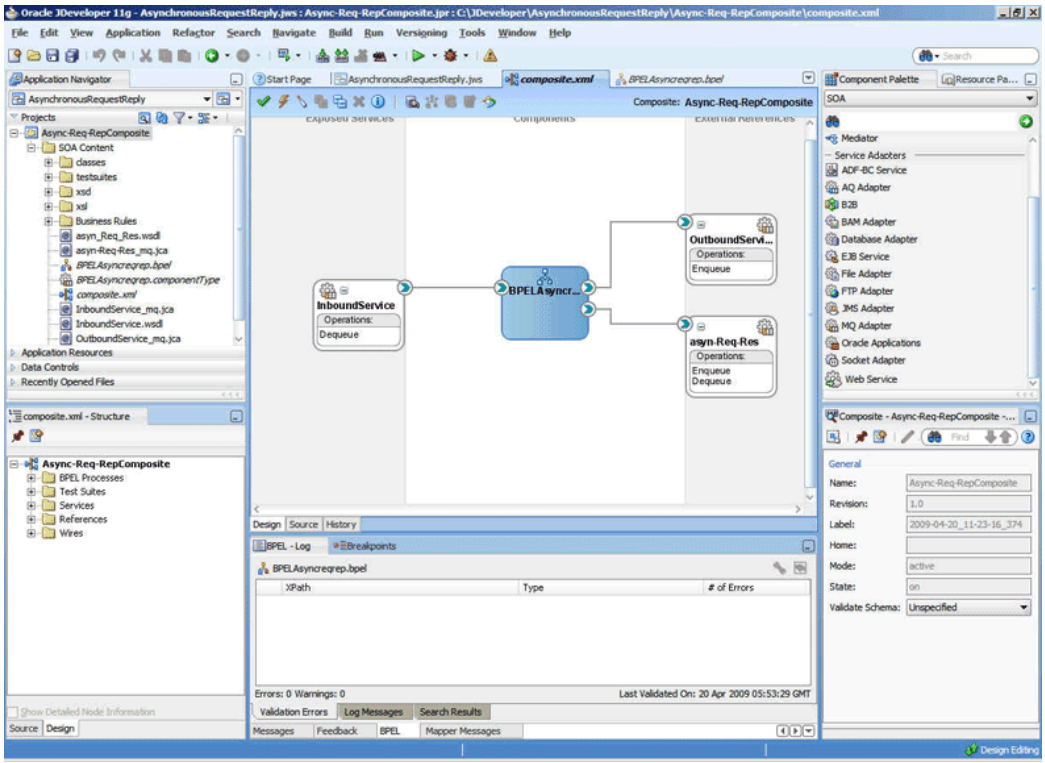
1. Drag the small triangle in the InboundService service in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.

2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in async-Req-Res in the External References area.

3. Similarly, drag the small triangle in the BPEL process in the Components area to the drop zone in OutboundService in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 10-83.

Figure 10-83 The JDeveloper - Composite.xml

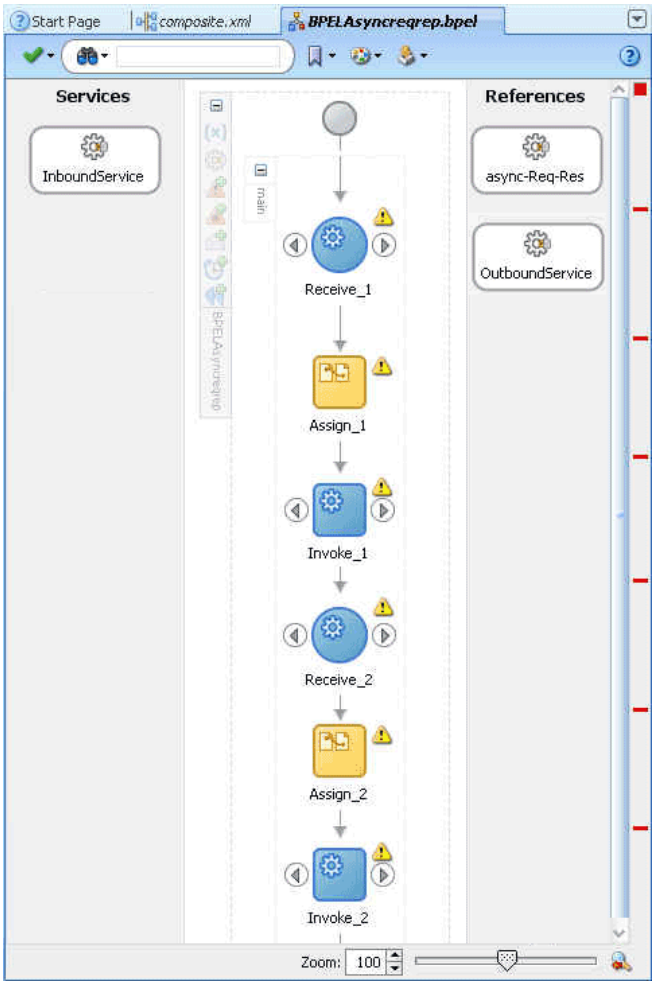


4. Click **File**, **Save All**.

5. Double-click **BPELAsyncreqrep**. TheDequeueEnqueueRFH2.bpel page is displayed.

6. Drag and drop the **Receive**, **Assign**, **Invoke**, **Receive**, **Assign**, **Invoke** activities in the order mentioned from the Components window to the Components area. The JDeveloper BPELAsyncreqrep.bpel page is displayed, as shown in Figure 10-84.

Figure 10-84 The BPELAsyncreqrep.bpel Page



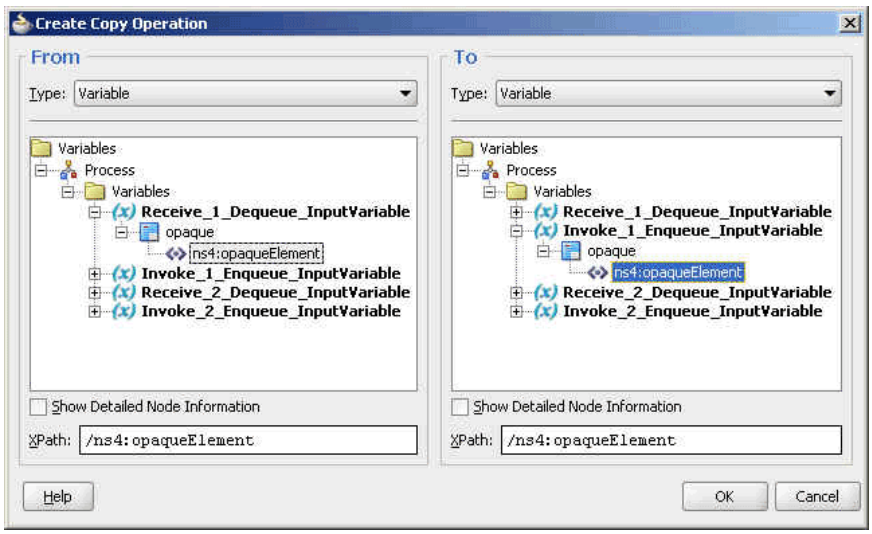
7. Drag and drop the first **Receive** activity to the InboundService adapter service. The Receive dialog is displayed.
8. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The Create Variable dialog is displayed.
9. Accept the defaults, and click **OK**.
10. Check the **Create Instance** box, and click **OK**.
11. Drag and drop the first **Invoke** activity to the async-Req-Res service. The Invoke dialog is displayed.
12. Click the **Automatically Create Input Variable** icon that appears at the end of the Input Variable field.
13. Accept the defaults, and click **OK**. The Invoke dialog is displayed.
14. Click **OK**.
15. Drag and drop the second **Receive** activity to the async-Req-Rep service. The Receive dialog is displayed.
16. Click the **Auto Create Variable** icon to create variable.

Note:

Do not check the Create Instance box.

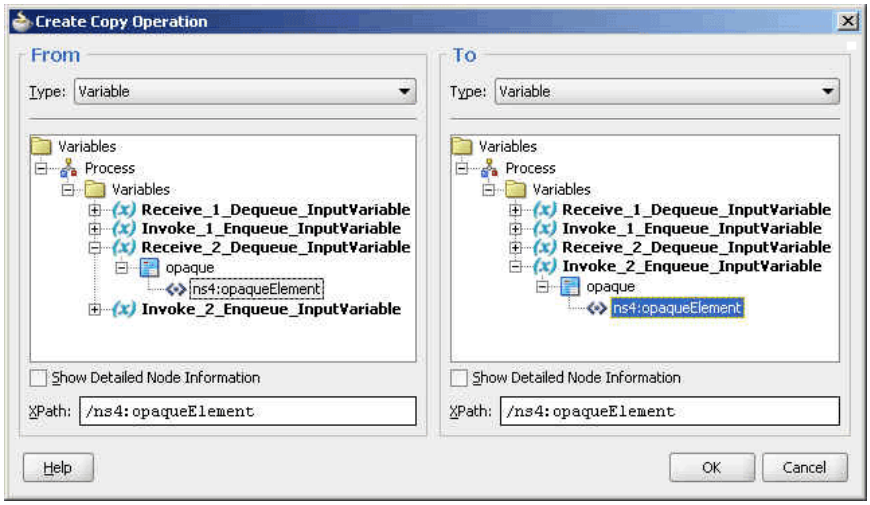
17. Click **OK** in the Receive dialog.
18. Drag and drop the second **Invoke** activity to OutboundService. The Invoke dialog is displayed.
19. Click the **Automatically Create Input Variable** icon to create a variable.
20. Click **OK** in the Invoke dialog.
21. Double-click the first **Assign** activity. The Assign dialog is displayed.
22. Click the plus icon, and select **Copy Operation**. The Create Copy Operation dialog is displayed.
23. Select the variables, as shown in Figure 10-85, and click **OK**.

Figure 10-85 The Create Copy Operation Dialog



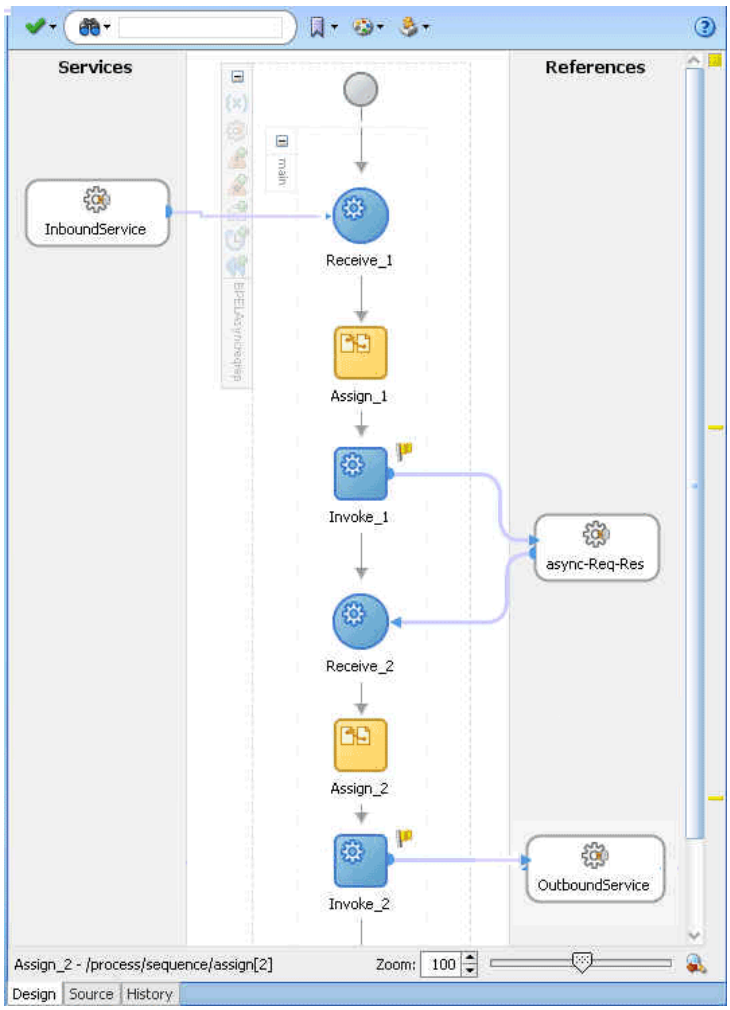
24. Click **OK** in the Assign dialog.
25. Double-click the second **Assign** activity. The Assign dialog is displayed.
26. Click the plus icon, and select **Copy Operation**. The Create Copy Operation dialog is displayed.
27. Select the variables, as shown in Figure 10-86, and click **OK**.

Figure 10-86 The Create Copy Operation Dialog



28. Click **OK** in the Assign dialog. The JDeveloper **BPELAsyncreqrep.bpel** page is displayed, as shown in Figure 10-87.

Figure 10-87 The BPELAsyncreqrep.bpel Page



Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Deploying Oracle JCA Adapter Applications from .

You must also create an application server connection. For more information about creating an application server connection, see Creating an Application Server Connection for Oracle JCA Adapters.

Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Log in to *http://servername:portnumber/em* using your username/password. The Oracle Fusion Middleware Control page is displayed.
2. In the left pane, navigate to **SOA**, **soa-infra (soa_server1)**. A list of all the composites that are deployed appears.
3. Click **Async-Req-RepComposite[1.0]**. The Async-Req-RepComposite[1.0] page is displayed.
4. Put a message that has the content that conforms to the address-csv.xsd and also contains the Reply Queue as the header in the test_in queue.
5. Wait for some time and then refresh the Fusion Middleware Control Console. An instance appears on the console. This is the instance that was triggered because of the processing that occurred.
6. Click the **Instances** tab.
7. Click the instance associated with this deployment. The Flow Trace page is displayed.
8. Click the **BPELAsyncreqrep** component instance. The Audit Trail page is displayed.
9. Click the **Flow** tab to debug the instance. The BPEL process instance flow is displayed.
10. Click an activity to view the relevant payload details.

Outbound Dequeue

This use case is the end-to-end demonstration of how MQ Adapter dequeues a single message at a time. This section contains the following topics:

- Prerequisites
- Designing the SOA Composite
- Creating an Outbound Dequeue Adapter Service
- Wiring Services and Activities
- Deploying with JDeveloper

- Monitoring Using the Fusion Middleware Control Console

Prerequisites

To perform the outbound dequeue use case, you require the following files from the `Adapters-101MQAdapterDequeueEnqueue` sample:

- `De-queueEn-queue/De-queueEn-queueComposite/xsd/singleString.xsd`

You also require the following files from the `artifacts.zip` file contained in the `Adapters-101MQAdapterDequeueEnqueue` sample:

- `artifacts/input/data.txt`

You can onbtain the `Adapters-101MQAdapterDequeueEnqueue` sample by accessing the Oracle SOA Sample Code site.

You must also create a queue named `test_out`.

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.

2. Expand the **General** node, and select the **Applications** category.

3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.

4. In the Name Your Application screen, enter `OutboundDequeue` in the **Application Name** field, and then click **Next**. The Name Your Project screen is displayed.

5. In the **Project Name** field, enter `OutboundDequeueComposite` and from the **Available** list, select SOA and click the right-arrow button.

6. Click **Next**. The Configure SOA Settings screen is displayed.

7. In the Composite Template list, select **Composite With BPEL** and then click **Finish**. The Create BPEL Process dialog is displayed.

8. Enter `BPELOutboundDequeue` in the **Name** field, select **Synchronous BPEL Process** in the Template box.

9. Click **Browse** at the end of the Input field. The Type Chooser dialog is displayed.

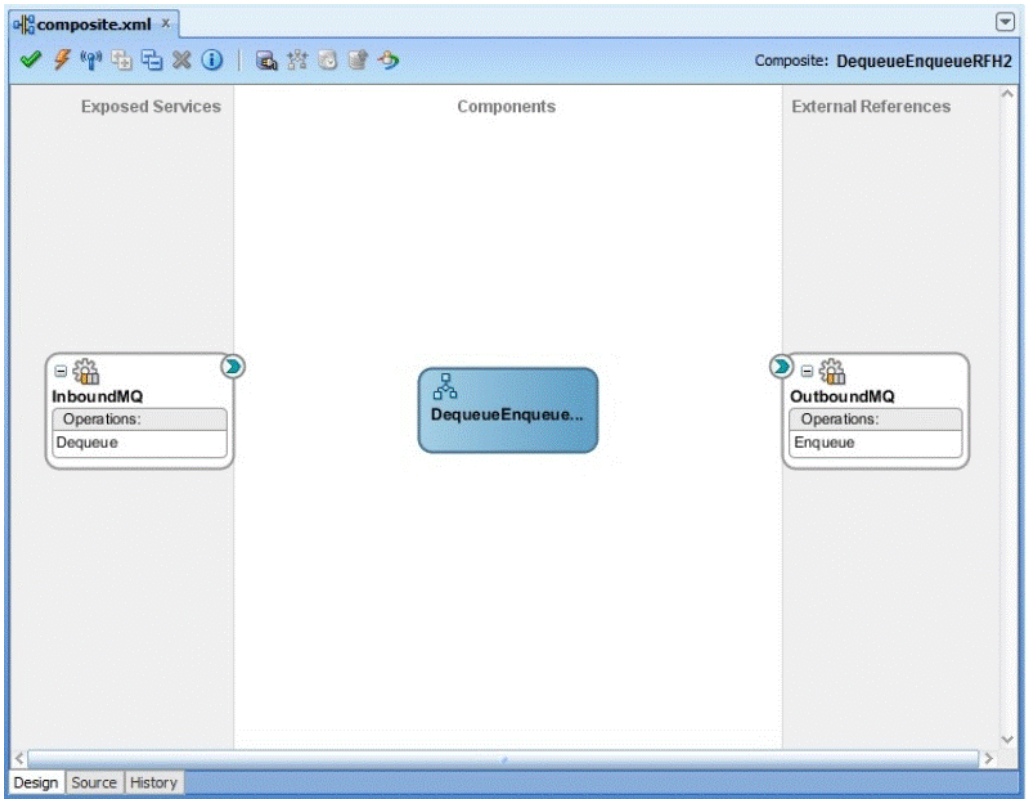
10. Select **Project Schema Files**, **singleString.xsd**, **singleString**, and then click **OK**.

11. Click **Browse** at the end of the Output field. The Type Chooser dialog is displayed.

12. Select **Project Schema Files**, **singleString.xsd**, **singleString**, and then click **OK**.

13. Click **OK**. The OutboundDequeue application and OutboundDequeueComposite project appears in the design area, as shown in Figure 10-88.

Figure 10-88 The JDeveloper - Composite.xml



Creating an Outbound Dequeue Adapter Service

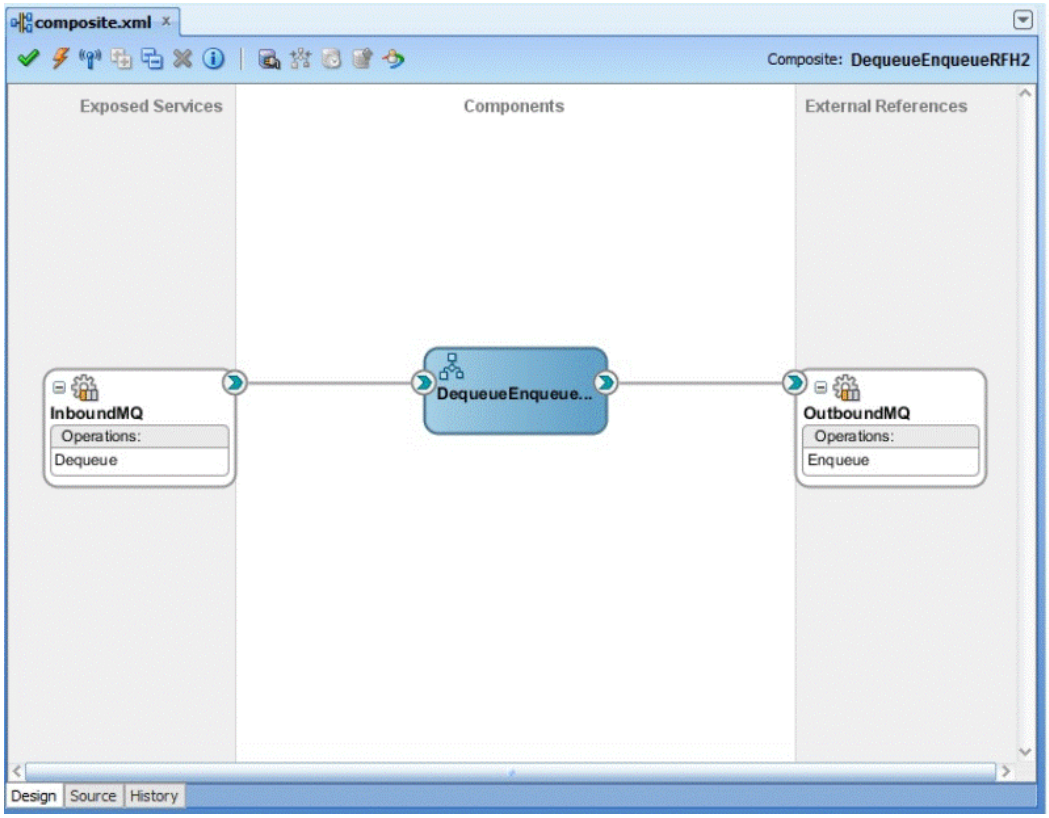
Perform the following steps to create an adapter service that dequeues the message to a queue:

1. Drag and drop **MQ Adapter** from the Components window into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `OutboundDequeueService` in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Get Message from MQ** and **Synchronous**, and click **Next**. The Get Message from MQ page is displayed.
7. Enter `test_out` in the **Queue Name** field and enter `10` in the **Wait Interval** field, and then click **Next**. The Messages page is displayed.
8. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
9. Select **Project Schema Files**, **singleString.xsd**, and then **singleString**, and click **OK**. The singleString.xsd file appears in the URL field in the Messages page.

10. Click **Next**. The Finish page is displayed.

11. Click **Finish**. You have now configured the inbound adapter service, and the composite.xml page is displayed, as shown in Figure 10-89.

Figure 10-89 The JDeveloper Page - Composite.xml Page



12. Click **File**, **Save All**.

Wiring Services and Activities

You have to assemble or wire the three components that you have created: Client, BPEL process, and Outbound adapter reference. Perform the following steps to wire the components:

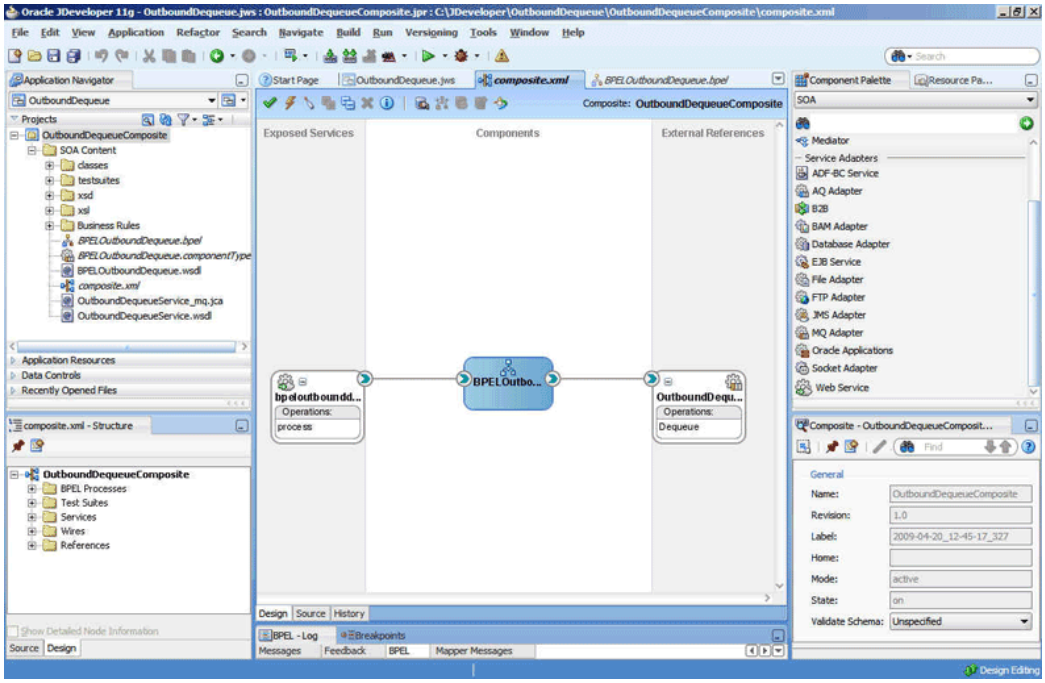
1. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in OutboundDequeueService in the External References area.

2. Double-click **BPELOutboundDequeue**. The BPELOutboundDequeue.bpel page is displayed.

3. Drag and drop the **Invoke** and **Assign** activities in the order mentioned from the Components window to the Components area in between the receiveInput and replyOutput activities.

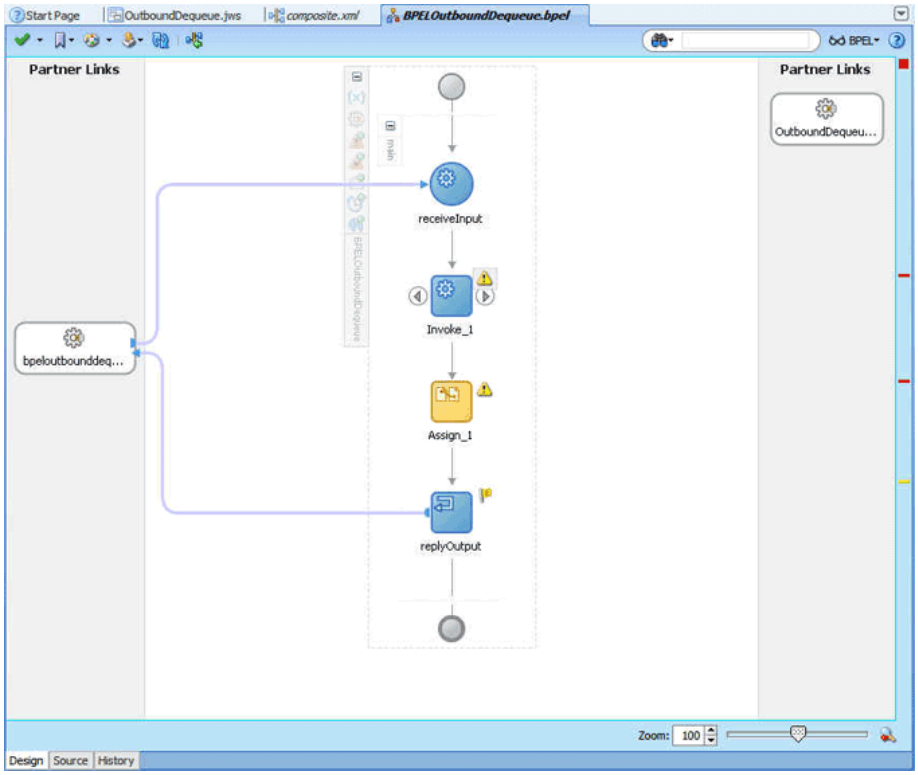
   The composite.xml page is displayed, as shown in Figure 10-90.

Figure 10-90 The JDeveloper - Composite.xml Page



The JDeveloper BPELOutboundDequeue.bpel page is displayed, as shown in Figure 10-91.

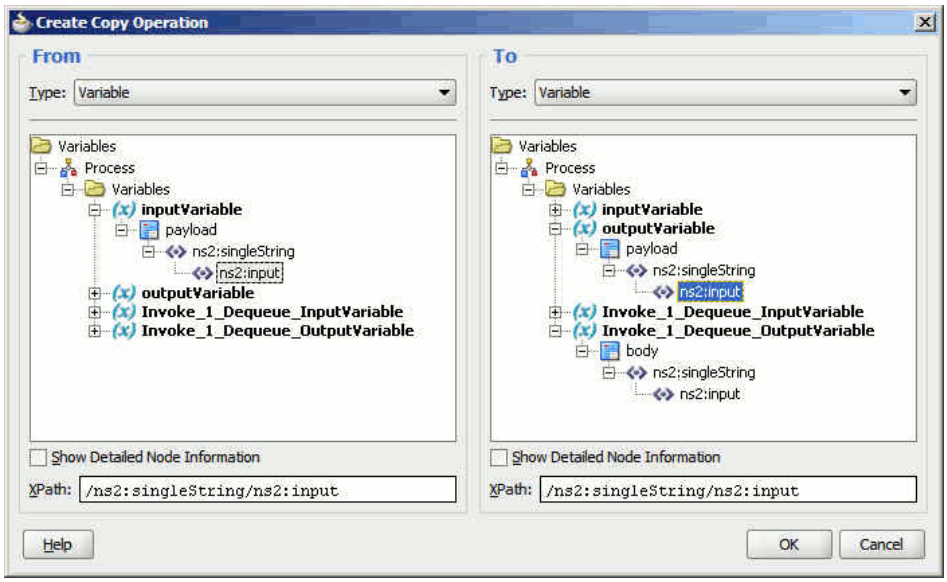
Figure 10-91 The BPELOutboundDequeue.bpel Page



4. Drag and drop the **Invoke** activity to the OutboundDequeueService adapter reference. The Invoke dialog is displayed.

5. Click the **Auto Create Variable** icon that appears at the end of the Input Variable field. The **Create Variable** dialog is displayed.
6. Accept the defaults, and click **OK**.
7. Repeat the same for the output variable and click **OK**.
8. Double-click the **Assign** activity. The Assign dialog is displayed.
9. Click the plus icon and select Copy Operation. The Create Copy Operation dialog is displayed.
10. Select the variables, as shown in Figure 10-92, and then click **OK**.

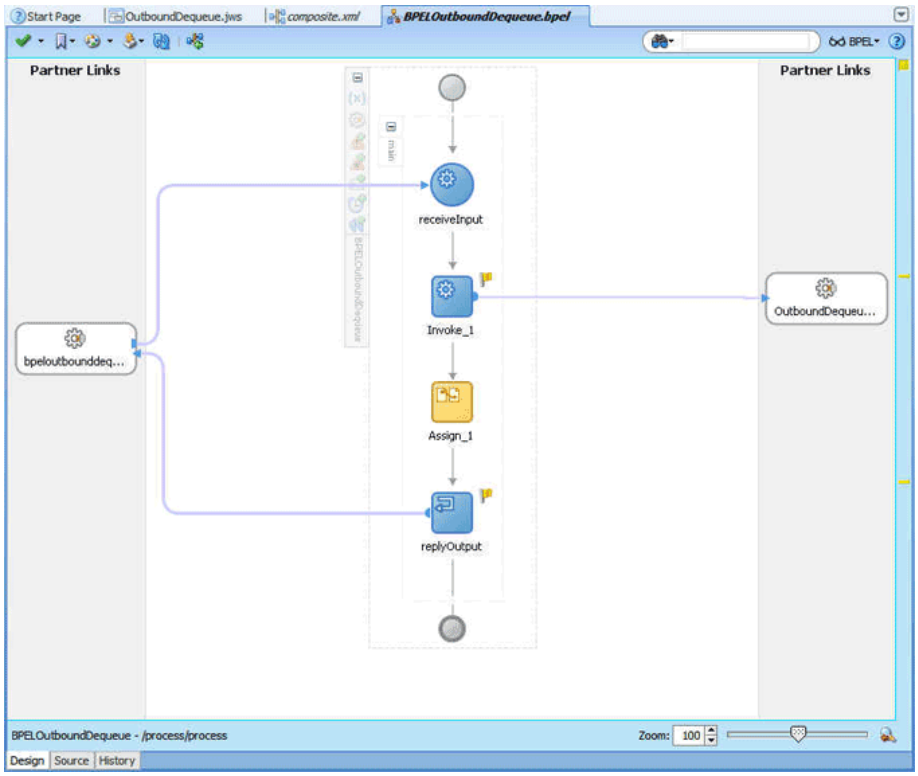
Figure 10-92 Create Copy Operation Dialog



11. Click **OK** in the Assign dialog.

    The BPELOutboundDequeue.bpel page appears, as shown in Figure 10-93.

Figure 10-93 The BPELOutboundDequeue.bpel Page



Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Deploying Oracle JCA Adapter Applications from .

You must also create an application server connection. For more information about creating an application server connection, see Creating an Application Server Connection for Oracle JCA Adapters.

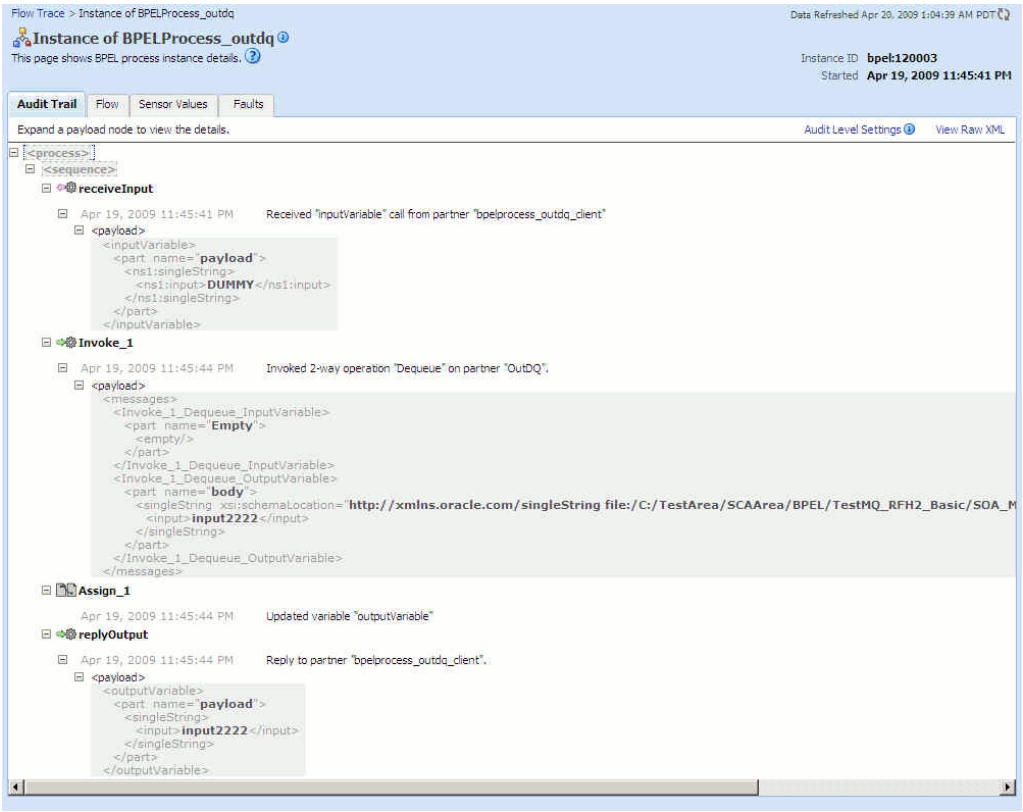
Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Log in to *http://servername:portnumber/em* using your username/password.
2. In the left pane, navigate to **SOA**, **soa-infra (soa_server1)**. A list of all the composites that are deployed appears.
3. Click **OutboundDequeueComposite[1.0]**. The OutboundDequeueComposite[1.0] page is displayed.
4. Click the **Test** button. The Test Web Service page is displayed.
5. Click the **Request** tab, and scroll to the Input Arguments pane.
6. Enter `Test Outbound Dequeue` in the **Input** field, and then click the **Test Web Service** button.

7. Wait for some time and then click the **Response** tab. The message in the singleString xsd that you provided appears in the Response tab.
8. Click the **Instances** tab.
9. Click the instance associated with this deployment. The Flow Trace page is displayed.
10. Click the **BPELOutboundDequeue** component instance. The Audit Trail page is displayed, as shown in Figure 10-94.

Figure 10-94 Audit Trail Page



11. Click the **Flow** tab to debug the instance. The BPEL process instance flow is displayed.
12. Click an activity to view the relevant payload details.

Configuring a Backout Queue

This use case demonstrates how a backout queue must be configured for Oracle MQ Series Adapter. Oracle MQ Series Adapter dequeues a message and enqueues the same message after transformation from the MQ Series queue. During this process, a failure can occur either during an invoke activity or when a response is being sent. You must configure a Backout Queue to send the rejected messages to a Backout Queue instead of the default rejected messages folder. This section contains the following topics:

- Prerequisites
- Designing the SOA Composite

- Creating an Inbound Adapter Service
- Creating an Outbound Adapter Service
- Wiring Services and Activities
- Deploying with JDeveloper
- Monitoring Using the Fusion Middleware Control Console

Prerequisites

To perform the use case for configuring a backout queue, you must ensure that the adapter JNDI is configured for XA. Also, you require the `singleString.xsd` file, which you can create using the following code:

```
<schema attributeFormDefault="unqualified"
      elementFormDefault="qualified"
      targetNamespace="http://xmlns.oracle.com/singleString"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="singleString">
            <complexType>
                  <sequence>
                        <element name="input" type="string"/>
                  </sequence>
            </complexType>
      </element>
</schema>
```

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.

2. Expand the **General** node, and select the **Applications** category.

3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.

4. In the Name Your Application screen, enter `MQ_BackoutQ_Retry` in the **Application Name** field, and then click **Next**. The Name Your Project screen is displayed.

5. In the **Project Name** field, enter `SOA_BackoutQ_Retry` and from the **Available** list, select **SOA** and click the right-arrow button.

6. Click **Next**. The Configure SOA Settings screen is displayed.

7. In the Composite Template list, select **Composite With BPEL**, and then click **Finish**. The Create BPEL Process dialog is displayed.

8. Enter `BPELProcess_BackoutQ_Retry` in the **Name** field, and select **Define Service Later** from the Template box.

9. Click **OK**. The MQ_BackoutQ_Retry application and the SOA_BackoutQ_Retry project appears in the design area.

Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message from a queue:

1. Drag and drop **MQ Adapter** from the Components window into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `InboundService` in the **Service Name** field, and click **Next**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Get Message from MQ and Send Reply/Reports** (Synchronous), and click **Next**. The Get Message from MQ and Send Rep page is displayed.
7. Enter `INBOUND_QUEUE` in the **Queue Name** field, and click **Next**. The Response page is displayed.
8. Accept the defaults, and click **Next**. The Messages page is displayed.
9. Click **Browse** at the end of the URL fields. The Type Chooser dialog is displayed.
10. Select **Project Schema Files**, **singleString.xsd**, and then **singleString**.
11. Click **OK**. The singleString.xsd file appears in the URL fields in the Messages page.
12. Click **Next**. The Finish page is displayed.
13. Click **Finish**. You have now configured the inbound adapter service, and the composite.xml page is displayed with an inbound adapter added.
14. Add the Backout Queue properties to the corresponding `JCA` file (`ReqReply_mq.jca`), as shown in the following sample:

```
<property name="BackoutQueueName" value="BACKOUT.QUEUE"/>
<property name="MaximumBackoutCount" value="5"/>
<property name="BackoutRetries" value="3"/>
```

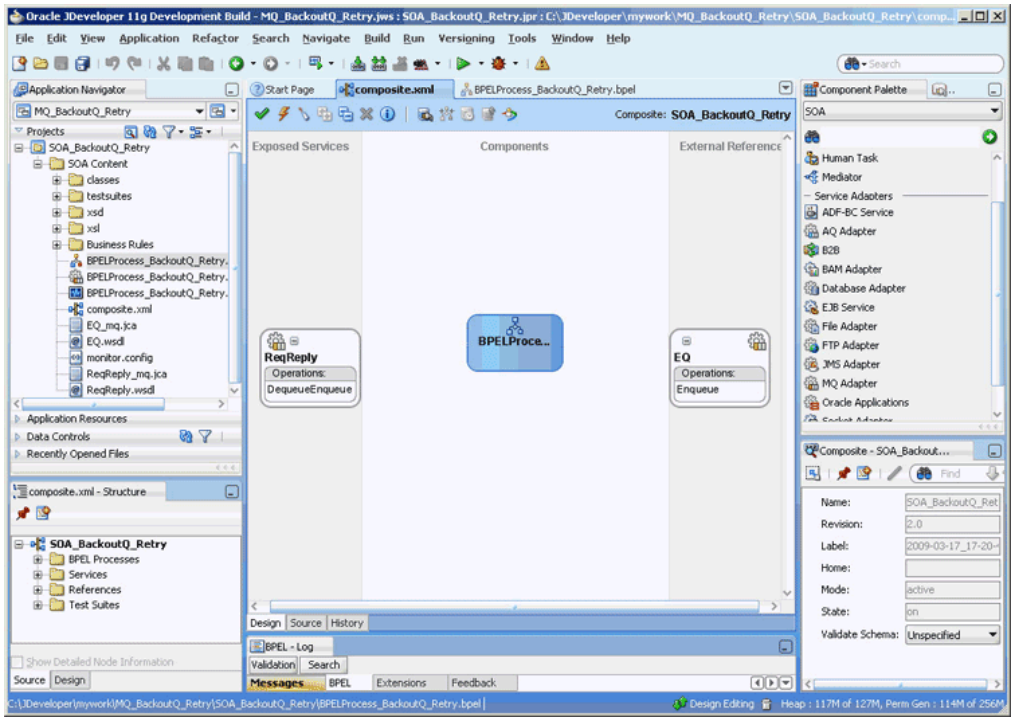
Creating an Outbound Adapter Service

Perform the following steps to create an adapter service that enqueues the messages.

1. Drag and drop **MQ Adapter** from the Components window into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `EQ` in the **Service Name** field, and click **Next**. The MQ Series Connection page is displayed.

4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Put Message into MQ**, and click **Next**. The Put Message into MQ page is displayed.
7. Enter a `test_out` in the **Queue Name** field, and click **Next**. The Advanced Options page is displayed.
8. Accept the defaults and click **Next**. The Messages page is displayed.
9. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
10. Select **Project Schema Files**, **singleString.xsd**, and then **singleString**, and click **OK**. The singleString.xsd file appears in the URL field in the Messages page.
11. Click **Next**. The Finish page is displayed.
12. Click **Finish**. You have now configured the outbound adapter service, and the composite.xml page is displayed, as shown in Figure 10-95.

Figure 10-95 The JDeveloper Page - Composite.xml Page



Wiring Services and Activities

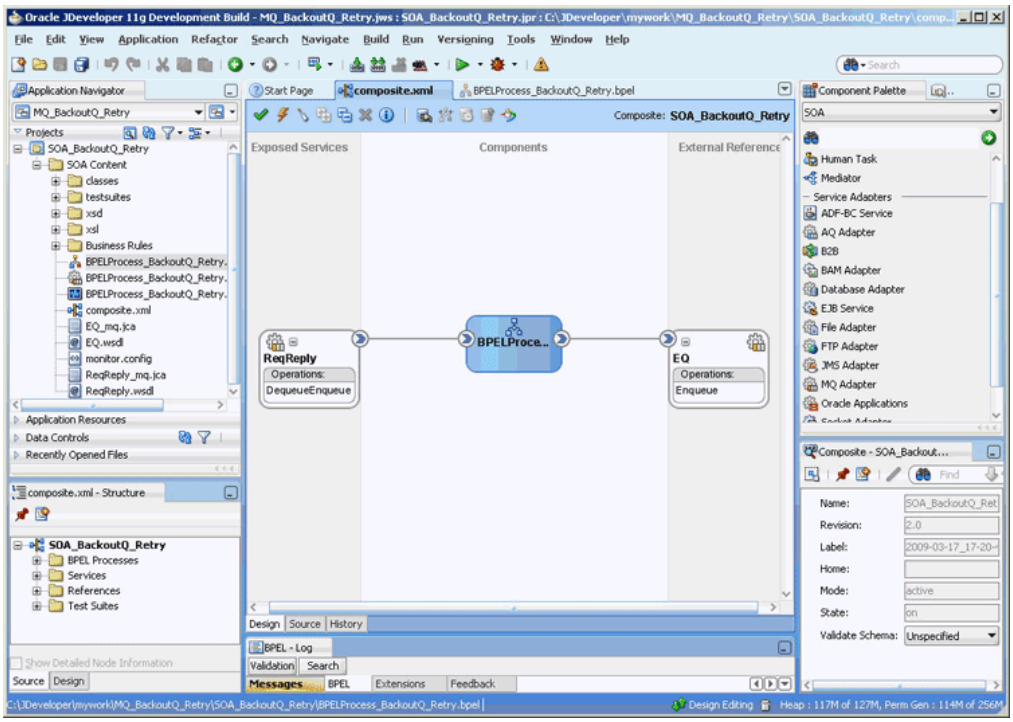
You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the InboundService in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.

2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in OutboundService in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 10-96.

Figure 10-96 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.

4. Double-click **BPELProcess_BackoutQ_Retry**. The BPELProcess_backoutQ_Retry.bpel page is displayed.

5. Drag and drop the **Receive**, **Assign**, **Invoke**, and **Reply** activities in the order mentioned from the Components window to the Components area.

6. Drag and drop the **Receive** activity to ReqReply. The Receive dialog is displayed.

7. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The Create Variable dialog is displayed.

8. Accept the defaults, and click **OK**.

9. Check the **Create Instance** box, and click **OK**.

10. Drag and drop the **Reply** activity to ReqReply. The Reply dialog is displayed.

11. Enter `ReplyOutput` in the **Name** field.

12. Click the **Browse Variables** icon that appears at the end of the Variable field. The Variable Chooser dialog is displayed.

13. Select **replyOutput_DequeueEnqueue_OutputVariable**, and click **OK**. The variable appears in the Reply dialog.

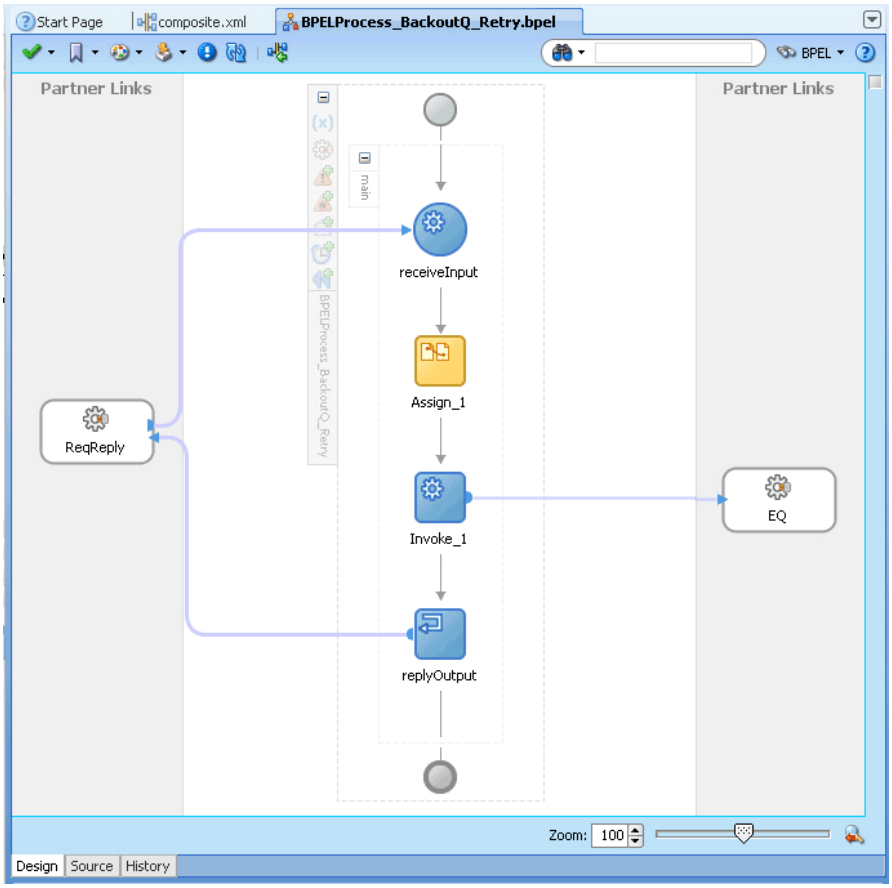
14. Click **OK**.
15. Drag and drop the **Invoke** activity to EQ. The Invoke dialog is displayed.
16. Click the **Automatically Create Input Variable** icon that appears at the end of the Input Variable field.
17. Accept the defaults, and click **OK**. The Invoke dialog is displayed.
18. Click **OK**.
19. Double-click the **Assign** activity. The Assign dialog is displayed.
20. Click the plus icon, and select **Copy Operation**. The Create Copy Operation dialog is displayed.
21. In the Create Copy Operation dialog, select `receiveInput_DequeueEnqueue_InputVariable` as the From Type and select the variable in the To pane to which the copy operation is being created.

    The following is a code snippet from the BPELProcess_BackoutQ_Retry.bpel file, with the copy operation defined:

```
<assign name="Assign_1">
      <copy>
        <from variable="receiveInput_DequeueEnqueue_InputVariable"
              part="singleString" query="/ns3:singleString/ns3:input"/>
        <to variable="Invoke_1_Enqueue_InputVariable" part="body"
            query="/ns3:singleString/ns3:input"/>
      </copy>
      <copy>
        <from variable="receiveInput_DequeueEnqueue_InputVariable"
              part="singleString" query="/ns3:singleString/ns3:input"/>
        <to variable="replyOutput_DequeueEnqueue_OutputVariable"
            part="singleString" query="/ns3:singleString/ns3:input"/>
      </copy>
    </assign>
```

22. Click **OK**. The BPELdequeueenqueue.bpel page appears, as shown in Figure 10-97.

Figure 10-97 The BPELProcess_BackoutQ_Retry.bpel Page



Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Deploying Oracle JCA Adapter Applications from .

You must also create an application server connection. For more information about creating an application server connection, see Creating an Application Server Connection for Oracle JCA Adapters.

Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Disable (Put Inhibit) the test_out queue or the reply queue that is provided with the Inbound message and then put a message to the `INBOUND_QUEUE`.
3. Wait for some time and then refresh the Fusion Middleware Control Console. Instances that are triggered because of the processing appear on the console.

Note:

The number of instances that are triggered must be equal to `BackoutRetries + 1`.

4. Click the **Instances** tab.
5. Click an instance associated with this deployment. The Flow Trace page is displayed.
6. Click the **BPELProcess_BackoutQ_Retry** component instance. The Audit Trail page is displayed.
7. Click the **Flow** tab to debug the instance. The BPEL process instance flow is displayed.
8. Click an activity to view the relevant payload details.

CCDT Use Cases

You can configure the MQ Series Adapter to use CCDT for connection details; you can use the CCDT to connect to the first available queue manager from a list of queue managers.

For example, in this use case, there are three queue managers QM1, QM2 and QM3 with the basic properties as indicated in the table bekow.

The MQ adapter can, dynamically at runtime, connect to any of the three queue managers, depending on the one that is available (that is, if QM1 is down, the MQ adapter automatically connects to QM2; and if QM2 is also down, then the MQ Adapter must connect to QM3).

Example Queue Manager Properties and CCDT Configuration

Example queue manager properties are provided in this table.

Queue Manager Name	QM1	QM2	QM3
Hostname	localhost	localhost	10.177.255.25
PortName	1414	2414	1414
Server Connection Channel Name	channel.QM1	channel.QM2	channel.QM3

To achieve this, the CCDT must be configured as follows :

Channel name	Queue manager name	Connection name
channel.QM1	MyQM_group	localhost(1414)
channel.QM2	MyQM_group	localhost(2414)
channel.QM3	MyQM_group	10.177.255.25(1414)

Configuringa ConnectionFactoryJNDI

Once you have created a CCDT, a single ConnectionFactory JNDI must be configured to use this CCDT. The following ConnectionFactory properties must be configured in that JNDI:

- `CCDTurl`
- `QueueManagerName`

Configuring the CCDTurl

The CCDTurl must point to the URL of the CCDT file that is used by the MQ Series Adapter to supply client connection details. For example, the values provided can be either of:

- `file:/scratch/username/ccdt/AMQCLCHL.TAB`
- `ftp://userName:password@myServer/definitionPath/AMQCLCHL.TAB`

Configuring the QueueManagerName

In the use case, the value of the QueueManagerName property must be set to `MyQM_group`.

This name indicates that MQ Series Adapter should connect to the first available queue manager which has a client definition entry in the CCDT having Queue manager name as `MyQM_group`.

The property QueueManagerName is matched against the queue manager name defined in the CCDT and not the actual queue manager name. In general, the QueueManagerName should be provided appropriate values, using the considerations listed below:

- If you only specify a queue manager name, for example `QM_default`, the CCDT will be searched in alphabetical order for a client channel that contains in its definition a QueueManager name that matches exactly (and is case sensitive) to the one specified
- If an asterisk is included at the beginning of the specified queue manager name, for example `QM_default`, then the CCDTis searched in alphabetical order of channel name, for an entry that matches the queue manager name with or without the asterisk. If two or more client channel definitions have the queue manager names defined as QM_default, then the first available queue manager is connected to.
- If the queue manager name is not specified in the CCDT, then the Queue Manager name in the JNDI should be a "*".

These two properties are the only required configuration for informing the MQ Series Adapter to use CCDT for connection details. The use of CCDT does not affect the configuration required for other MQ adapter features such as support for SSL, Exits or XA

If other ConnectionFactory properties such as Hostname, PortNumber, ChannelName are configured and the CCDT is also configured, the CCDT will take precedence over the those properties.

Once these ConnectionFactory properties are set, and this JNDI is used in any composite process, the MQ Adapter connects to the first available queue manager from QM1, QM2 and QM3.

Reading Single or Multiple RFH2 Rules and Formatting Header Version 2 Headers

You can dequeue and read MQ messages that contain single or multiple RFH2 headers, in addition to enqueueing messages with multiple RFH2 headers. Ths feature includes the following functionality:

- Reading and writing the properties from the fixed portion of the RFH2 headers.
- Reading and writing multiple occurrences of any individual folder within any RFH2 header.
- Reading and writing multiple RFH2 header occurrences in a single message.

The RFH2 header enables the message producer to add more header properties to the payload and to provide other additional information.

By providing the MQ adapter the ability to read and write this information as header properties, you can perform specific processing of the message payload depending on the RFH2 header properties.

The following use cases provide examples of two types of scenarios.

- Inbound and Outbound with Multiple RFH2 Headers on Both Sides
- Outbound Dequeue with Multiple RFH2 Headers

Inbound and Outbound with Multiple RFH2 Headers on Both Sides

This sample demonstrates the use of the MQ Adapter for processing MQ messages containing one or more RFH2 headers.

An MQ message containing two RFH2 headers is dequeued from a queue and a new message with the same payload, but with updated RFH2 headers is enqueued to another queue.

The steps for creating this use case sample include:

- Designing the SOA Composite
- Creating an Inbound Adapter Service
- Creating an Outbound Adapter Service
- Wiring Services and Activities
- Deploying with JDeveloper

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.

2. Expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.

4. In the Name Your Application screen, enter `DequeueEnqueueRFH2` in the **Application Name** field, and then click **Next**. The Name Your Project screen is displayed.
5. In the **Project Name** field, enter `DequeueEnqueueRFH2` and from the **Available** list, select **SOA** and click the right-arrow button.
6. Click **Next**. The **Configure SOA Settings** screen is displayed.
7. In the Composite Template list, select **Composite With BPEL**, and then click **Finish**. The Create BPEL Process dialog is displayed.
8. Enter `DequeueEnqueueRFH2` in the **Name** field, and select **Define Service Later** from the Template box.
9. Click **OK**. The `DequeueEnqueueRFH2` application and the `DequeueEnqueueRFH2` project appears in the design area.

Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message and put the message to a queue:

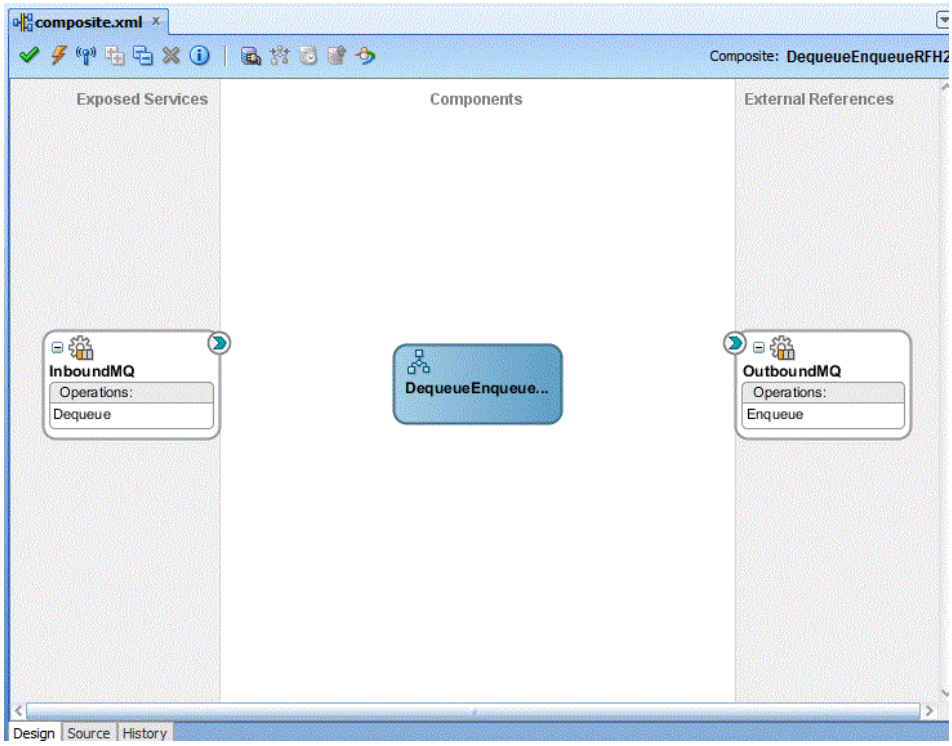
1. Drag and drop **MQ Adapter** from the Components window into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `InboundMQ` in the **Service Name** field, and click **Next**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Get Message from MQ** (do not check Synchronous), and click **Next**. The Get Message from MQ page is displayed.
7. Enter `queue1` in the **Queue Name** field, select Choose Other Schema, click **Next**. The Messages page is displayed.
8. Select **Native Format is not required (Schema is opaque)**, and click **Next**. The Finish page is displayed.
9. Click **Finish**. You have now configured the inbound adapter service, and the composite.xml page is displayed with an inbound adapter added.

Creating an Outbound Adapter Service

1. Drag and drop **MQ Adapter** from the Components window into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.

3. Enter `OutboundMQ` in the **Service Name** field, and click **Next**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Put Message into MQ**. Click Next. The Put Message into MQ page is displayed.
7. Enter `queue2` in the **Queue Name** field. Click Next. The Advanced Options page is displayed.
8. Accept the defaults and click **Next**. The Messages page is displayed.
9. Select `Native Format Translation is Not Required (Schema is Opaque)` and click **Next**. The Finish page is displayed.
10. Click **Finish**. You have now configured the outbound adapter service, and the `composite.xml` page is displayed, as shown in Figure 10-95.

Figure 10-98 The JDeveloper Page - Composite.xml Page



Wiring Services and Activities

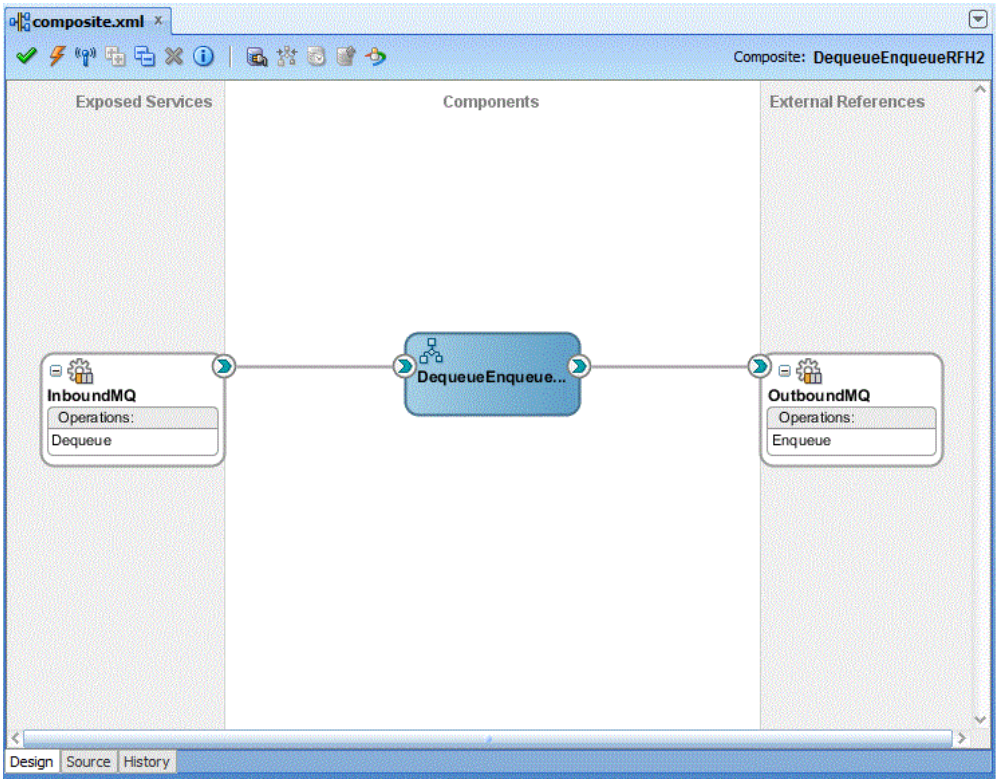
You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the InboundMQ in the ExposedServices area to the drop zone that appears as a green triangle in DequeueEnqueueRFH2 in the Components area.

2. Drag the small triangle in **DequeuEnqueueRFH2** in the Components area to the drop zone that appears as a green triangle in OutboundMQ in the External References area.

   The composite.xml page is displayed, as shown in Figure 10-90.

Figure 10-99 The JDeveloper - Composite.xml Page

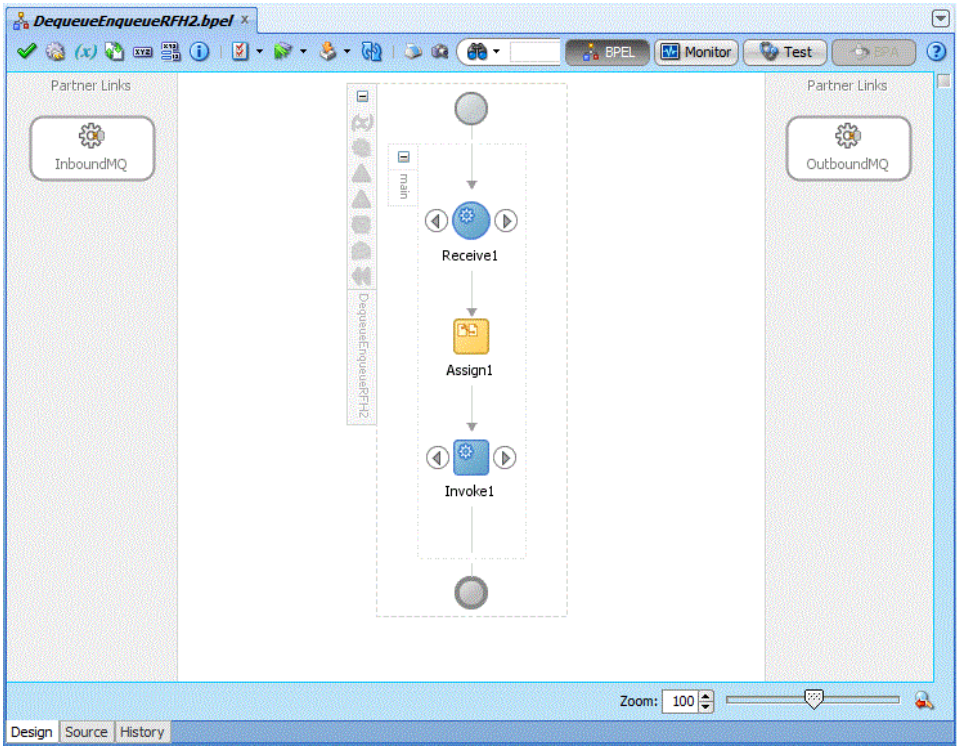


3. Click **File**, **Save All**.

4. Double-click DequeueEnqueueRFH2. The DequeueEnqueueRFH2.bpel page is displayed.

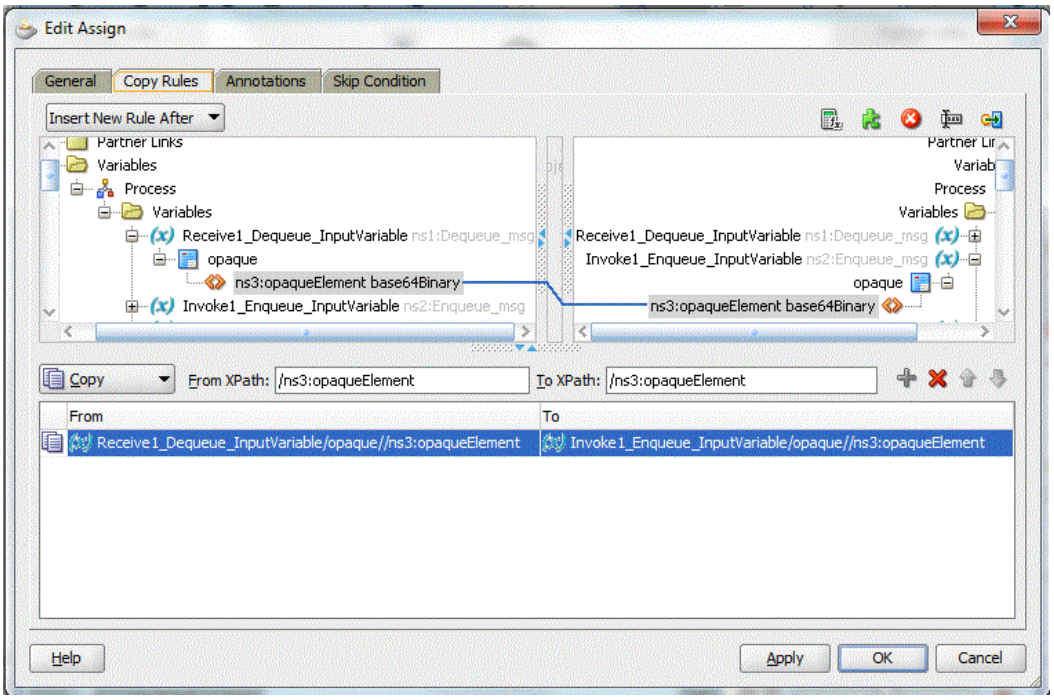
5. Drag and drop the **Receive**, **Assign** and **Invoke** activities in the order mentioned from the Components windowto the Components area. The **JDeveloper DequeueEnqueueRFH2.bpel** page is displayed.

Figure 10-100 The DequeueEnqueueRFH2.bpel Page"



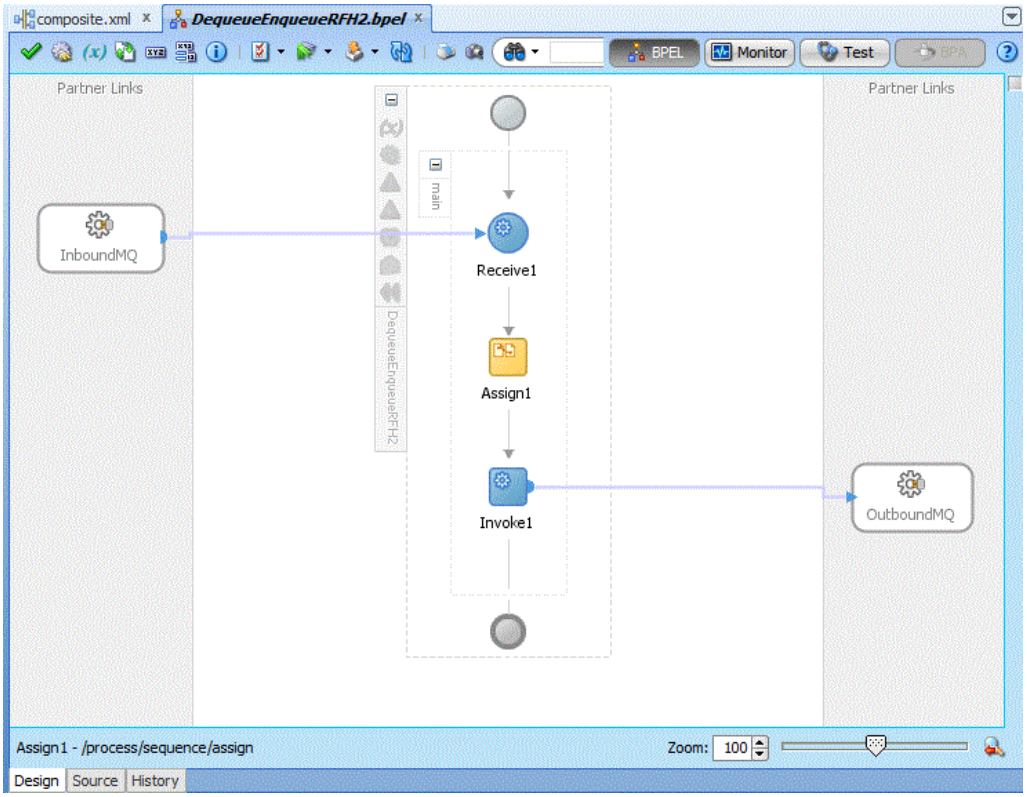
6. Drag and drop the **Receive** activity to the Inbound Service. The **Receive** dialog is displayed.
7. Click the **AutoCreate Variable** icon that appears at the end of the Variable field. The **Create Variable** dialog is displayed.
8. Accept the defaults, and click **OK**.
9. Click the Create Instance box, and click OK.
10. Drag and drop the **Invoke** activity to the Outbound Service. The Invoke dialog is displayed.
11. Click the **Automatically Create Input Input Variable** icon that appears at the end of the Input Variable field.
12. Accept the defaults, and click **OK**. The **Invoke** dialog is displayed.
13. Click **OK**.
14. Double-click the **Assign** activity. The **Assign** dialog is displayed.
15. Select the variables, and click the **Plus** icon.

Figure 10-101 The Assign Activity Dialog



16. Click **OK** in the **Assign** dialog. The JDeveloper `DequeueEnqueueRFH2.bpel.html` is displayed.

Figure 10-102 The DequeueEnqueueRFH2.bpel page



17. Create temporary variables to store the RFH2 header portions.

a. Open the Source tab of the `DequeueEnqueueRFH2.bpel` page.

b. Under the <variables> tab add the following new variables:

```
<variable name="RFH2.StructId" type="xsd:string"/>
<variable name="RFH2.Version" type="xsd:string"/>
<variable name="RFH2.Encoding" type="xsd:string"/>
<variable name="RFH2.CodedCharSetId" type="xsd:string"/>
<variable name="RFH2.Format" type="xsd:string"/>
<variable name="RFH2.Flags" type="xsd:string"/>
<variable name="RFH2.NameValueCCSID" type="xsd:string"/>
<variable name="RFH2.JMSFolder" type="xsd:string"/>
<variable name="RFH2.MCDFolder" type="xsd:string"/>
<variable name="RFH2.USRFolder" type="xsd:string"/>
<variable name="RFH2.USRFolder_2" type="xsd:string"/>
<variable name="RFH2.PSCFolder" type="xsd:string"/>
<variable name="RFH2extrafolder" type="xsd:string"/>
<variable name="RFH2_2.StructId" type="xsd:string"/>
<variable name="RFH2_2.Version" type="xsd:string"/>
<variable name="RFH2_2.Encoding" type="xsd:string"/>
<variable name="RFH2_2.CodedCharSetId" type="xsd:string"/>
<variable name="RFH2_2.Format" type="xsd:string"/>
<variable name="RFH2_2.Flags" type="xsd:string"/>
<variable name="RFH2_2.NameValueCCSID" type="xsd:string"/>
<variable name="RFH2_2.JMSFolder" type="xsd:string"/>
<variable name="RFH2_2.JMSFolder_2" type="xsd:string"/>
<variable name="RFH2_2.MCDFolder" type="xsd:string"/>
<variable name="RFH2_2.USRFolder" type="xsd:string"/>
<variable name="RFH2_2.USRFolder_2" type="xsd:string"/>
<variable name="RFH2_2.PSCFolder" type="xsd:string"/>
<variable name="TotalRFH2" type="xsd:string"/>
```

18. Configure the **Receive** Activity to receive RFH2 header properties in the temporary variables created.

a. Open the Source tab of the `DequeueEnqueueRFH2.bpel` page.

b. Under the <receive> tag add the following entries.

```
<bpelx:property name="jca.mq.RFH2.StructId" variable="RFH2.StructId"/>
<bpelx:property name="jca.mq.RFH2.Version" variable="RFH2.Version"/>
<bpelx:property name="jca.mq.RFH2.Encoding" variable="RFH2.Encoding"/>
<bpelx:property name="jca.mq.RFH2.CodedCharSetId"
variable="RFH2.CodedCharSetId"/>
<bpelx:property name="jca.mq.RFH2.Format" variable="RFH2.Format"/>
<bpelx:property name="jca.mq.RFH2.Flags" variable="RFH2.Flags"/>
<bpelx:property name="jca.mq.RFH2.NameValueCCSID"
variable="RFH2.NameValueCCSID"/>
<bpelx:property name="jca.mq.RFH2.JMSFolder" variable="RFH2.JMSFolder"/>
<bpelx:property name="jca.mq.RFH2.MCDFolder" variable="RFH2.MCDFolder"/>
<bpelx:property name="jca.mq.RFH2.USRFolder" variable="RFH2.USRFolder"/>
<bpelx:property name="jca.mq.RFH2.USRFolder_2" variable="RFH2.USRFolder_2"/>
<bpelx:property name="jca.mq.RFH2.PSCFolder" variable="RFH2.PSCFolder"/>
<bpelx:property name="jca.mq.RFH2.mq_usr" variable="RFH2extrafolder"/>
<bpelx:property name="jca.mq.RFH2_2.StructId" variable="RFH2_2.StructId"/>
<bpelx:property name="jca.mq.RFH2_2.Version" variable="RFH2_2.Version"/>
<bpelx:property name="jca.mq.RFH2_2.Encoding" variable="RFH2_2.Encoding"/>
<bpelx:property name="jca.mq.RFH2_2.CodedCharSetId"
variable="RFH2_2.CodedCharSetId"/>
<bpelx:property name="jca.mq.RFH2_2.Format" variable="RFH2_2.Format"/>
<bpelx:property name="jca.mq.RFH2_2.Flags" variable="RFH2_2.Flags"/>
<bpelx:property name="jca.mq.RFH2_2.NameValueCCSID" variable=
```

```
"RFH2_2.NameValueCCSID"/>
<bpelx:property name="jca.mq.RFH2_2.JMSFolder" variable="RFH2_2.JMSFolder"/>
<bpelx:property name="jca.mq.RFH2_2.JMSFolder_2"
variable="RFH2_2.JMSFolder_2"/>
<bpelx:property name="jca.mq.RFH2_2.MCDFolder" variable="RFH2_2.MCDFolder"/>
<bpelx:property name="jca.mq.RFH2_2.USRFolder" variable="RFH2_2.USRFolder"/>
<bpelx:property name="jca.mq.RFH2_2.PSCFolder" variable="RFH2_2.PSCFolder"/>
<bpelx:property name="jca.mq.RFH2_2.USRFolder_2"
variable="RFH2_2.USRFolder_2"/>
<bpelx:property name="jca.mq.RFH2.Total.Headers" variable="TotalRFH2"/>
```

19. Configure the **Invoke** activity to push modified RFH2 header properties to the outbound message.

a. Open the Source tab of the DequeueEnqueueRFH2.bpel page

b. Under the <invoke> tag add the following entries

```
      <bpelx:inputProperty name="jca.mq.MQMD.Format"
expression="'RF_HDR_2'"/>
      <bpelx:inputProperty name="jca.mq.RFH2.StructId"
variable="RFH2.StructId"/>
      <bpelx:inputProperty name="jca.mq.RFH2.Version"
variable="RFH2.Version"/>
      <bpelx:inputProperty name="jca.mq.RFH2.CodedCharSetId"
variable="RFH2.CodedCharSetId"/>
      <bpelx:inputProperty name="jca.mq.RFH2.Encoding"
variable="RFH2.Encoding"/>
      <bpelx:inputProperty name="jca.mq.RFH2.Flags" variable="RFH2.Flags"/>
      <bpelx:inputProperty name="jca.mq.RFH2.Format" expression="'MQHRF2
'"/>
      <bpelx:inputProperty name="jca.mq.RFH2.NameValueCCSID"
variable="RFH2.NameValueCCSID"/>
      <bpelx:inputProperty name="jca.mq.RFH2.JMSFolder"
variable="RFH2.JMSFolder"/>
      <bpelx:inputProperty name="jca.mq.RFH2.MCDFolder"
variable="RFH2.MCDFolder"/>
      <bpelx:inputProperty name="jca.mq.RFH2.USRFolder"
variable="RFH2.USRFolder"/>
      <bpelx:inputProperty name="jca.mq.RFH2.mq_usr"
variable="RFH2extrafolder"/>
      <bpelx:inputProperty name="jca.mq.RFH2_2.StructId" expression="'RFH
'"/>
      <bpelx:inputProperty name="jca.mq.RFH2_2.Version" expression="'2'"/>
      <bpelx:inputProperty name="jca.mq.RFH2_2.CodedCharSetId"
expression="'819'"/>
      <bpelx:inputProperty name="jca.mq.RFH2_2.Encoding"
expression="'273'"/>
      <bpelx:inputProperty name="jca.mq.RFH2_2.Flags" expression="'0'"/>
      <bpelx:inputProperty name="jca.mq.RFH2_2.Format" expression="'MQSTR
'"/>
      <bpelx:inputProperty name="jca.mq.RFH2_2.NameValueCCSID"
expression="'1208'"/>
      <bpelx:inputProperty name="jca.mq.RFH2_2.JMSFolder"
expression="'&lt;jms>&lt;Dst>MYTOPIC&lt;/Dst>&lt;Exp>2000&lt;/
Exp>&lt;Pri>4&lt;/Pri>&lt;Cid>22344&lt;/Cid>&lt;Rto>REPLY.QUEUE&lt;/
Rto>&lt;Gid>3334&lt;/Gid>&lt;Seq>2&lt;/Seq>&lt;Dlv>1&lt;/
Dlv>&lt;xxx>UserSpace&lt;/xxx>&lt;/jms>'"/>
      <bpelx:inputProperty name="jca.mq.RFH2_2.JMSFolder_2"
variable="RFH2_2.JMSFolder_2"/>
      <bpelx:inputProperty name="jca.mq.RFH2_2.MCDFolder"
expression="'&lt;mcd>&lt;Msd>jms_object&lt;/Msd>&lt;/mcd>'"/>
```

```
      <bpelx:inputProperty name="jca.mq.RFH2_2.USRFolder"
variable="RFH2_2.USRFolder"/>
      <bpelx:inputProperty name="jca.mq.RFH2_2.USRFolder_2"
variable="RFH2_2.USRFolder_2"/>
```

20. Click **File**, Save **All**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Deploying Oracle JCA Adapter Applications from .

You must also create an application server connection. For more information about creating an application server connection, see Creating an Application Server Connection for Oracle JCA Adapters.

You must place the MQ message with the correct RFH2 headers in the inbound queue.

The sample is configured to obtain two RFH2 headers from the inbound message (this can be changed by configuring the BPEL process as required). The test input MQ message should have the following:

- Payload: Any message (any format will be acceptable becaise we are using Opaque)
- RFH2 header 1: Should contain 2 USR folders, 1 JMS folder, 1 PSC folder, 1 MCD folder and 1 mq_usr folder.
- RFH2 header 2: Should contain 2 USR folders, 2 JMS folders, 1 PSC folder and 1 MCD folder.

Deploy the sample and put the message to the inbound queue. Check the outbound queue for a new message having modified RFH2 headers (as configured in the BPEL invoke activity)

Outbound Dequeue with Multiple RFH2 Headers

This sample demonstrates the use of the MQ Adapter for obtaining MQ messages containing one or more RFH2 headers in an outbound dequeue scenario.

In this sample, an MQ message containing two RFH2 headers is dequeued from a queue in the outbound queue scenario.

The steps for creating this use case sample include:

- Designing the SOA Composite
- Creating an Outbound Dequeue Adapter Service
- Wiring Services and Activities
- Deploying with JDeveloper

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.

2. Expand the **General** node, and select the **Applications** category.

3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.

4. In the Name Your Application screen, enter `OutboundDequeueRFH2` in the **Application Name** field, and then click **Next**. The Name Your Project screen is displayed.

5. In the **Project Name** field, enter `OutboundDequeueRFH2` and from the **Available** list, select SOA and click the right-arrow button.

6. Click **Next**. The Configure SOA Settings screen is displayed.

7. In the Composite Template list, select **Composite With BPEL** and then click **Finish**. The Create BPEL Process dialog is displayed.

8. Enter `OutboundDequeueRFH2` in the **Name** field, select **Synchronous BPEL Process** in the Template box.

9. Click **Browse** at the end of the Input field. The Type Chooser dialog is displayed.

10. Select **Project Schema Files**, **singleString.xsd**, **singleString**, and then click **OK**.

11. Click **Browse** at the end of the Output field. The Type Chooser dialog is displayed.

12. Select **Project Schema Files**, **singleString.xsd**, **singleString**, and then click **OK**.

13. Click OK. The OutboundDequeueRFH2 application and OutboundDequeueRFH2 Composite project appear in the design area.

Creating an Outbound Dequeue Adapter Service

Perform the following steps to create an adapter service that dequeues the message to a queue:

1. Drag and drop **MQ Adapter** from the Components window into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.

2. Click **Next**. The Service Name page is displayed.

3. Enter `OutboundDQ` in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.

4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.

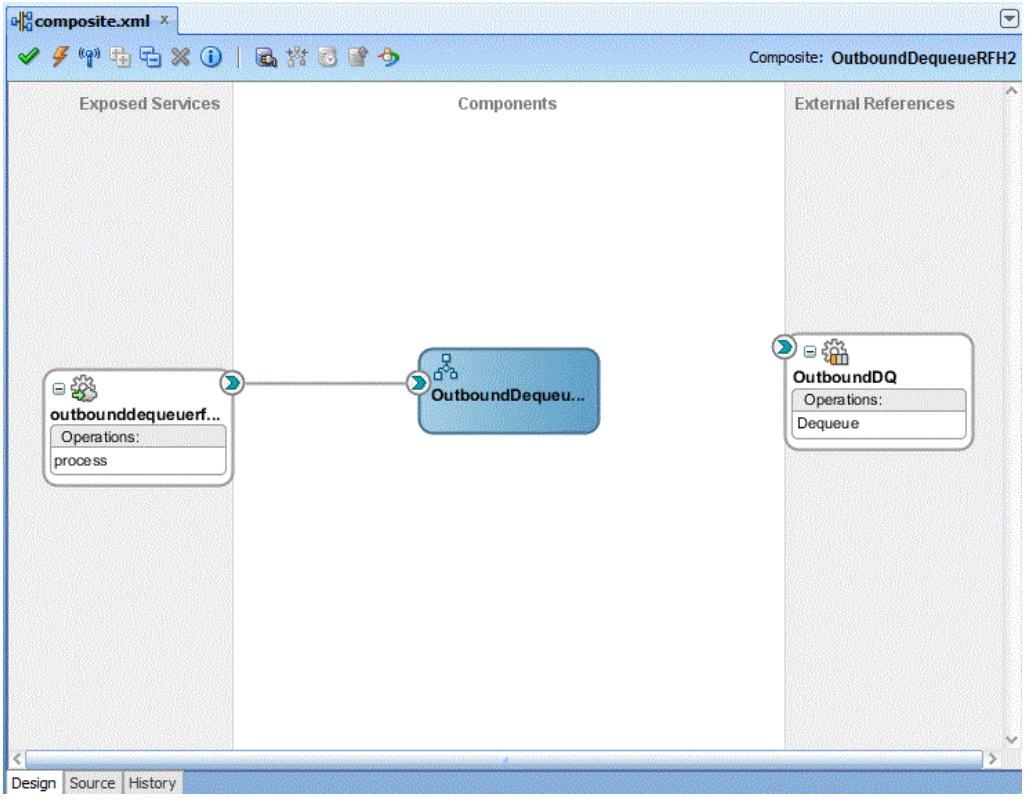
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.

6. Select **Get Message from MQ** and **Synchronous**, and click **Next**. The Get Message from MQ page is displayed.

7. Enter `queue1` in the **Queue Name** field and enter `10 seconds` in the **Wait Interval** field, and then click **Next**. The Messages page is displayed.

8. Select Native Form at Translation is not required (Schema is Opaque) and click **OK**.
9. Click **Next**. The Finish page is displayed.
10. Click **Finish**. You have now configured the inbound adapter service, and the composite.xml page is displayed, as shown in Figure 10-103.

Figure 10-103 The JDeveloper-composite.xml



11. Click **File**, **Save All**.

Wiring Services and Activities

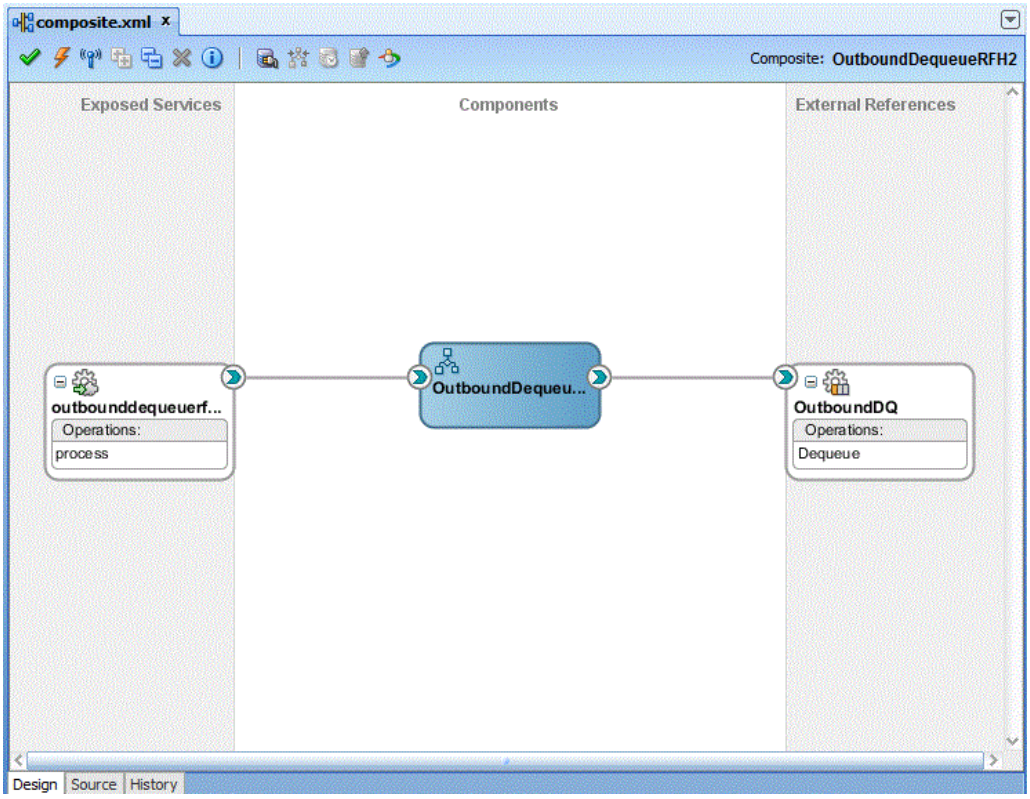
You must assemble or wire the three components that you have created: Client, BPEL process, and OutboundDQ adapter reference.

Perform the following steps to wire the components:

1. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the OutboundDequeueService in the External References area.
2. Double-click the OutboundDequeueRFH2 bpel process. The OutboundDequeueRFH2.bpel page is displayed.

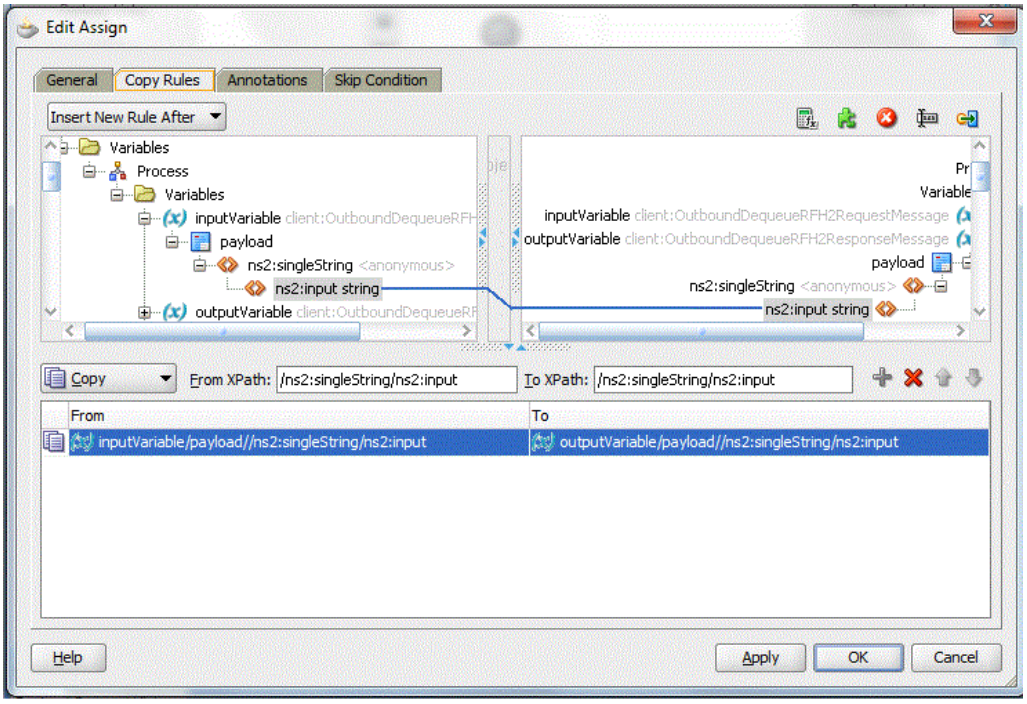
   The JDeveloper Composite.xml appears, as shown in Figure 10-106.

Figure 10-104 The JDeveloper - Composite.xml



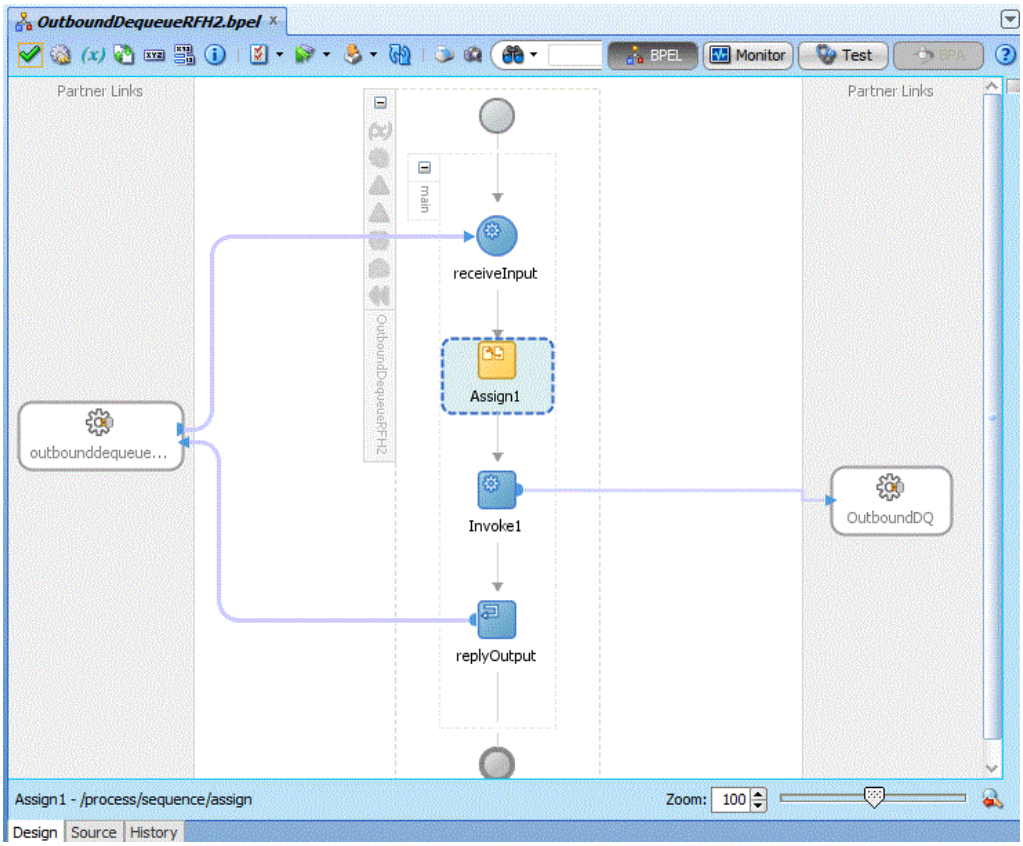
3. Click **File**, **Save All**.
4. Double-click **OutboundDequeueRFH2.bpel**. The **OutboundDequeueRFH2.bpel** page is displayed.
5. Drag and drop the **Invoke** and **Assign** activities in the order mentioned from the Components window to the Components area in between the `receiveInput` and `replyOutput` activities.
6. Drag and drop the **Invoke** activity to the OutboundDQ adapter reference. The Invoke dialog is displayed.
7. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The **Create Variable** dialog is displayed.
8. Accept the defaults, and click **OK**.
9. Repeat the same for the output variable and click OK.
10. Double-click the **Assign** activity. The **Assign** dialog is displayed.
11. Select the variables, and click the **Plus** icon.

Figure 10-105 The Assign Activity Dialog



12. .Click **OK** in the Assign dialog. The JDeveloper `BPELOutboundDequeue.bpel` page is displayed.

Figure 10-106 The JDeveloper BPELOutboundDequeue.bpel page



13. Create temporary variables to store the RFH2 header portions.

    a. Open the Source tab of the DequeueEnqueueRFH2.bpel page.

    b. Under the <variables> tab add the following new variables.

```
<variable name="RFH2.StructId" type="xsd:string"/>
<variable name="RFH2.Version" type="xsd:string"/>
<variable name="RFH2.Encoding" type="xsd:string"/>
<variable name="RFH2.CodedCharSetId" type="xsd:string"/>
<variable name="RFH2.Format" type="xsd:string"/>
<variable name="RFH2.Flags" type="xsd:string"/>
<variable name="RFH2.NameValueCCSID" type="xsd:string"/>
<variable name="RFH2.JMSFolder" type="xsd:string"/>
<variable name="RFH2.MCDFolder" type="xsd:string"/>
<variable name="RFH2.USRFolder" type="xsd:string"/>
<variable name="RFH2.USRFolder_2" type="xsd:string"/>
<variable name="RFH2.PSCFolder" type="xsd:string"/>
<variable name="RFH2extrafolder" type="xsd:string"/>
<variable name="RFH2_2.StructId" type="xsd:string"/>
<variable name="RFH2_2.Version" type="xsd:string"/>
<variable name="RFH2_2.Encoding" type="xsd:string"/>
<variable name="RFH2_2.CodedCharSetId" type="xsd:string"/>
<variable name="RFH2_2.Format" type="xsd:string"/>
<variable name="RFH2_2.Flags" type="xsd:string"/>
<variable name="RFH2_2.NameValueCCSID" type="xsd:string"/>
<variable name="RFH2_2.JMSFolder" type="xsd:string"/>
<variable name="RFH2_2.JMSFolder_2" type="xsd:string"/>
<variable name="RFH2_2.MCDFolder" type="xsd:string"/>
<variable name="RFH2_2.USRFolder" type="xsd:string"/>
<variable name="RFH2_2.USRFolder_2" type="xsd:string"/>
<variable name="RFH2_2.PSCFolder" type="xsd:string"/>
<variable name="TotalRFH2" type="xsd:string"/>
```

14. Configure the **Invoke** Activity to receive the RFH2 header properties from the outbound dequeue message.

    a. Open the Source tab of the DequeueEnqueueRFH2.bpel page.

    b. Under the <invoke> tag add the following entries.

```
<bpelx:outputProperty name="jca.mq.RFH2.StructId" variable="RFH2.StructId"/>
      <bpelx:outputProperty name="jca.mq.RFH2.Version"
variable="RFH2.Version"/>
      <bpelx:outputProperty name="jca.mq.RFH2.Encoding"
variable="RFH2.Encoding"/>
      <bpelx:outputProperty name="jca.mq.RFH2.CodedCharSetId"
variable="RFH2.CodedCharSetId"/>
      <bpelx:outputProperty name="jca.mq.RFH2.Format"
variable="RFH2.Format"/>
      <bpelx:outputProperty name="jca.mq.RFH2.Flags" variable="RFH2.Flags"/>
      <bpelx:outputProperty name="jca.mq.RFH2.NameValueCCSID"
variable="RFH2.NameValueCCSID"/>
      <bpelx:outputProperty name="jca.mq.RFH2.JMSFolder"
variable="RFH2.JMSFolder"/>
      <bpelx:outputProperty name="jca.mq.RFH2.MCDFolder"
variable="RFH2.MCDFolder"/>
      <bpelx:outputProperty name="jca.mq.RFH2.USRFolder"
variable="RFH2.USRFolder"/>
      <bpelx:outputProperty name="jca.mq.RFH2.USRFolder_2"
variable="RFH2.USRFolder_2"/>
      <bpelx:outputProperty name="jca.mq.RFH2.PSCFolder"
```

```
variable="RFH2.PSCFolder"/>
      <bpelx:outputProperty name="jca.mq.RFH2.mq_usr"
variable="RFH2extrafolder"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.StructId"
variable="RFH2_2.StructId"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.Version"
variable="RFH2_2.Version"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.Encoding"
variable="RFH2_2.Encoding"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.CodedCharSetId"
variable="RFH2_2.CodedCharSetId"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.Format"
variable="RFH2_2.Format"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.Flags"
variable="RFH2_2.Flags"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.NameValueCCSID"
variable="RFH2_2.NameValueCCSID"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.JMSFolder"
variable="RFH2_2.JMSFolder"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.JMSFolder_2"
variable="RFH2_2.JMSFolder_2"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.MCDFolder"
variable="RFH2_2.MCDFolder"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.USRFolder"
variable="RFH2_2.USRFolder"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.PSCFolder"
variable="RFH2_2.PSCFolder"/>
      <bpelx:outputProperty name="jca.mq.RFH2_2.USRFolder_2"
variable="RFH2_2.USRFolder_2"/>
      <bpelx:outputProperty name="jca.mq.RFH2.Total.Headers"
variable="TotalRFH2"/>
```

15. Click **File**, **Save All**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Deploying Oracle JCA Adapter Applications from .

You must also create an application server connection. For more information about creating an application server connection, see Creating an Application Server Connection for Oracle JCA Adapters.

The MQ message with the correct RFH2 headers must be put to the outbound dequeue queue. The sample is configured to obtain two RFH2 headers from the message (this can be changed by configuring the bpel process as required). The test MQ message should have the following:

- Payload: Any message (any format will do since Opaque is being employed)
- RFH2 header 1: Should contain 2 USR folders, 1 JMS folder, 1 PSC folder, 1 MCD folder and 1 mq_usr folder.
- RFH2 header 2: Should contain 2 USR folders, 2 JMS folders, 1 PSC folder and 1 MCD folder.

Deploy the sample and put the message to the outbound dequeue queue. Open the Fusion Middleware Control console and invoke the sample using the Test utility. Wait for a time and examine the instance audit trail.

Processing Messages as Attachment

This section demonstrates how to configure the inbound and outbound MQ Adapter to process messages as attachments.

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an ‘ application and a project for the use case, perform the following:

1. In JDeveloper, click File, select New and then select Application. The New Gallery dialog is displayed.
2. Expand the General node, and select the Applications category.
3. In the Items list, select SOA Application and click OK. The Create SOA Application Wizard is displayed.
4. In the Name Your Application screen, enter `MQAsAttachment` in the **Application Name** field, and then click Next. The Name Your Project screen is displayed.
5. In the **Project Name** field, enter MQAsAttachment and click Next. The Configure SOA Settings screen is displayed.
6. In the Composite Template list, select **Composite With BPEL**, and then click **Finish**. The Wizard displays the Create BPEL Process dialog.
7. Enter `MQAsAttachment` in the **Name** field, and select **Define Service Later** from the Template box.
8. Click OK. The `MQAsAttachment` application and the `MQAsAttachment` project appears in the design area.

Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message from a queue:

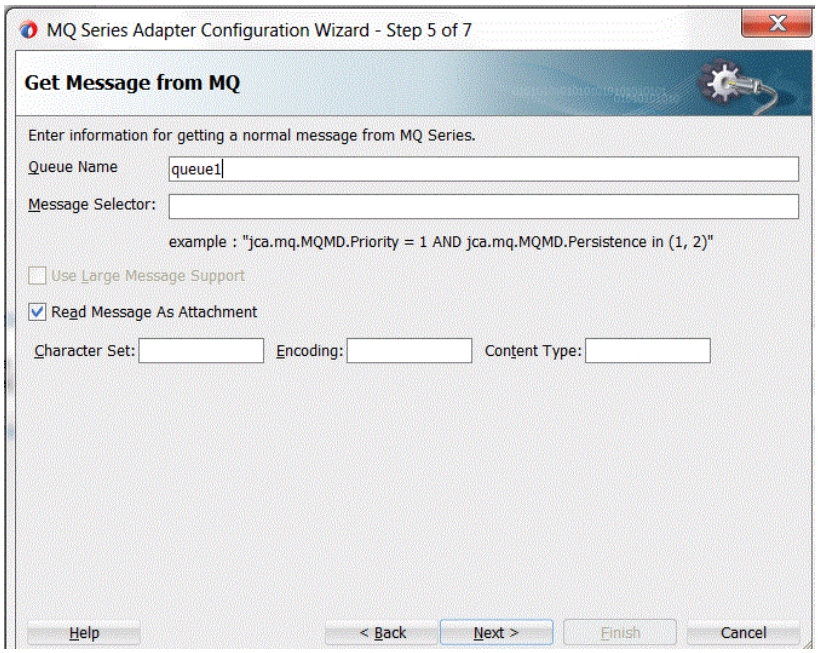
1. Drag and drop the MQ Adapter from the **Component Palette** into the Exposed Services swim lane. The **Adapter Configuration Wizard Welcome** page is displayed
2. Click **Next**. The **Service Name** page is displayed.
3. Enter `mqService` in the **Service Name** field, and click **Next**. The **MQ Series Connection** page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The **Adapter Interface** page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The **Operation Type** page is displayed.
6. Select **Get Message from MQ** (do not check **Synchronous**), and click **Next**. The **Get Message from MQ** page is displayed.

7. Enter `queue1` in the **Queue Name** field.

8. Select the **Read Message As Attachment** checkbox as shown in Figure 10-107 and click **Next**.

   Note: You must ignore **Character Set**, **Encoding**, and **Content Type** fields for this use case. Populate these fields with values only if you are using third-party applications that must read this attachment. The attachment in the present use case is finally consumed by an outbound Oracle MQ Adapter, rather than a third-party application, hence these values are not required.

Figure 10-107 Selecting the Read Message as Attachment Checkbox



9. Click **Finish**. You have now configured the inbound adapter service, and the `composite.xml` page is displayed with an inbound adapter added.

Creating an Outbound Adapter Service

To create an outbound adapter service:

1. Drag and drop an MQ Adapter from the **Components** into the External References swim lane. The Wizard displays the **Adapter Configuration Wizard Welcome** page.

2. Click Next. The Wizard displays the **Reference Name** page.

3. Enter `mqReference` in the **Service Name** field, and click **Next**. The **MQ Series Connection** page is displayed.

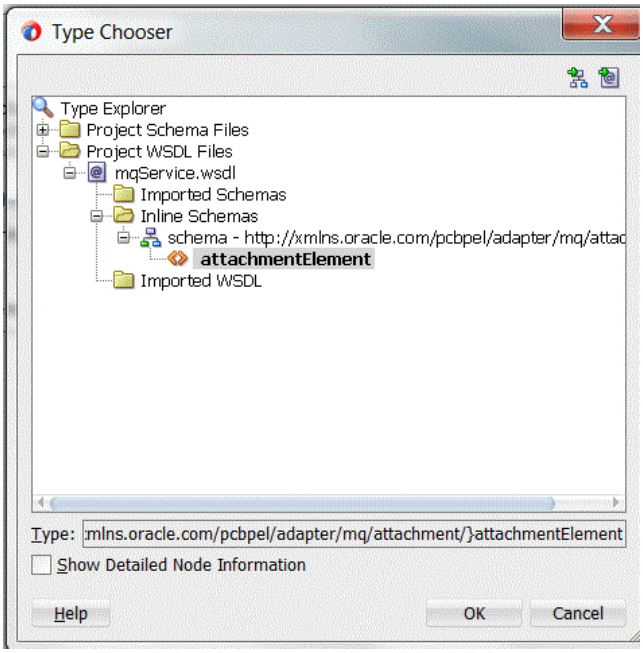
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The **Adapter Interface** page is displayed

5. Select **Define from operation and schema (specified later)**, and click **Next**. The **Operation Type** page is displayed.

6. Select **Put Message into MQ** and click **Next**. The **Put Message into MQ** page is displayed.

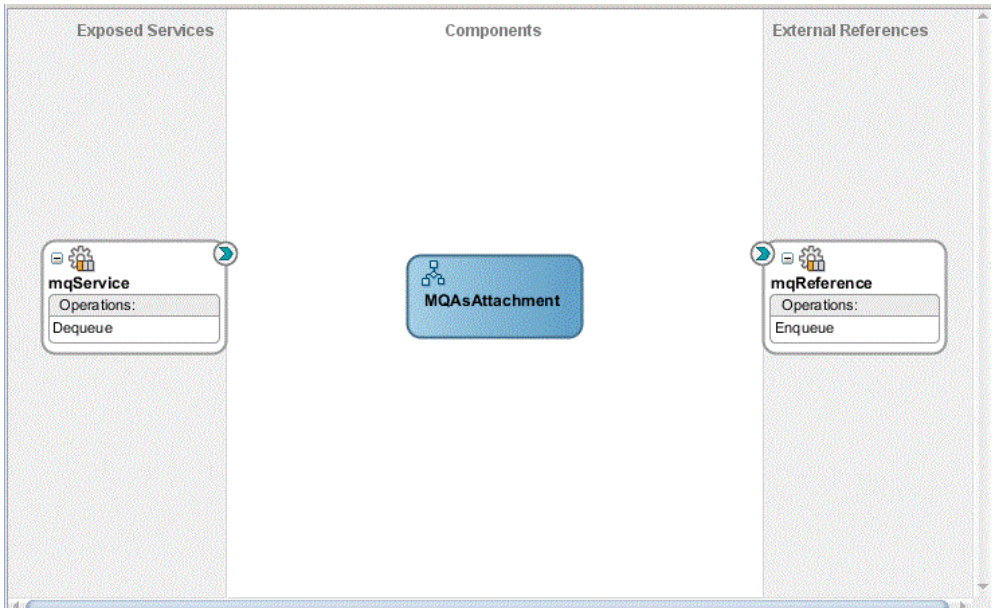
7. Enter `queue2` in the **Queue Name** field and click **Next**. The **Advanced Options** page is displayed.
8. Accept the defaults and click **Next**. The **Messages** page is displayed.
9. Click Browse For Schema File that appears at the end of the URL field. The Type Chooser dialog is displayed.
10. Click Project WSDL Files, mqService.wsdl, Inline Schemas, and attachmentElement, as shown in Figure 10-108.

Figure 10-108 Selecting attachmentElement



11. Select **OK**. The Wizard populates the URL field in the **Messages** page with `AttachmentIn.wsdl`.
12. Click **Next**. The **Finish** page is displayed.
13. Select **Finish**. You have now configured the outbound adapter service, and the `composite.xml` page is displayed, as shown in Figure 10-109.

Figure 10-109 Composite.xml after Outbound Adapter Service Configured



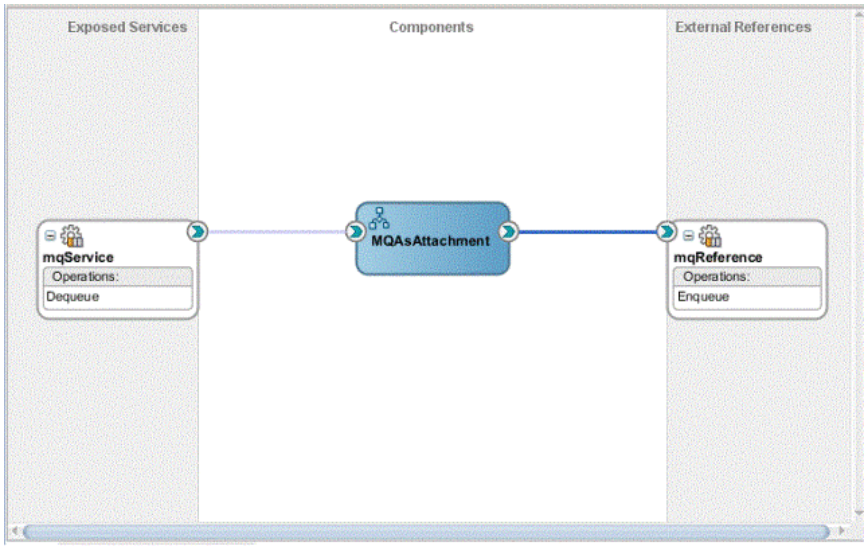
Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the mqService in the Exposed Services area to the drop zone that appears as a green triangle in the MQAsAttachment in the Components area.
2. Drag the small triangle in the MQAsAttachment in the Components area to the drop zone that appears as a green triangle in `mqReference` in the External References area.

   The JDeveloper Composite.xml appears as shown in .Figure 10-110.

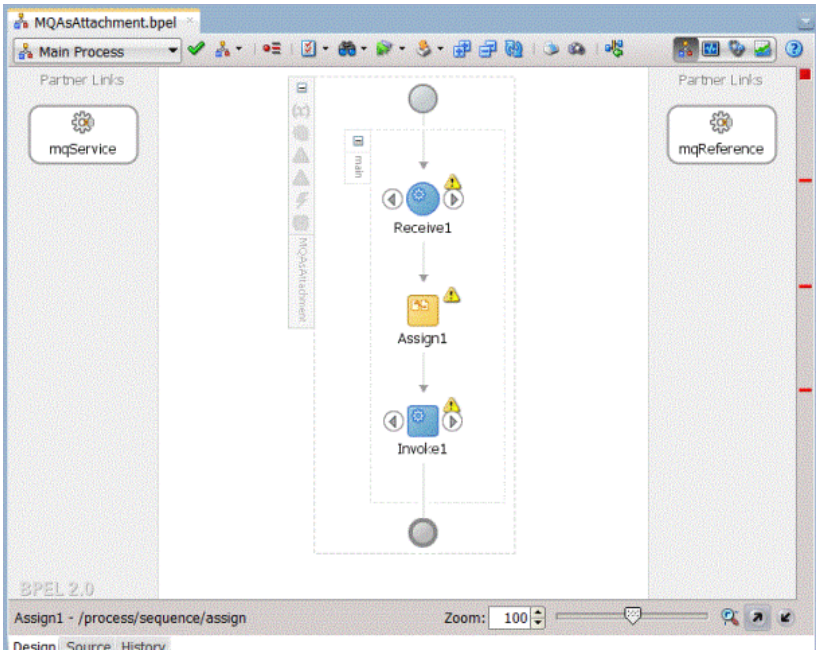
Figure 10-110 The composite.xml with Activities and Services Wired



3. Click **File**, **Save All**.
4. Double-click `MQAsAttachment`. The `MQAsAttachment.bpel` page is displayed.
5. Drag and drop the Receive, Assign, and Invoke activities in the order mentioned from the Component Palette to the Components area.

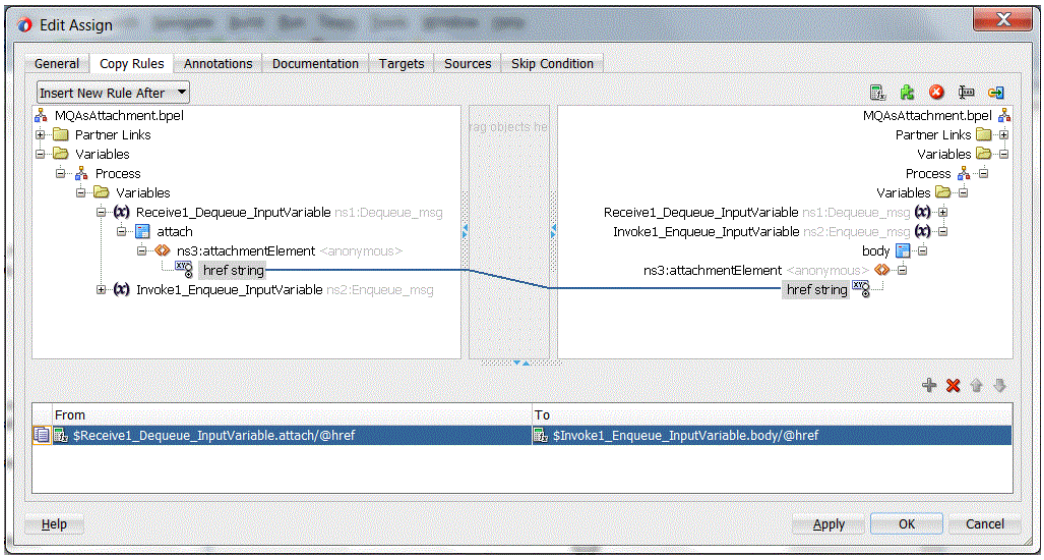
   The JDeveloper **MQAsAttachment.bpel** page is displayed, as shown inFigure 10-111 below.

Figure 10-111 The JDeveloper MQAsAttachment.bpel page



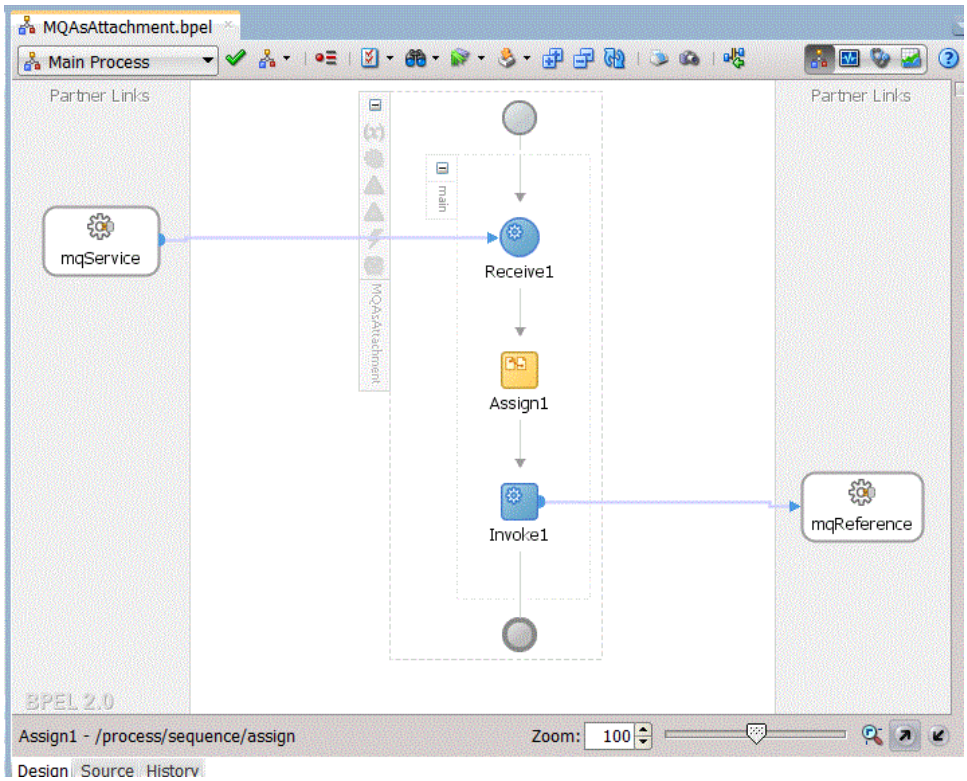
6. Drag and drop the **Receive** activity to `mqService`. The **Receive** dialog is displayed.
7. Click the **Auto Create Variable** icon that appears at the end of the **Variable** field. The Create Variable dialog is displayed.
8. Accept the defaults, and click **OK**.
9. Check the **Create Instance** box, and click **OK**.
10. Drag and drop the **Invoke** activity to `mqReference`. The **Invoke** dialog is displayed.
11. Click the **Automatically Create Input Variable** icon that appears at the end of the **Input Variable** field.
12. Accept the defaults, and click **OK**. The **Invoke** dialog is displayed.
13. Click **OK**.
14. Double-click the **Assign** activity. The **Assign** dialog is displayed.
15. Select the variables, as shown in Figure 10-112, then select the **Plus** icon.

Figure 10-112 The Assign Activity Dialog



16. Click OK in the **Assign** dialog. The JDeveloper **MQAsAttachment.bpel** page is displayed, as shown in Figure 10-113.

Figure 10-113 The MQAsAttachment.bpel page



17. Click **File**, Save **All**.

Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Creating an Application Server Connection for Oracle JCA Adapters.
2. Deploy the application. For more information, see Deploying Oracle JCA Adapter Applications from .

11

Oracle JCA Adapter for UMS

This chapter describes how to use the Oracle User Messaging Service Adapter, which provides a JCA Adapter that wraps the Oracle User Message Service (UMS), an Oracle Fusion Middleware Component that enables communication between users and applications. The chapter also provides information on UMS Adapter concepts, features, configuration, and error handling.

This chapter includes the following sections:

UMS and UMS Adapter Concepts

This section includes the following topics:

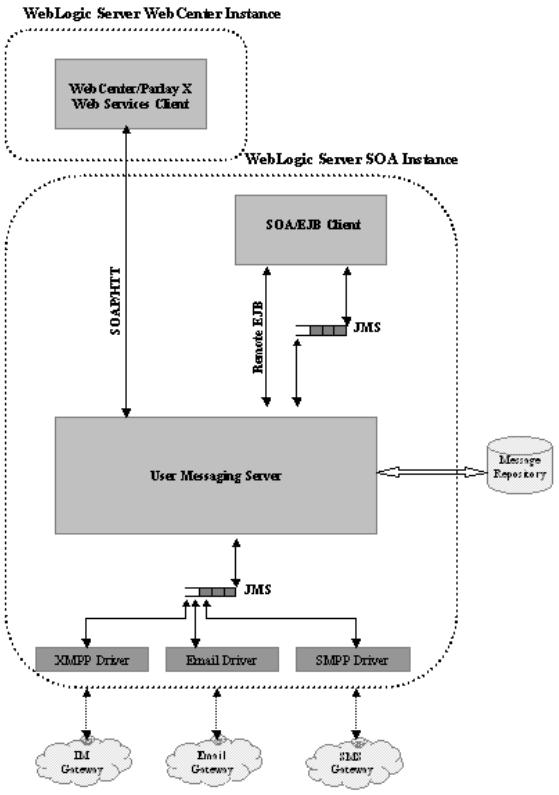
User Messaging Service

The User Messaging Service is an Oracle Fusion Middleware Component that enables communication between users and application. It consists of the following:

- UMS Server: The UMS Server orchestrates message flows between applications and users. The server routes outbound messages from a client application to the appropriate driver, and routes inbound messages to the correct client application. The server also maintains a repository of previously sent messages in a persistent store, and correlates delivery status information with previously sent messages.
- UMS Drivers: UMS Drivers connect UMS to the messaging gateways, adapting content to the various protocols supported by UMS. Drivers can be deployed or undeployed independently of one another depending on what messaging channels are available in a given installation.
- UMS client applications: UMS client applications implement the business logic of sending and receiving messages. A UMS client application might be a SOA application that sends messages as one step of a BPEL workflow, or a WebCenter Spaces application that can send messages from a web interface.

See Figure 11-1 for more context on the UMS Server.

Figure 11-1 The User Messaging Server in Context



UMS supports various messaging channels such as Email, SMS, Instant Messaging, and Voice. UMS provides a messaging proxy between the Oracle BPEL or Mediator products and the external world. The User Messaging Service provides two-way messaging: Inbound and Outbound messaging, and provides robust message delivery, including delivering delivery status, and message resend through Enterprise Messages. UMS also provides support for failover address. In summary, the UMS provides a scalable, highly available solution to communication between users and applications.

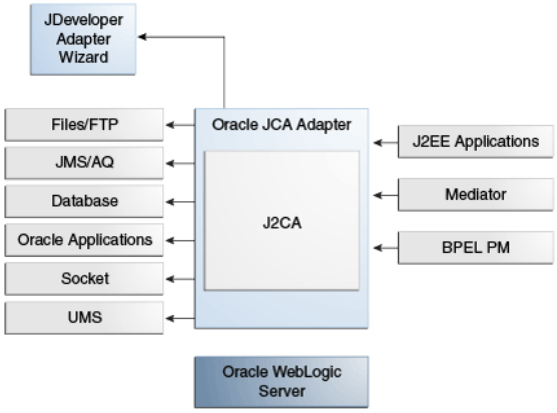
For additional information on the User Messaging Service, see the *Oracle® Fusion Middleware Administering Oracle User Messaging Service*.

Oracle UMS Adapter

The User Messaging Service Adapter implements the Java Enterprise Edition Connector Architecture (JCA) version 1.5. The UMS Adapter, in effect, wraps the User Messaging Service, thus enabling communication over messaging channels that include Email.

The UMS Adapter is part of the overall Adapter architecture.J2EE Applications, Mediator and BPEL processes communicate with the Oracle WebLogic Server. See the following diagram.

Figure 11-2 The UMS Adapter as Part of the Adapter Architecture



Oracle UMS Adapter Features

The UMS Adapter includes the following Outbound features

- Support of Email Messaging, IM and SMS Channels.
- Use of Message Filters-the UMS Adapter enables you to filter IM and SMS messages using Content, Recipient, Sender and Subject.
- Sending Email messages with Subject and Body and one or more attachments. The user can also send out IM, SMS messages.
- Receive message delivery status information from UMS.
- Translation Support for the message body.
- SSL/TLS security for the outbound SMTP server (this availability is provided through the Java Messaging Service).
- XA or global transaction support for outbound transactions.

The UMS Adapter provides the following Inbound features:

- Support for Email, IM and SMS Messaging channels.
- Use of Message Filters-the UMS Adapter enables you to filter email messages in two ways:

  1. Establishing Message filters through the Adapter Configuration Wizard Messages Filter Screen. These include Message filters, Blacklist and Whitelist filters. Message Filters provide the ability to filter incoming messages based on the Email To address, From address, CC address, Subject, and Mail Headers. Mails can similarly be ignored using this filtering. Note: there is no filtering available on the BCC address.

  2. Writing and packaging a Java Callout, and providing the name of the callout through the Adapter Configuration Wizard. Refer to Custom Java Callout for more information.

- Reception of messages with Subject, Body and one or more attachments along with internet mail and mime headers.

- A Polling/Listener interface. The UMS Adapter polls mailboxes for incoming email on various schedules you establish, which can be both sequential and parallel.
- Translation support for the message body.
- XA or global transaction support for inbound(applicable to both polling and listener mode) s. Note that if the UMS Adapter flow is Outbound, and even if the UMS Adapter is set for XA Transactions to False, if the flow is part of an-already initiated Transaction from BPEL or from an Inbound Adapter, then the same Outbound UMS Adapter behaves as if it were performing XA transactions.
- IMAP/POP3 servers with SSL (available through the UMS Server)

Note:

You cannot set and get headers with Keyword headers when you use the MS-2010 Mail server with the UMS Adapter

---

UMS Adapter Message Concepts

The UMS Adapter enables you to provide different message formats.

For many of your email use cases, you might not want to specify a schema as the message payload could be plain text and you want it received as is. In that case, you can then select the `Message is String type` checkbox on the Messages Screen.

XSD files are required for translation of messages. If you want to define a new schema or convert an existing data type definition (DTD) or COBOL Copybook you must select `Define Schema for Native Format` to supply an XSD file.

Selecting `Define Schema for Native Format` starts the Native Format Builder wizard. This wizard guides you through the creation of a native schema file from file formats that include comma-separated value (CSV), fixed-length, DTD, and COBOL Copybook.

After the native schema file is created, the Messages page is displayed, with the Schema File URL and Schema Element fields filled in. For more information, see Creating Native Schema Files with the Native Format Builder Wizard.

Unlike other adapters, the UMS Adapter uses a predefined Message Schema to represent the message it uses.

An example of the Message Schema that the UMS Adapter uses follows.

```
<?xml version= "1.0" encoding= "UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://platform.integration.oracle/blocks
                    /adapter/fw/metadata/Inbound_UMS"
            targetNamespace="http://platform.integration.oracle/
                   blocks/adapter/              fw/metadata/Inbound_UMS"
            xmlns:imp1="http://xmlns.oracle.com/pcbpel/adapter/opaque/"
            elementFormDefault="qualified"
                attributeFormDefault="unqualified">
<xsd:import namespace="http://xmlns.oracle.com/pcbpel/adapter/opaque/"
schemaLocation="opaque.xsd"/>
    <xsd:complexType name="MessageType">
        <xsd:sequence>
            <xsd:element ref="imp1:opaqueElement"/>
            <xsd:element name="attachment"
            type="AttachmentType" minOccurs="0"
```

```
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="AttachmentType">
        <xsd:attribute name="href" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="ResponseType">
        <xsd:sequence>
            <xsd:element name="MessageId"
             type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="message" type="MessageType"/>
    <xsd:element name="response" type="ResponseType"/>
</xsd:schema>Status OpenFixedClosed
```

You define the schema according to your translation requirement through the UMS Adapter Configuration Wizard Message screen, and as defined for the message body content). The UMS Adapter imports the xsd you specify into the message schema used by the UMS Adapter.

For example, see the schema snippet in the following example, where the user-defined schema `singleString.xsd` is imported through the UMS Adapter Configuration Wizard and refers to the element `singleString`, which is defined under `singleString.xsd` through the UMS Adapter Configuration Wizard.

Example - Schema Snippet for SingleString.xsd

```
<?xml version= '1.0' encoding= 'UTF-8' ?>
<xsd:schema attributeFormDefault="unqualified"
            elementFormDefault="qualified"
            targetNamespace="http://xmlns.oracle.com
                /singleString"
            xmlns="http://xmlns.oracle.com/singleString"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            nxsd:encoding="US-ASCII" nxsd:
                useArrayIdentifiers="true"
            nxsd:stream="chars" nxsd:version="NXSD">
    <xsd:complexType name="singleString">
        <xsd:sequence>
            <xsd:element name="input" type="xsd:string"
                         nxsd:style="terminated"
                         nxsd:terminatedBy=";"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="payload" type="singleString"/>
</xsd:schema
```

Custom Java Callout

On the Java Callout Screen you can specify a custom Java class with custom logic that can be invoked before the Email, SMS, or IM message is processed by the UMS Adapter.

Use Cases for Custom Java Callout

One simple use case of the Custom Java Callout is to match the sender address with an address in the LDAP or that resides in a staging area and which has been recorded earlier. To provide this use case, you must implement the interface `oracle.tip.pc.services.translation.util.ICustomCallout`.

This interface defines a single method execute with a return value of `boolean`. Depending on the return value, the message is either processed or rejected:

```
public Inteface CustomCallout{
  public boolean execute (Message message) throws exception;
}
```

Where `Message` is the `Message` class from the UMS Session Description Protocol (SDP) Java API, a well-defined Java API provided by the UMS Server, which can be found at `$MW_HOME./oracle_common/modules/oracle.sdp.client_12.1.2/messaging-api.jar`. The Boolean value returned indicates whether to accept and process the message or to reject the message.

```
package oracle.adapter.custom;

import java.io.File;
import oracle.sdp.messaging.Message;
import oracle.tip.pc.services.translation.
             util.ICustomCallout;

public class UMSAdapter_CustomCall implements          ICustomCallout{

        @Override
        public boolean execute(Message message)              throws exception {
        String emailFromAddress = message.
        getSenders()[0].getValue();

String fileName = "/tmp/OracleStore/staging/".
concat(emailFromAddress).concat(".usr");
                File file = new File(fileName);
                if(file.exists()) {
                        return true;
                }
                return false;
        }
}
```

In another use case, as provided through the sample code, a user with the email id of scott.tiger@example.com registers through an internet store web site. The user would be recorded under a staging area by using a file name scott.tiger@example.com.

An email is sent to the user directly to his email id to reply to the sent email.

If the user replies to the email, the UMS Adapter picks the email. You can subsequently use a Java Callout to check the user file under the staging area and ensure that the user is registered through the web site.

Using the Custom Callout Facility

To use the Custom Callout facility, you must

1. Indicate the name of the class on the Java Callout Screen in the UMS Adapter Configuration Wizard.
2. Bundle the class and other required custom classes as a jar file.
3. Place the jar file under your Composite Application, under the `SCA-INF/lib` directory.

4. Ensure that the compiled Java class (.class file) is directly placed under the Composite Project Folder `\SCA-INF\classes`:

Transaction Support

The UMS Adapter, by default, uses an XA Transaction with an Inbound Scenario for both polling and listener mode. For this purpose, the Adapter configures a connection factory instance with the property `XATransaction` set to true.A default JNDI instance is available, by default, which is configured to use XA. The name of this default JNDI instance is `eis/ums/UMSAdapterInbound`.For outbound scenarios, the UMS Adapter leverages the XA support from the UMS. UMS will participate in a global transaction if one is already started, for example through BPELYou can choose to define your own JNDI instances and use them, but you should keep in mind this discussion.

Inbound Error Handling

The UMS Adapter uses the default rejection handling mechanism on the Inbound side of the Adapter for rejecting bad messages. For example, any translation-related errors result in message rejection. Refer to Creating Fault Policies in this Guide for more information on fault policies and adapters.

Under retriable error conditions, and when you specify retry-related endpoint properties, the UMS Adapter tries to re-publish the Inbound message for the configured number of retries before rejecting the message. Transactions are then set for rollback under XA, per the previous JNDI and XA discussion.

Outbound Error Handling

The UMS Adapter throws an exception for transient (recoverable) error conditions such as connection errors. For retriable errors, you can use a retry policy supported by the Adapter framework; to do this, you can set the binding property `jca.retry.count` to a retry count you want. Again, as with other Adapters, if you do not set the property, the retry is carried through according to the fault policy.

You can define non-retriable connection errors for outbound transactions through a fault policy. The maximum number of re-connection attempts can be defined through `fault-policy.xml`.

Adapters translate data from Native representation to standard XML format and back based on the metadata captured by your work at design time, through the Adapter Configuration Wizard. A translation error is thrown when there is an exception thrown by carrying out the translation, and a corresponding binding fault is also thrown.

You can set Endpoint properties related to Oubound retriable errors, as shown in Table 11-1

Table 11-1 UMS Adapter Outbound Endpoint Properties

Property	Description
`jca.retry.count`	Indicates the maximum number of retries before throwing retriable error conditions back to the invoking service engine.
`jca.retry.interval`	Indicates the time interval between retries, measured in seconds.

Table 11-1 (Cont.) UMS Adapter Outbound Endpoint Properties

Property	Description
jca.retry.backoff	Indicates the retry interval growth factor, measured in positive integers.
`jca.retry.maxInterval`	Indicates the maximum value of retry interval; that is a cap if the value is greater than 1.
`jca.retry.maxPeriod`	Indicates the maximum total retry period. Retries do not occur longer than the value specified in the parameter.

Retry Mechanism for Failed Outgoing Notifications with Status Reporting

The UMS Adapter makes use of the existing infrastructure provided by UMS for retrying failed outgoing exceptions. Currently, the UMS Server supports the viewing of failed notifications from Fusion Middleware Control, in addition to the resending of messages.

Inbound Receive Notification in a Cluster (Through Polling or Through a Listener)

The UMS API supports an environment where the UMS Server and its clients are deployed in a cluster environment.The UMS Adapter also supports high availability in an active-active setup.

The configuration details for UMS Adapter to work properly in a clustered environment follows. UMS deployment in a cluster must look the same. On a node in the cluster, you have the UMS adapter, the UMS server EAR and all driver EARs. They must all co-exist on the node.Note that in, for example, the Adapter Configuration Wizard when you create a cluster, you do target the UMS adapter and all UMS to the cluster. In a later step, in the Configuration Wizard you list all servers that are to be part of the cluster.

All UMS Adapter activations of the *same* composite application use the *same* unique `ApplicationName` configuration parameters. The UMS Adapter synthesizes the application name parameter from the Inbox address on which the specific endpoint is to listen.

This synthesis enables *all* activations of a specific composite in a cluster to share all configuration and artifacts such as Access Points and Message Filters

The `ApplicationInstanceName` configuration parameter is synthesized automatically through the UMS API implementation and the UMS Adapter depends on that synthesis.

Additionally, the UMS Adapter API implementation guarantees that in a cluster environment no two applications will receive the same message.

The UMS Adapter also supports active fail-over of an Inbound service that is active-passive in a clustered environment. You can enable this active fail-over for UMS Adapter support through a JCA service binding property (`composite.xml`)singleton, set to `true`.

Listening and Polling work the same way using a UMS adapter in a clustered environment as they do in a non-clustered environment.

UMS Adapter Properties and Mime Type Configuration

There are several properties associated with the UMS Adapter that you can use to provide additional configuration. Some of these are already set for you when you use the UMS Adapter Configuration Adapter Wizard. All applicable Internet Mail Headers and Mime headers can be configured through normalized message properties/headers.

Table 11-2 lists the Activation Spec properties applicable to the UMS Adapter.

Table 11-2 UMS Adapter Activation Spec Properties

Property Name	Description
`JavaCalloutImpl`	Name of the Java class that defines custom logic for a message filtering or any other check. This class is a concrete implementation of the ICustomCallout interface.
`ConsumeMode`	Specifies how the adapter will receive messages from UMS. Set to poller for polling mode Or set to listener for listener mode.
`To`	Address from which to receive incoming messages. One or more comma separated IM, SMS, or Email addresses for the IM, SMS, or Email delivery type.
`Delivery Type`	Email support is provided for receiving and sending outgoing messages.
`PollingInterval`	Polling interval in seconds for poller consume mode.
`MessageFilters`	Specify one or more message filters. A single filter would comprise of a Java Pattern String to match the incoming message against, along with the field type and the action (Accept or reject) to be taken.

Table 11-3 provides a list of interaction specification properties available.

Table 11-3 UMS Adapter Interaction Specification Properties

Property Name	Description
`Delivery Type`	Email, IM and SMS support for receive and sending outgoing messages.
`Subject`	Subject of Outgoing Message.
`From`	Sender addresses of outgoing message.
`To`	One or more recipient addresses.
`Reply-to`	Reply-To address.
`CC`	One or more cc addresses for email delivery.
`Bcc`	One or more Bcc addresses for email delivery.
`SendEmailAsAttachment`	`True`, to send email as an attachment.

The UMS Adapter exposes all applicable internet messages headers and Mime message headers and mime part headers (within a multipart construct.)

Mime headers are applicable only for the first body part of the message that is the UMS Adapter payload.

Mime headers for attachments are stored along with the attachment as normalized message properties that can be manipulated from within a BPEL process.

Table 11-4 describes all the applicable headers defined by the internet message format along with mapping and corresponding adapter header.

Table 11-4 Message Headers

Header Field Name	Minimum Occurrence	Maximum Occurrence
`Return-path`	0	1
`Received`	0	unlimited
`Resent-Date`	0	unlimited
`Resent-From`	0	unlimited
`Resent-Sender`	0	1
`Resent-To`	0	unlimited
`Resent-Cc`	0	unlimited
`Resent-Bcc`	0	unlimited
`Resent-Message-ID`	0	unlimited
`Date`	1	unlimited
`From`	1	unlimited
`Sender`	0	1
`Reply-to`	0	1
`To`	0	1
Cc	0	1
Bcc	0	1
`Message-ID`	0	1
`In-Reply-To`	0	1
`References`	0	1
`Subject`	0	1
`Comments`	0	unlimited
`Keywords`	0	unlimited

Note:

You can often use the BPEL Invoke activity's property tab to select jca.properties to set a value from the Invoke activity. However, you cannot set `jca.ums.message-id`, as these message IDs are auto-generated.

Table 11-5 describes all applicable Mime message headers.

For the outbound UMS Adapter, you can use the property `JCA.UMS.MSG.CONTENT-TYPE`, which can be used for specifying encoding. But if you do not set the value, the outbound UMS adapter uses UTF-8 encoding for email by default.

Table 11-5 Mime-Part Message Headers

Header Field Name	Mapped Adapter Header Field Name	Notes
Content-Type	`jca.ums.part.content-type`	-
Content-Transfer-Encoding	`jca.ums.part.content-transfer-encoding`	-
Content-ID	`jca.ums.part.content-id`	-
Content-Description	`jca.ums.part.content-description`	-
Content-Disposition	`jca.ums.part.content-disposition`	-
Content-Language	`jca.ums.msg.content-language`	-
Mime-Extension-field	`jca.ums.part.mime-extension-headers`	Any other mime header field hat begins with the string "Content-". You can add more than one header as `Content-* : value CRLF Content*- : value` `CRLF - \r\n`

Proprietary Headers

.

The UMS Adapter enables you to add any proprietary headers. Table 11-6 shows the information required for doing so.

Table 11-6 Proprietary Headers

Header Name	Notes
`jca.ums.msg.proprietary-headers`	More than one proprietary header can be added in the following format: Header Name : value CRLF Header Name : value (Header Name – this should be similar to ums.adapter.xxxxx CRLF - \r\n)

Email Attachments

The UMS Message XML can contain a list of Attachment elements that have an `href` attribute. The Attachment Manager stores other mime details associated with the attachment as MimeType, Content ID along with a stream object (which is the attachment content).

Currently, the UMS Adapter supports both inbound and outbound attachments.

See the following example for a sample XML Message with attachment element.

Example - Sample XML Message with Attachment Element

```
<Receive1_ReceiveEmail_InputVariable>
<part name="body" >
  <Email>
    <payload>This is a test mail.-Sagar</payload>
      <attachment href=
          "0DF86C104BF511EoAF5977BAA7C7CFD9"/>
  </Email>
  </part>
</Receive1_ReceiveEmail_InputVariable>
```

On the inbound side, the UMS Adapter sets all mime details before passing the attachment to the Attachment Manager. The Attachment Manager is a fabric attachment manager internal component, responsible for storing attachment contents and MIME headers to the database. This functionality enables you to pass attachments as links/href ids between different SOA components.

Mail Attachment Handling

The UMS Adapter uses the Fabric Attachment Manager to store and retrieve attachments.

The UMS Message schema defines Attachment element with a href attribute:

Example - UMS Message Schema Defines Attachment Element with href Element

```
<Receive1_ReceiveEmail_InputVariable>
<part name="body" >
  <Email>
    <payload>This is a test mail.-Sagar</payload>
      <attachment href=
           "0DF86C104BF511EoAF5977BAA7C7CFD9"/>
  </Email>
  </part>
</Receive1_ReceiveEmail_InputVariable>
```

The UMS Message XML can have list of attachment elements with a href attribute. The The Attachment manager stores other mime details associated with attachment as MimeType; Content ID and others, in addition to a stream object (attachment content). Sample XML Message with attachment element

On the inbound side, the UMS adapter sets all mime details before passing attachment to Attachment Manager, while on the outbound, the UMS Adapter extracts any mime details received along with attachment object and uses them while creating the outgoing SDP message notification.

The Fabric Attachment Manager updates the href attribute with a key after storing the attachment to the database. You can use this key later to retrieve attachment content.

The sample Normalized Message payload map has an XML structure having attachment element, which is passed from UMS Adapter to BPEL/Mediator service engine.

A sample XML Message with attachment element follows:

```
<Receive1_ReceiveEmail_InputVariable>
<part name="body" >
  <Email>
    <payload>This is a test mail</payload>
    <attachment href=
         "0DF86C104BF511EoAF5977BAA7C7CFD9"/>
  </Email>
  </part>
</Receive1_ReceiveEmail_InputVariable>
```

Retrieving Mime Information Associated with an Attachment in BPEL

The following example shows how attachments can be retrieved in BPEL. First, obtain the Mime Information associated with the attachment.

Example - Obtaining the Mime Information Associated with an Attachment

```
<assign name"Assign1">
<copy>
   <from exporession="oraReadBnaryFromFileWithMimeHeaders
     ('/home/testuser/oracle_  sig_logo.gif','one',
       'image/gif', '', '7bit',
             'oracle_logo_gif_file', 'en/ja')"/>
   <to variable="invoke="1_SendNotification_InputVariable"
      part="body"
    query='/ns3:message/ns3:attachment[1]"/>
</copy>
<bpelx:InsertAfter>
    <bpelx:from variable=Invoke1_SendNotification
                          _InputVariable"
            part="body" query=/ns3:message:
                      /ns3:attachment"/>
<bpelx:to variable=Invoke1_SendNotification_InputVariable"
            part="body"
            part="body" query=/ns3:message:/ns3:
                          attachment"/>

</bpelx:insertAfter>
<copy>
    <from expression="ora.
                readBinaryFromFileWithMimeHeaders
```

```
        {'/home/testuser/
          install-mq-7.0.txt',",",","," '
                           install_mq_file',''}"/>
     <to variable="Invoke1_SendNotification_InputVariable"
          part="body"
          query="/ns3:message/ns3:attachment[2]"/>
</copy>
```

Setting Mime Information for Multiple Attachments in BPEL

The following snippet shows how you can set Mime Information for multiple attachments in BPEL.

```
<assign name="Assign1">
<copy>
<from exporession="oraReadBnaryFromFileWithMimeHeaders
       ('/home/testuser/oracle_sig_logo.gif','one',
          'image/gif', '', '7bit',
            'oracle_logo_gif_file', 'en/ja')"/>
<to variable="invoke="1_SendNotification_InputVariable"
         part="body"
    query='/ns3:message/ns3:attachment[1]"/>

</copy>
<bpelx:InsertAfter>
  <bpelx:from variable=Invoke1_SendNotification_InputVariable"
        part="body" query=/ns3:message:/ns3:attachment"/>
<bpelx:to variable=Invoke1_SendNotification_InputVariable"
           part="body"
      part="body" query=/ns3:message:/ns3:attachment"/>

</bpelx:insertAfter>
   <from exporession=
         "oraReadBnaryFromFileWithMimeHeaders('/home/             testuser/oracle_
sig_logo.gif','one',
             'image/gif', '', '7bit',    'oracle_logo_gif_file', 'en/ja')"/>
<copy>
   <to variable="invoke="1_SendNotification_InputVariable"
          part="body"
    query='/ns3:message/ns3:attachment[2]"/>
</copy>
```

UMS Adapter Inbound and Outbound Operations

Operations you configure using the UMS Adapter Configuration Wizard include the following Inbound and Outbound Operations:

- `Inbound Receive Notification`
- `Outbound Send Notification`
- `Outbound Send Notification (Message ID as Reply)`
- `Outbound Get Message Status`- (An outbound synchronous request reply where you provide message id and optional recipient information to receive the message delivery status information from the UMS.)

Oracle UMS Adapter Inbound ReceiveNotification Concepts

In this scenario, the UMS Adapter registers an access point on the UMS Server to consume and process incoming notifications. This section provides an overview of

different configurations and concepts associated with Oracle UMS Adapter Inbound Receive Notification.

Oracle UMS Outbound Send Notification Concepts

In this scenario, the UMS Adapter processes outgoing send notifications. This section provides an overview of different configurations and concepts associated with the Oracle UMS Adapter Outbound send notification, of which there are two types, the normal send and the send with Receive Message id as reply request.

Receive Message id as reply request

A type of outbound synchronous request is the `Receive Message id as reply request`. It is a variation of the one way invoke; however, with `Receive Message id as reply` request, a unique message ID is replied back from the UMS Server. You can use this type of request if you want to get this message id to check message delivery status information. UMS stores all delivery-related status information received from messaging gateways. The Message id can be used to retrieve this status information, using Fusion Middleware Control.

If you choose to use this type of request, you select the `Receive Message Id as a reply request` checkbox when selecting an Operation.

Configuring the Oracle UMS Adapter

You configure the User Message Service using the Fusion Middleware Control Console and the Email Adapter Configuration Wizard in JDeveloper when you configure the UMS Adapter.

To configure the User Messaging service drivers, use the Fusion Middleware Control Console. Specifically, to use the Email messaging channel, you must configure specific properties. The following section provides information on configuring the Email driver for the UMS Adapter.

For details on configuring SMPP, XMPP and VoiceXML driver, see "Configuring Oracle User Messaging Service" in the Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite.

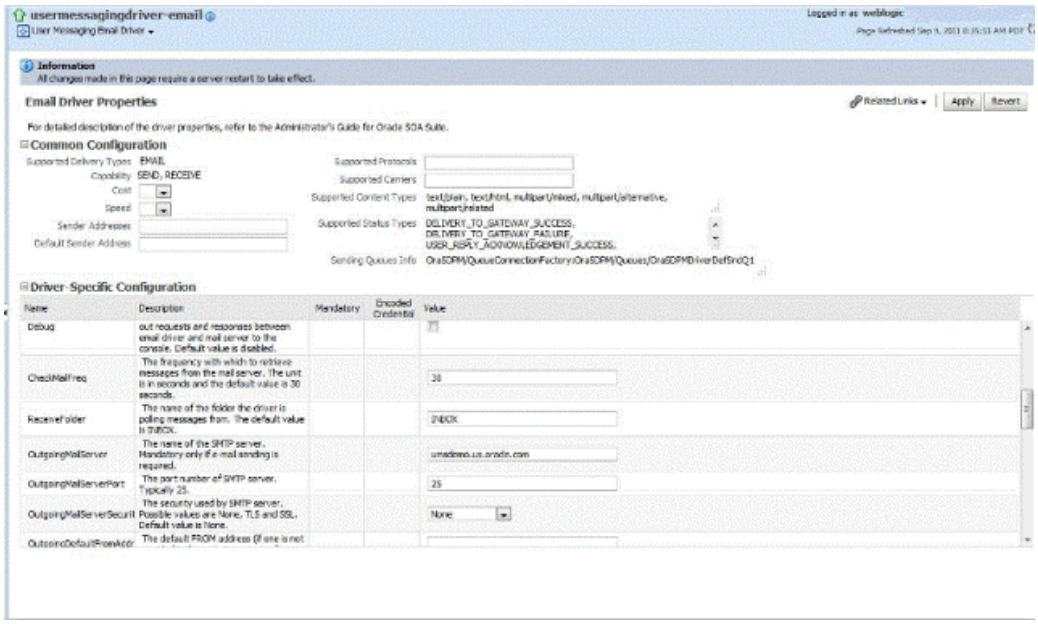
Configuring the Email Driver for the UMS Adapter - Outbound Connectivity

In this procedure, you provide the necessary input to setup the UMS Email Driver for outbound connectivity with the email server.

1. To configure the Email Driver using the Fusion Middleware Control, click **User Messaging Email Driver -> Email Driver Properties**.
2. Enter the name of the SMTP server in the **OutgoingMailServer** location.
3. Enter the port number of the SMTP server in the **OutgoingMailServerPort**. Typically, this is 25.
4. Enter the type of security you want to use with the SMTP server. Possible values are `None`, `TLS` and `SSL`. Default value is `None`.
5. Provide the username used for SMTP authentication in **OutgoingDefaultFromAddr**. This is required only if SMTP authentication is supported by the SMTP server. (An alternative field is **OutgoingUsername**)

6. Provide the password used for SMTP authentication for **OutgoingPassword**. This is required if SMTP authentication is supported by the SMTP server.

Figure 11-3 The Fusion Middleware Control Console Showing the Email Driver Properties Screen



Configuring the Email Driver for UMS Adapter - Inbound Connectivity

The following is the minimum configuration which you must use to set up the UMS Email Driver for inbound scenarios.

1. Enter the **MailAccessProtocol**. This is the E-mail receiving protocol. The possible values are `IMAP` and `POP3`. This value is required only if email receiving is supported on the driver instance.

2. Enter the value for the **ReceiveFolder**. The name of the folder the driver is polling messages from. The default value is `INBOX`.

3. Enter the value of the **IncomingMailServer**. This is the host name of the incoming mail server. Required only if e-mail receiving is supported on the driver instance.

4. Enter the value of the I**ncomingMailServerPort**. This is the port number of IMAP4 that is, 143 or 993) or POP3 (that is, 110 or 995) server.

5. Enter the value of the **IncomingMailServerSSL**. This indicates if you want to enable SSL when connecting to the IMAP4 or POP3 server. The default value is disabled.

6. Enter the email addresses for the IncomingMailIDs. These are the email addresses corresponding to the user names. This is required only if email receiving is supported on the driver instance.

7. Enter the list of user name for the `IncomingUserIDs`. This is the list of user names of the mail accounts from which the driver instance is polling. Each name must be separated by a comma, for example, `foo,bar`. Required only if email receiving is supported on the driver instance.

8. Enter the `IncomingUserPasswords`. This is the list of passwords corresponding to the user names. This is required only if the driver instance supports email receiving.

Note:

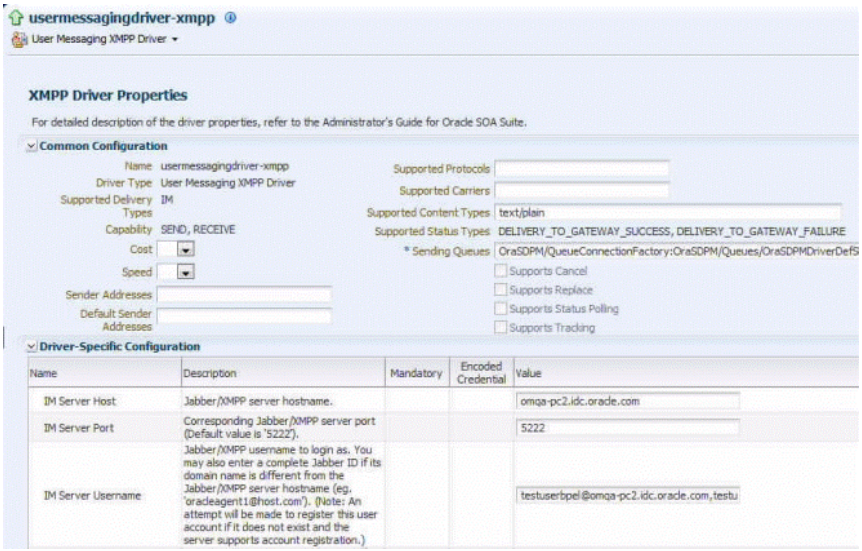
If you specify a list of email ids, ensure you specify the userIds and passwords in the same order. They should correspond one to one in the same order that you specified in FMW Control.

For more details on configuration, see Oracle® Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite.

Configuring the User Messaging XMPP Driver

On the UMS side, you must configure the XMPP driver from the Fusion Middleware Control Console, as in Figure 11-4.

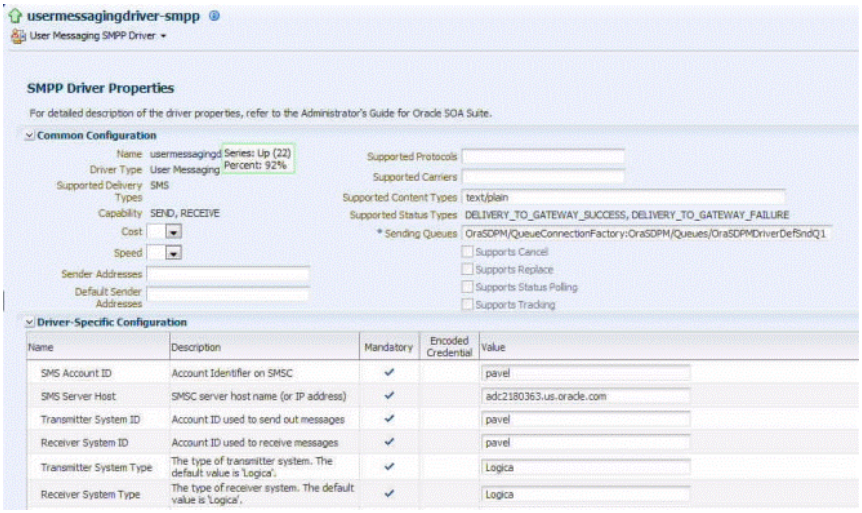
Figure 11-4 Configuring the User Messaging XMPP Driver



Configuring the User Messaging SMPP Driver

On the UMS side, you must configure the SMPP driver from the Fusion Middleware Control Console, as below.

Figure 11-5 Configuring the User Messaging SMPP Driver



Configuring the HTTP Proxy for Firewall traversal

If your Email/XMPP or SMPP Server is outside your firewall and requires an HTTP Proxy configuration, traversal should be configured outside of configuring the UMS Adapter.

For example, such traversal could be configured as JVM arguments in the setDomainEnv.sh. file. For example,

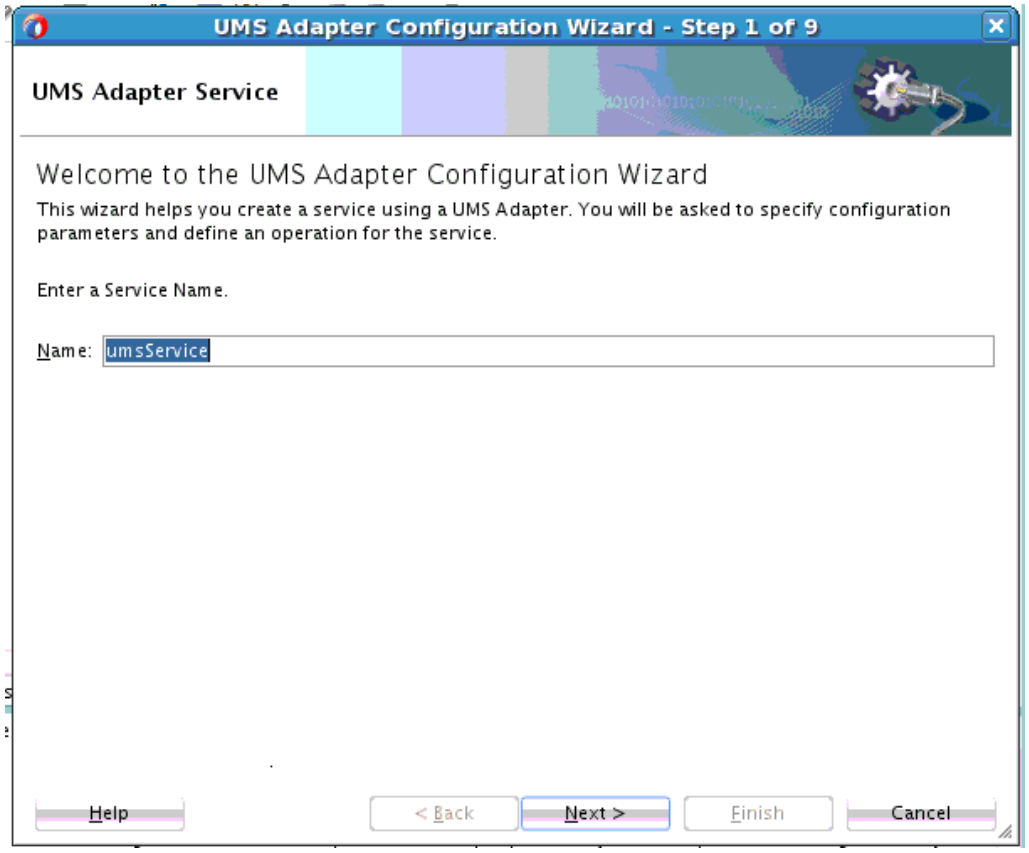
```
JAVA_OPTIONS="${JAVA_OPTIONS} -Dhttp.proxySet=true
 -Dhttp.proxyHost=www.proxy.myserver.com  -Dhttp.proxyPort=80
```

Designing the Adapter Service and the BPEL Process for Inbound Connectivity

Use the UMS Adapter Configuration Wizard within JDeveloper to design the inbound UMS Adapter reference.

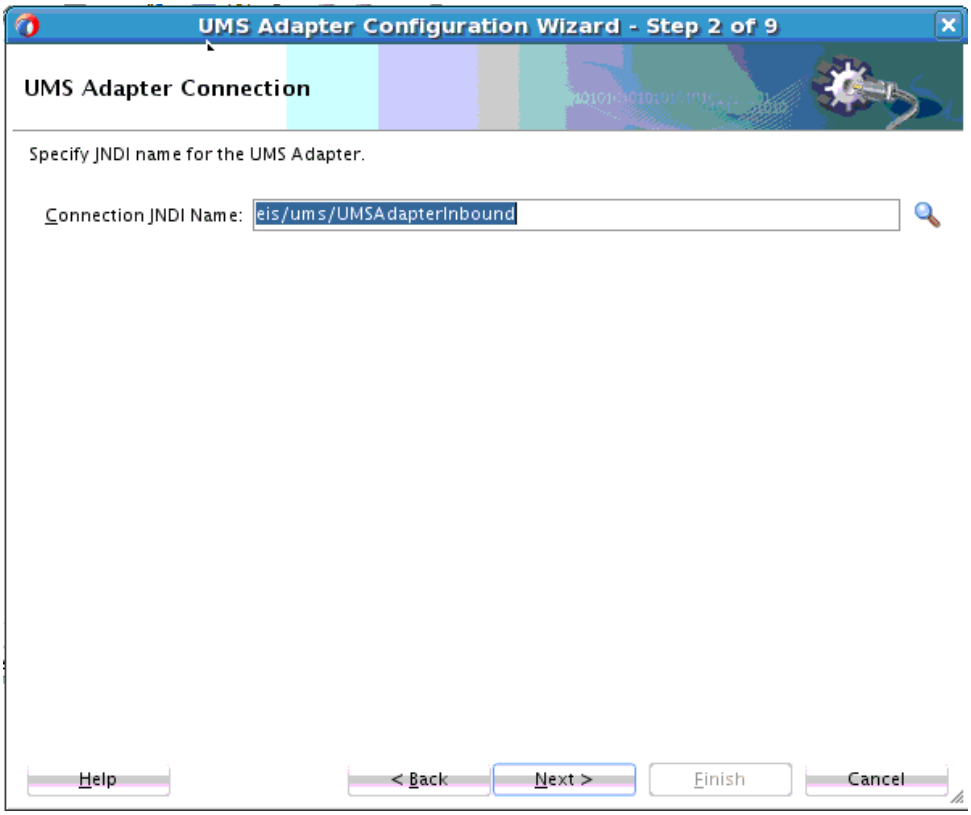
1. Drag and drop the **UMS Adapter** from the Components window to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed. The UMS Adapter also displays a Service or Reference name, which you can choose to modify, as required.

Figure 11-6 UMS Adapter Configuration Wizard, Welcome Screen



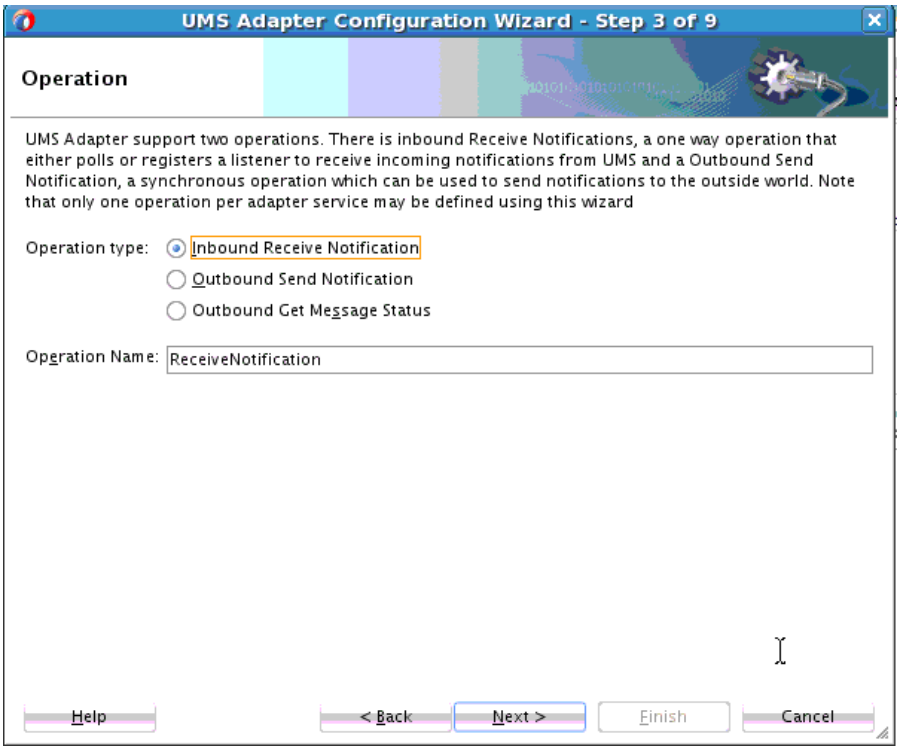
2. On the **UMS Adapter Connection** page, enter the Connection JNDI Name. Here, `eis/ums/UMSAdapterInbound` is specified as the JNDI name, which is the default JNDI name for Inbound Connections.

3. ***Figure 11-7 UMS Adapter Configuration Wizard Connection Screen***



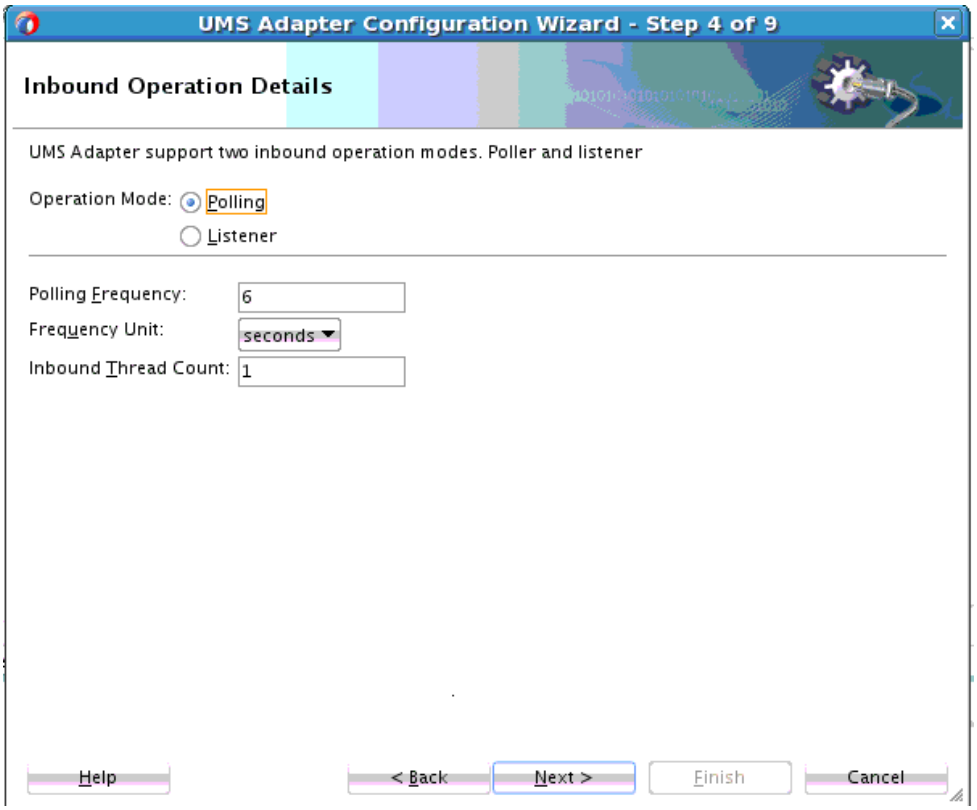
4. On the **Operation Type** screen of the UMS Adapter Configuration wizard, select the operation to perform. Based on your selection, different adapter configuration wizard pages appear and prompt you for configuration information. For Inbound connectivity, on the Operation screen, select `Inbound Receive Notification` as the operation type and click **Next**.

Figure 11-8 The UMS Adapter Configuration Wizard Operation Screen, Inbound Receive Operation Type Chosen



5. The **Notification Details** page of the Adapter Configuration Wizard enables you to specify the mode, `Polling` or `Listener`, in which to receive incoming notifications from the UMS Server.

Figure 11-9 UMS Adapter Configuration Wizard Notification Details Screen



6. Selecting the **Polling mode** enables you to specify inbound polling parameters:

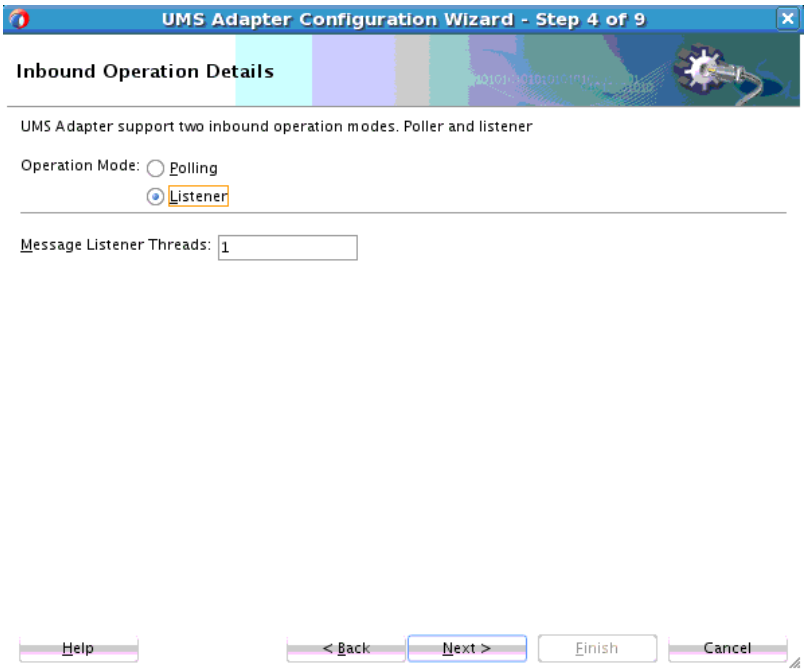
   - **Polling Frequency**. The frequency with which to poll the UMS for new notifications to retrieve. The default values displayed for Polling Frequency = 6, the Frequency Unit =seconds, and the InboundThreadCount=1 .
   - **Frequency unit**. Specify seconds, minutes, hours, days or weeks as the unit for frequency.
   - **Inbound Thread Count**. Specify the number of polling threads.

   The UMS Adapter receives messages until the messages are available in the Inbox. When there are no more messages, and only then, the UMS Adapter sleeps for the polling interval you specify on this screen. This sleeping activity avoids mounting large number of messages in the Inbox, within high-incoming message volume scenarios. Each polling thread retrieves one message at a time, processes it and then publishes it.

7. Selecting the **Listener Mode** enables you to specify the number of **Message Listener Threads**. This property controls the number of listener worker threads on the UMS Server side. The default value is 1. Specifying this property means the UMS Server will provide multi-threaded asynchronous receiving of incoming notifications.

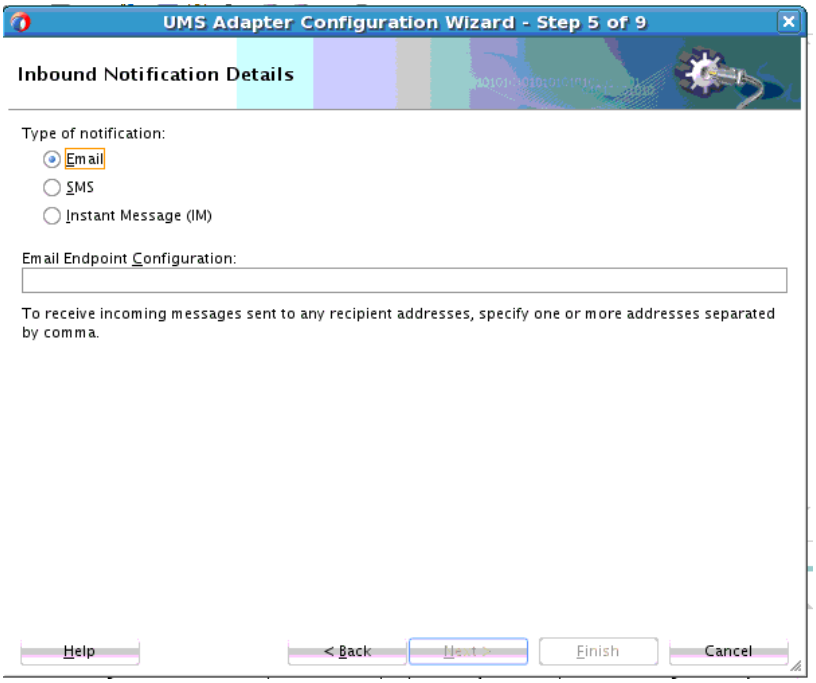
   Note that Polling Mode is a pull mechanism, which you can use to fetch messages intermittently. You can use Listening Mode to get messages in real time, as and when they arrive. Your use of either depends on your business implementation scenario.

Figure 11-10 UMS Adapter Configuration Wizard Notification Details with Listener Mode Selected



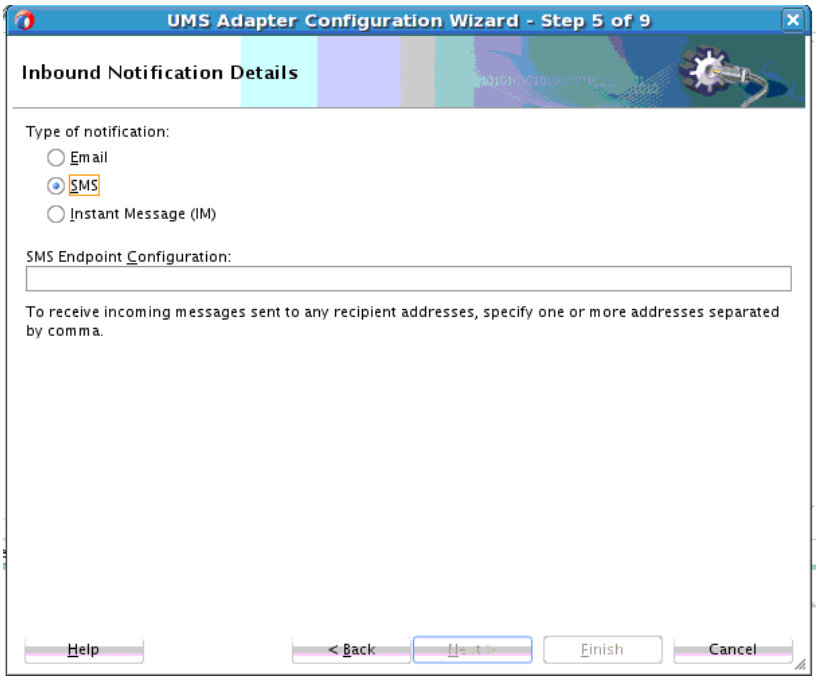
8. The second page of the **Notification Details Page** of the UMS Adapter configuration wizard enables you select the type of notification to receive with the Notification Endpoint Configuration. It configures the end point attributes after selecting the type of Notification as Email, SMS, or IM. For example, selecting Email, users enters other attribute details as one or more incoming mail addresses. You can specify more than one comma-separated email mail box address from which you want to receive email notifications.

Figure 11-11 The UMS Adapter Configuration Wizard Inbound Notification Details Screen, Second Page



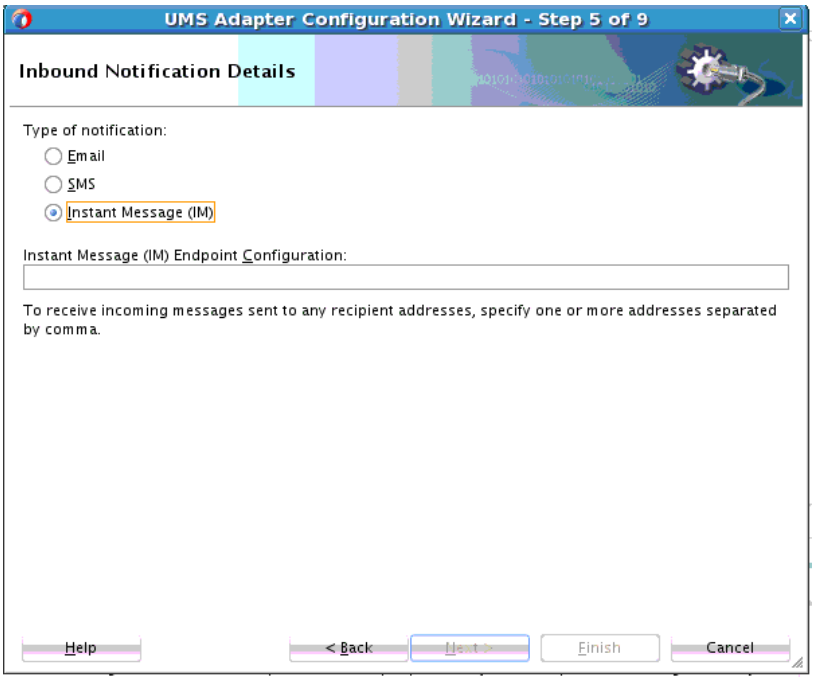
If you choose SMS, the recipient address is a mobile number. You can also specify a mobile number range, for example "16501230000, 16501234999" means all numbers from 16501230000 to 16501234999 (inclusive).

Figure 11-12 Inbound Notification Details Screen with SMS Selected



For Instant Messaging inbound notification, you can indicate an instant messaging id.

Figure 11-13 Inbound Notification Details with Instant Messaging Selected



9. Click **Next** to continue, or **Finish** to complete using the UMS Adapter Configuration Wizard without configuring message filters against incoming messages.

10. You can use message filters against incoming messages. A message filter contains a matching criterion and an action. You can register a series of message filters. They are applied in order against an incoming (received) message; if the specified criterion matches the message, the action is taken.

    For example, you can implement any required blacklists, by rejecting all messages from a given sender address.

    You can specify three different types of filters on this screen. (For more information on Java Patterns, or Regular Expressions, see the reference at http://download.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html and the tutorial at http://download.oracle.com/javase/tutorial/essential/regex)

    a. **Blacklist Filter**-Blacklist filters match against an incoming message's sender address, and reject the message if the sender address matches the given Java pattern. (That is, a Java regular expression).

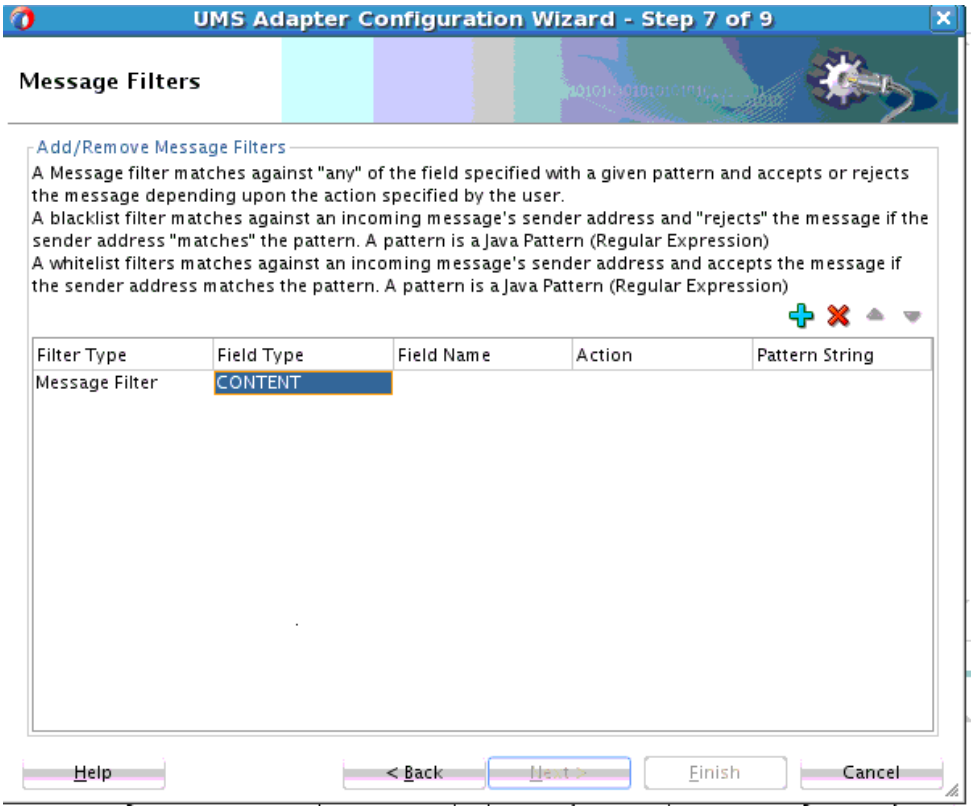
    b. **Whitelist Filter**-Whitelist filters match against an incoming message's sender address and accepts the message if the sender address matches the given Java pattern.

    c. **Message Filter**-A Message Filter matches against any of the fields you indicate with a given pattern and accepts or rejects the messages depending upon the action you specify. These fields include: `CONTENT`, `HEADER`, `RECIPIENT`, `REPLYTO`, `SENDER`, `SUBJECT`

    There are only two types of Actions that apply to the Message Filters, either `ACCEPT` or `REJECT` the message. Message Filters are applied in the same order in which you define them on the Message Filter page.

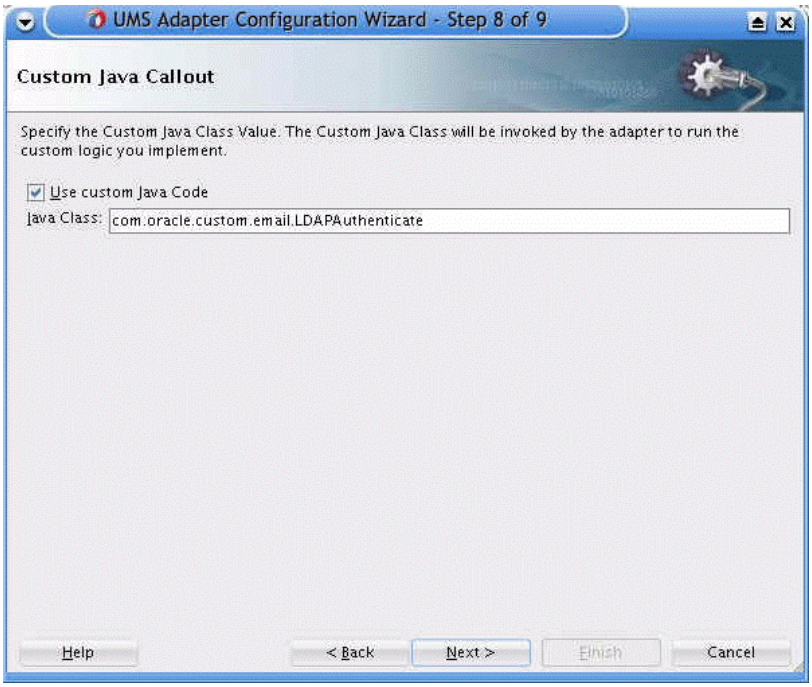
Click **Finish** to complete the configuration through the Wizard, or click **Next** to proceed to the Java Callout Screen to specify a Java class through which you can apply additional filtering.

Figure 11-14 UMS Adapter Message Filters Screen



11. On the **Java Callout** page, specify the name of the Java class you want the UMS Adapter to invoke and which will run custom logic you provide.

Figure 11-15 UMS Adapter Configuration Wizard Custom Java Callout Screen



12. If you want to provide a custom Java callout, select the checkbox and provide a classname in the text box. See the description in the custom Java Callout class section in this chapter.

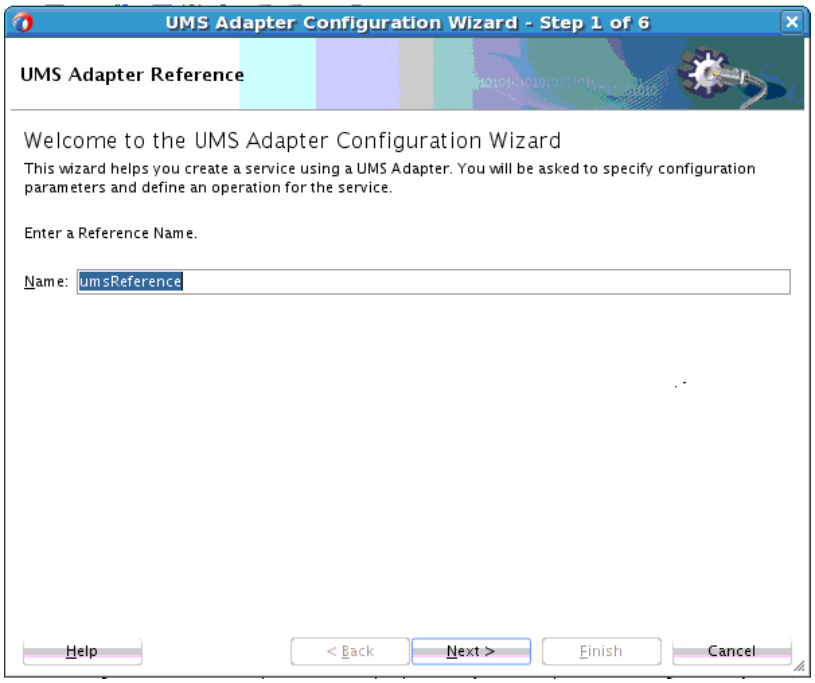
13. Click **Finish** on the page below to complete configuring the UMS Adapter service. When you finish configuring the Oracle UMS Adapter, a JCA file is generated for the inbound service. The file is named after the service name you specified on the **Service Name** page of the Adapter Configuration Wizard. You can rerun the Wizard later to change your operation definitions.

Designing the Adapter Service and the BPEL Process for Outbound Connectivity

Use the UMS Adapter Configuration Wizard within JDeveloper to design the outbound UMS Adapter reference.

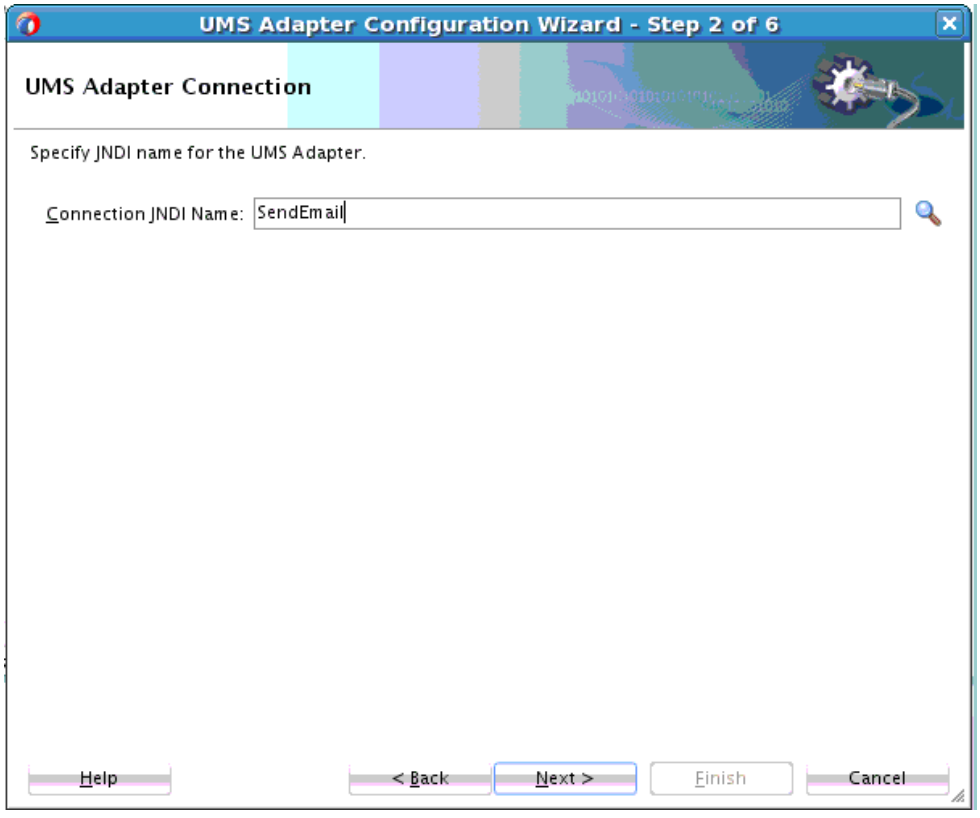
1. Drag and drop the **UMS Adapter** from the Components window to the External References swim lane.The Adapter Configuration Wizard Services/References page is displayed.

Figure 11-16 UMS Adapter Configuration Wizard Reference Welcome Screen



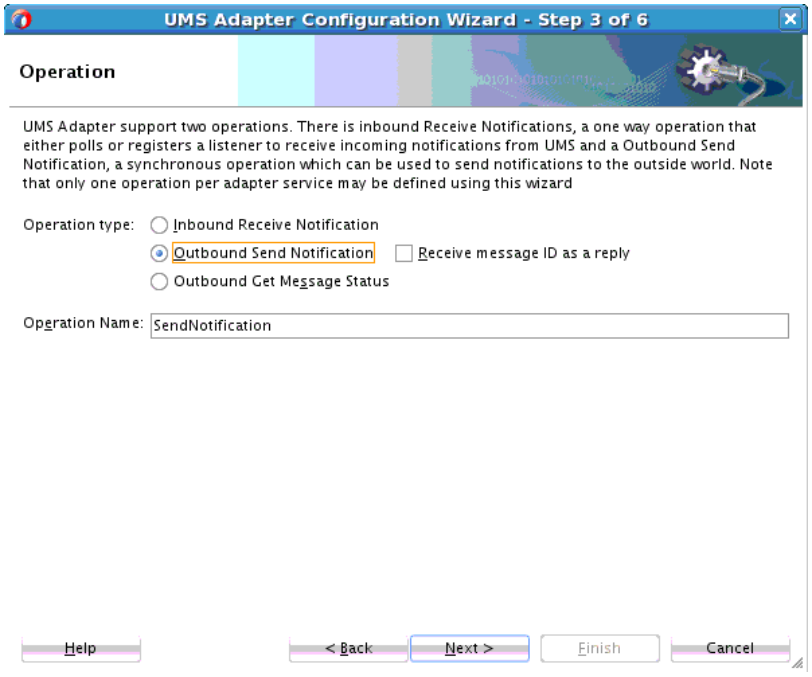
2. The **UMS Adapter Service Name** screen is displayed. Enter `SendMail` for the Outbound Service Name, for example.

Figure 11-17 UMS Adapter Configuration Wizard Service Name Screen, Outbound Send Mail Operation



3. On the **Operation** screen, select `Outbound` as the Operations type. The Operation Name defaults to `SendNotification`. Here, it is changed to SendEmail. If the message is outbound, it is a syncrhonous or one-way invoke for external notifications, and there is an option included to receive the unique message as a reply, which can be returned by the UMS Server after accepting the outbound notification request.

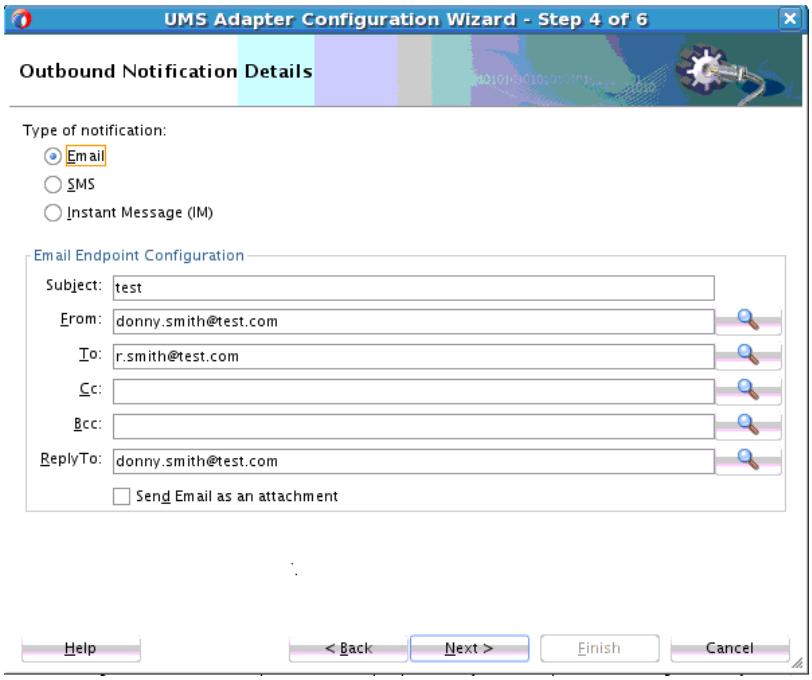
Figure 11-18 UMS Adapter Configuration Wizard Operation Screen with Outbound Send Operation Selected



4. The **Outbound Notification Details** screen appears. The Email button is selected. Enter the Endpoint Configuration detail items, or browse to find them. You also can specify a failover address for any primary address you provide. For example, `scott.tiger@sport.com:failover-id@example.com.` This failover addressing applies to To, CC and Bcc addresses.

   You can click each browse button and browse the identity service using the **Identity Look-up Dialog** to search and fill address attributes, as required.Click **Next** to proceed or **Finish** to complete using the Wizard.

Figure 11-19 The Outbound Notification Details Screen with Email Selected

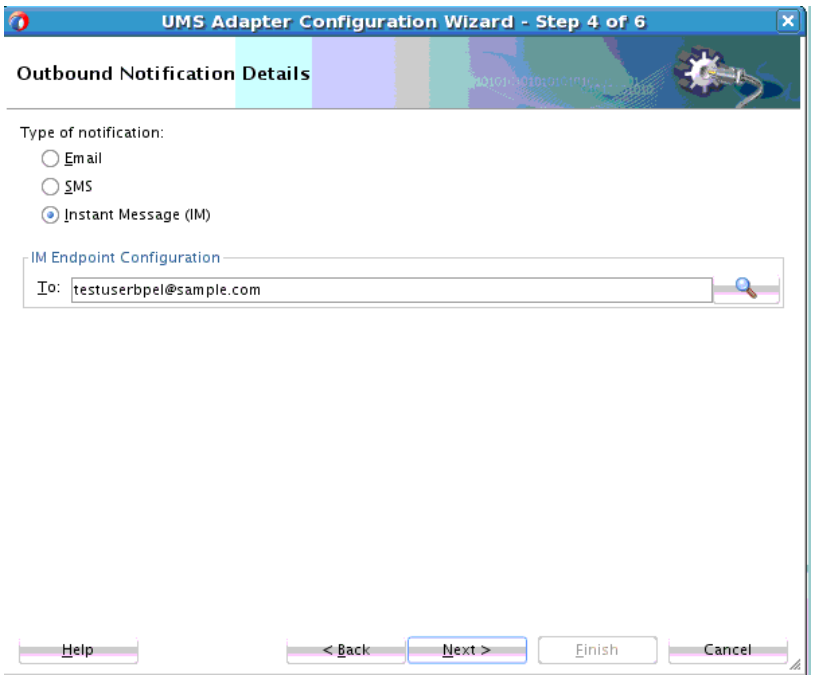


5. Define the message for the UMS Send operation using the **Messages Screen**. If you choose **Message is Opaque(Base64Binary)** or **Message is String Type**, you do not have to specify a URL for the Schema and a Schema Element. If you choose a URL for the Schema, you must specify a **Schema Element**.

   Note that when creating outbound notifications, you can also use SMS or Intant Messaging. In this example, you are using email notifications.

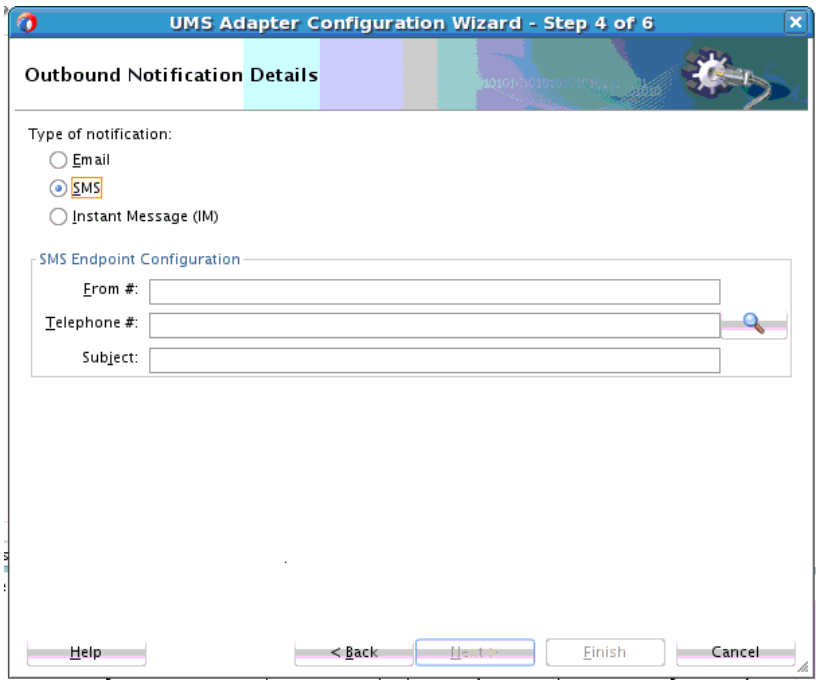
   The screen shot below shows **Instant Messaging** selected. In this case, the end point address is an Instant Messaging address. Addresses can be separated by commas; you can specify more than one comma-separated IM address from which you want to receive IM notifications. The address you supply must be in Instant Messaging format. You can use the browser button to reach the Identity Lookup screen to perform a lookup of the IM address you want to select.

Figure 11-20 Outbound Notification Details Screen with IM Selected



The screen shot below shows the **Outbound Notification Screen** with **SMS** selected. The end point is a mobile number. You can specify more than one comma-separated SMS mailbox address from which you want to receive email notifications. Ensure that the addresses are separated by a comma. You can also specify a mobile number range, for example, "16501230000, 16501234999" means all numbers from 16501230000 to 16501234999 (inclusive).

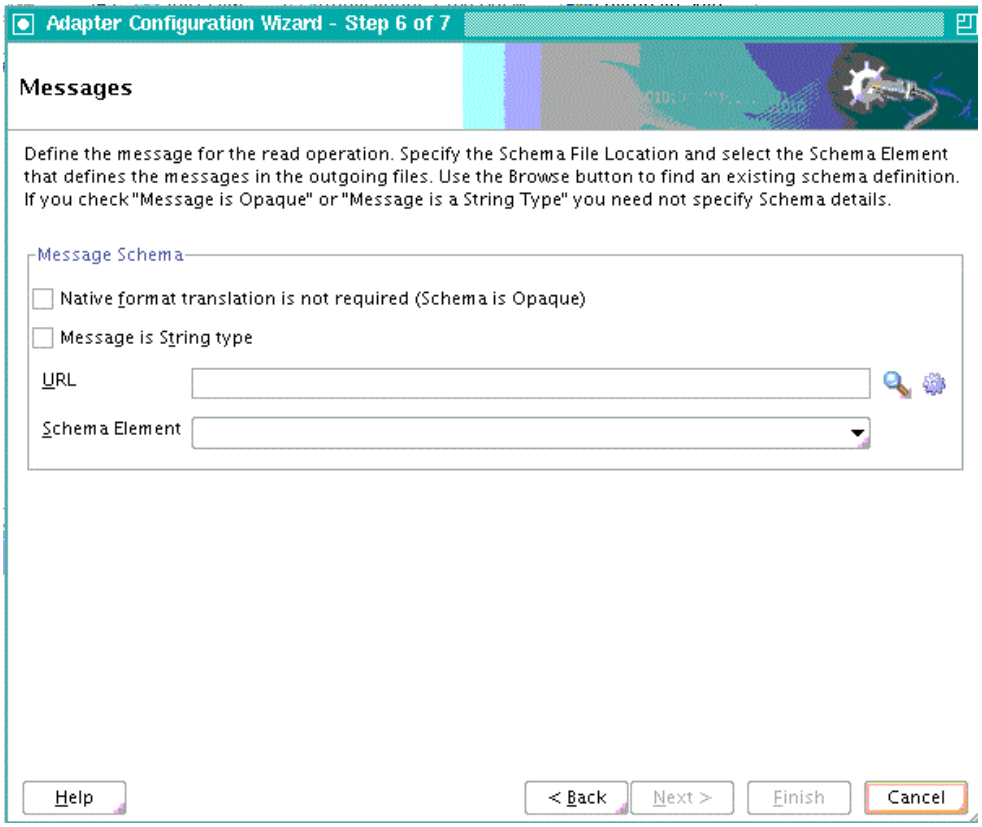
Figure 11-21 Outbound Notification Details Screen with SMS Selected



With any of these, you can use the magnifier icon to browse the identity service using the **Identity Look-up Dialog** to search and fill address attributes.

For all the messaging channels, you can opt to specify a user or a group rather than a device address. For example, `USER:weblogic` or `GROUP:admins`.

Figure 11-22 UMS Adapter Configuration Wizard Messages Screen



6. Click **Finish** to complete using the UMS Adapter Configuration Wizard.
7. The **Finish Screen** appears. You have completed using the UMS Adapter Configuration Wizard. Click **Finish** to complete creating the xsd and WSDL at the locations indicated on the screen.

Note:

Within Fusion Middleware Control, you can use Monitoring reports for the UMS Adapter as you would with other deployed Adapters. Note, however that for UMS Adapter in Fusion Middleware Control, the EIS connection shows as connected though in fact the connection is down.

12

Oracle JCA Adapter for LDAP

The Oracle LDAP adapter provides bi-directional connectivity with LDAP V3-compliant directory servers. The chapter provides information on how to model the LDAP adapter as a reference to perform CRUD (Create, Read, Update and Delete) operations and how to model the LDAP adapter as an inbound publication service on a directory server. Information describing various users, applications, files, printers, and other resources accessible from a network is often collected into a special database called an LDAP directory. Access to LDAP directories is a basic requirement of enterprise workflows. The LDAP Adapter Configuration Wizard provides a graphical and intuitive interface to model LDAP services to send requests to and receive responses from LDAP servers.

Figure 12-1 LDAP Adapter Architecture Diagram

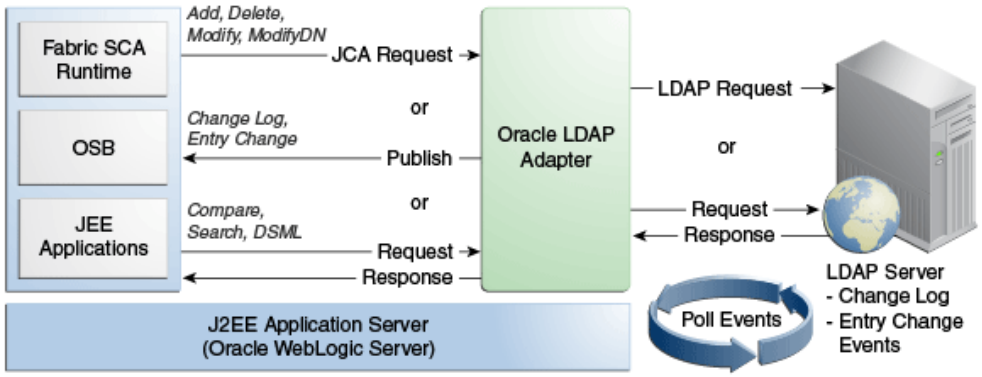


Figure 12-1 illustrates how the Oracle LDAP Adapter exposes synchronous and asynchronous service interfaces that can be used to execute CRUD operations on a target directory server. It illustrates how the Oracle LDAP Adapter can poll for change events from a source directory server and publish these events, thereby acting as a publication service. The LDAP Adapter also provides a DSML synchronous service which takes a DSMLv2 request, executes it on the target directory server and returns the DSMLv2 response.

The LDAP Adapter provides an extensive solution for configuring and connecting to the LDAP Server in a SOA SCA environment. The runtime component provides physical connectivity to the external LDAP Server. It can act as a reference, supporting Outbound services sending requests to the LDAP Server. It can also act as a service, supporting Inbound Services polling to change events on the LDAP Server.

This chapter includes the following sections:

- LDAP Concepts
- LDAP Adapter Configurations
- Oracle LDAP Adapter Features

- LDAP Adapter Samples

LDAP Concepts

LDAP (Lightweight Directory Access Protocol), is an Internet protocol for accessing information directories. A directory service is a distributed database application designed to manage the entries and attributes in a directory. LDAP runs over TCP/IP. LDAP enables clients to access different directory services based on entries. It makes the entries, along with their attributes and values, available to users and other applications, on a controlled-access basis.

The Oracle Adapter for LDAP provides rich support for LDAPv3 defined operations in addition to support for extensions. It also supports DSMLv2 and can be used as a DSML gateway service. The adapter can be configured to listen and publish change events from a source directory server. In addition, the adapter guarantees once and only once message delivery, high-availability, policy-based retry and fault-handling, automatic fail-over and several other features.

This section includes the following:

- LDAP Entries_ Attributes and Values
- LDAP Directory Structure
- Distinguished Names and Relative Distinguished Names
- LDAP Service and Service Client
- Referrals
- Aliases
- Controls

LDAP Entries, Attributes and Values

An LDAP directory contains entries that contain information pertaining to an entity. Each of the entry's attributes has a name and one or more values. The names of attributes are most often mnemonic strings, such as `cn` for common name, or `mail` for email address. For example, a company might have an employee directory. Each entry in the employee directory represents an employee. The employee entry contains such information as the name, e-mail address, and phone number. See the following example:

```
cn: John Doe
mail: johndoe@mydomain.com
mail: jdoe@mydomain.com
telephoneNumber: 471-6000 x.1234
```

Each part of the descriptive information, such as an employee's name, is known as an attribute. In the example, the Common Name (cn) attribute, represents the name of the employee. The other attributes are `mail` and `telephoneNumber`.

Each attribute can have one or more values. For example, an employee entry can contain a mail attribute whose values are `johndoe@mydomain.com` and `jdoe@oracle.com`. In the previous example, `the` mail attribute contains two mail values.

LDAP Directory Structure

The organization of a directory is a tree structure. The topmost entry in a directory is known as the root entry. This entry normally represents the organization that owns the directory.

Entries at the higher level of hierarchy represent larger groupings or organizations. Entries under the larger organizations represent smaller organizations that make up the larger ones.

The leaf nodes (or entries) of the tree structure represent the individual persons or resources.

Distinguished Names and Relative Distinguished Names

An entry is made up of a collection of attributes that have a unique identifier called a distinguished name (DN). A DN consists of a name that uniquely identifies the entry at that hierarchical level.

In the example below, John Doe and Jane Doe are different common names (cn) that identify different entries at that same level.

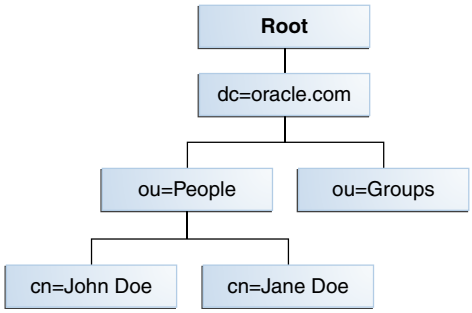
A DN is also a fully qualified path of names that trace the entry back to the root of the tree. For example, the distinguished name of the John Doe entry is:

```
cn=John Doe, ou=People, dc=mydomain.com
```

A relative distinguished name (RDN) is a component of the distinguished name. For example, `cn=John Doe, ou=People` is a RDN relative to the root RDN `dc=mydomain.com`.

DNs are used to describe the fully qualified path to an entry while an RDN is used to describe the partial path to the entry relative to another entry in the tree.Figure 12-2 illustrates an example of an LDAP directory structure with distinguished names and relative distinguished names.

Figure 12-2 The LDAP Hierarchy Tree



LDAP Service and Service Client

A directory service is a distributed database application designed to manage the entries and attributes in a directory. A directory service also makes the entries and attributes available to users and other applications. The Oracle Internet Directory server is an example of a directory service. Other directory services include Oracle Directory Server Enterprise Edition and Microsoft Active Directory.

A directory client accesses a directory service using the LDAP protocol. A directory client can use one of several client APIs available in order to access the directory service.

Referrals

The Oracle LDAP Adapter APIs query the results of a search based unspecified referral criteria. The search results can consist of a number of referrals.A referral is an entity that is used to redirect a client's request to another server. A referral contains the names and locations of other objects.

For example, an LDAP server sends a referral to the client to indicate that the information that the client has requested can be found at another location (or locations), possibly at another server or several servers.

The referral contains the URL of the LDAP server that holds the actual entry. The LDAP URL contains the server's host/port and an object's DN.

Aliases

The Oracle LDAP Adapter APIs query the results of a search based on specified alias dereferencing criteria.

An alias is an entity that is used to define the position of an entry within the current Directory Information Tree. An alias holds the DN of the entry that contains the information.

LDAP Adapter Configurations

The following sections relate to LDAP Adapter Configurations.

Controls

Controls provide a mechanism for extending existing operations in LDAP. Controls can be passed along with request operations in the LDAP adapter.

If included with the operation request message, you can use control information to provide additional information about the way the operation should be processed.

Request Control Format

The format of a request control is name-criticality-value tuples:

```
requestControls="controlName1|controlOID:criticality:prop1:value1:prop2:value2;
```

`controlName|controlOid` is the unique name/oid of the control that must be passed along with a request operation.

`criticality` is a boolean attribute that signifies the behavior if the control passed along with a request is not supported by the target directory server.You can optionally provide a value string that represents values that can be passed to a request control.

LDAP Control Restrictions

There are several restrictions in using LDAP controls with the LDAP Adapter:

- An unsupported control with criticality as `true` raises an exception and the LDAP operation is not executed on the target directory server.

- Controls that are only applicable to an operation are executed. If not applicable, then the controls passed along with the operation are ignored.
- Some controls are only applicable to a single operation whereas some controls are relevant to multiple operations. For example, `subtreeDeleteRequestControl` is only applicable to the delete operation whereas `assertionRequestControl` is applicable to all operations.
- Controls that are not understood by the LDAP Adapter are treated as generic. The behavior of such controls is not well defined.
- Only request controls are supported. The LDAP Adapter does not support Response controls.

See Table 12-1 for a list of Directory Servers, controls and LDAP Adapter operations.

Table 12-1 Supported Directory Server Controls and LDAP Adapter Operations

Operations	Controls (Name)	Controls (OID)	Microsoft Active Directory	ODSEE (Oracle Directory Server Enterprise Edition)	OID
Search	ManageDsaITRequestControl	2.16.840.1.113730.3.4.2	Not Supported	Supported	Supported
-	ServerSideSortRequestControl	1.2.840.113556.1.4.473	Supported	Supported	Supported
-	GetEffectiveRightsRequestControl	1.3.6.1.4.1.42.2.27.9.5.2	Not Supported	Supported	Not Supported
-	MatchedValuesRequestControl	1.2.826.0.1.3344810.2.3	Not Supported	Not Supported	Not Supported
-	-	-	-	-	-
Delete	SubtreeDeleteRequestControl	1.2.840.113556.1.4.805	Supported	Not Supported	Not Supported
-	-	-	-	-	-
Modify	PermissiveModifyRequestControl	1.2.840.113556.1.4.1413	Supported	Not Supported	Not Supported
-	-	-	-	-	-
ModifyDN	No control listed in wizard	-	-	-	-

Non Default Control Configuration for Design Time Wizard

The LDAP Adapter configuration wizard enables you to attach request controls along with an operation.The **Attach Control** screen operates in the following manner:

- By default, only the controls applicable to an operation and supported by the connected directory server are listed in the wizard.

- Example: `assertionRequest` and `SubtreeDelete` are controls applicable to the delete operation; but if the target directory server does not support the subtree delete operation, only the `assertionRequest` is listed.
- You can choose to add from other supported controls. These controls are treated as generic and you should test these controls thoroughly as the runtime behavior of these controls is not well-defined.

Control Availability

By default, only a few frequently used control names are displayed within the LDAP Configuration Wizard. If you want the Wizard to display the control names for non-named controls, you must follow these manual steps to update control definitions and provide the appropriate names and details.

1. In a terminal window, change to the SOA Adapters JDeveloper plugin library directory:

   ```
   cd <install-dir>/soa/plugins/jdeveloper/integration/adapters/lib
   ```

2. Unjar the LDAP Adapter jar.

   ```
   cd <install-dir>/soa/plugins/jdeveloper/integration/adapters/lib oracle/tip/
   adapter/ldap/config/control-oid.properties
   ```

3. Edit `oracle/tip/adapter/ldap/config/control-oid.properties` to provide the control and relevant details you want to provide.
4. Jar the file up again to include the changes.

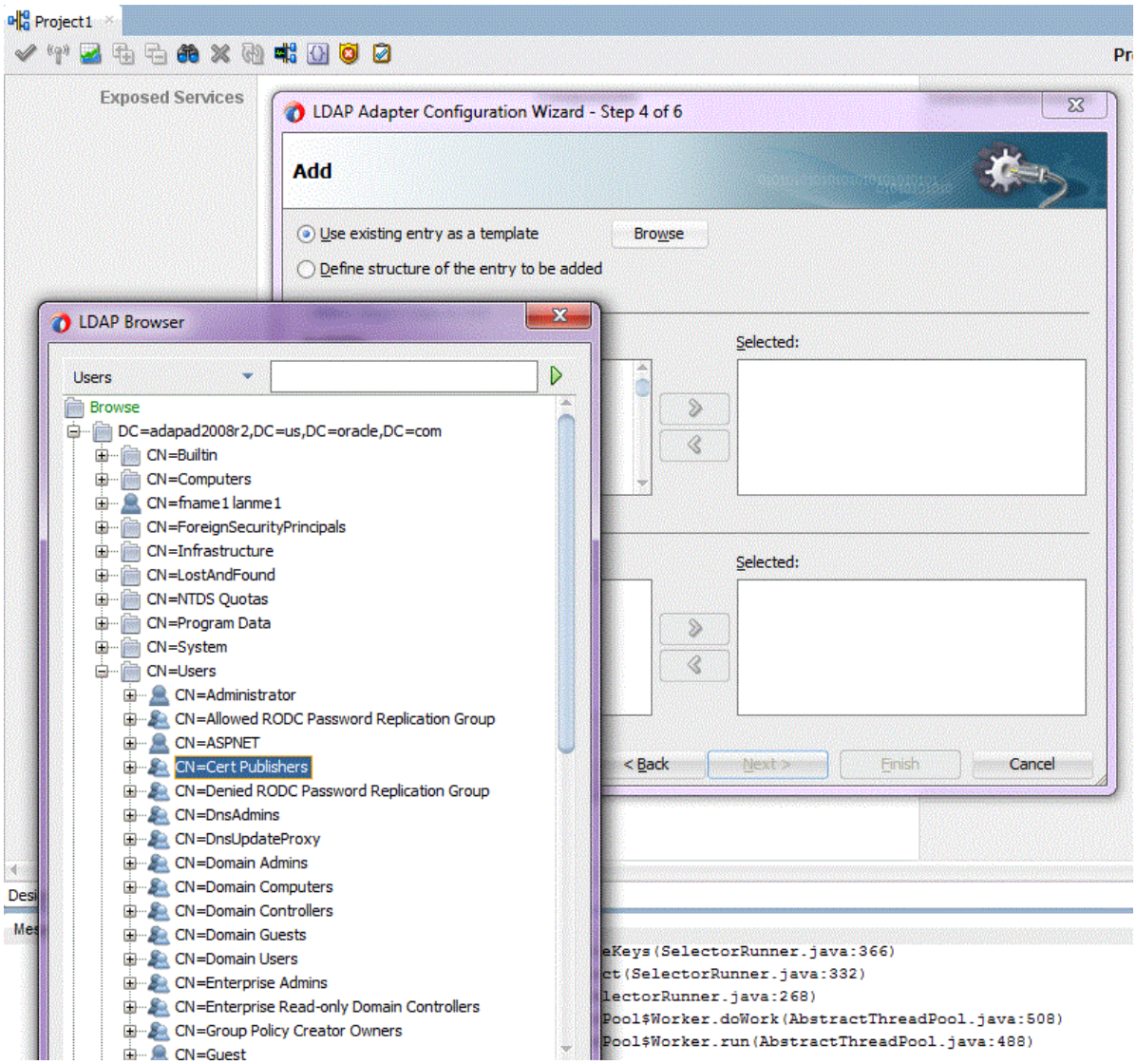
   ```
   jar -uvf LdapAdapter.jar  oracle/tip/adapter/ldap/config/control-oid.properties
   ```

5. Restart JDeveloper.

LDAP Browser

Use the LDAP Browser Page to show the Directory Information Tree (DIT). The Browser is called from LDAP Configuration Adapter wizard when you select Browse. See Figure 12-3.

Figure 12-3 The LDAP Browser Shown in Context



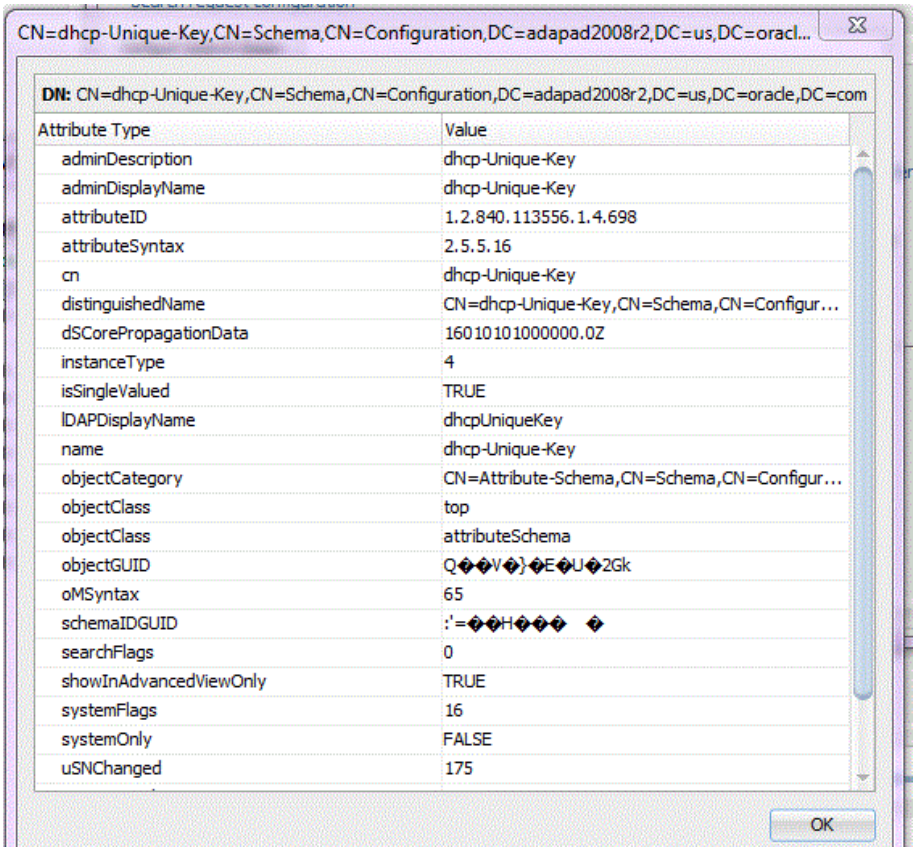
To choose an entry, select the entry in the DIT and click the **OK** button. The selected entry is used further in the LDAP Adapter Configuration wizard as a baseDN for a Search operation you perform, the Entry Change Notification operation or as a template for the Add operation.

Attribute Viewer

To see attributes related to an item, double-click the selected entry.

The Attribute viewer displays in a two-column layout. The left column contains the attribute descriptions (or names), the right column contains the attribute values. Each line represents a name-value pair. The section immediately previous, the attributes table displays the distinguished name of the current entry. See Figure 12-4.

Figure 12-4 The LDAP Configuration Wizard Browser Attribute Viewer

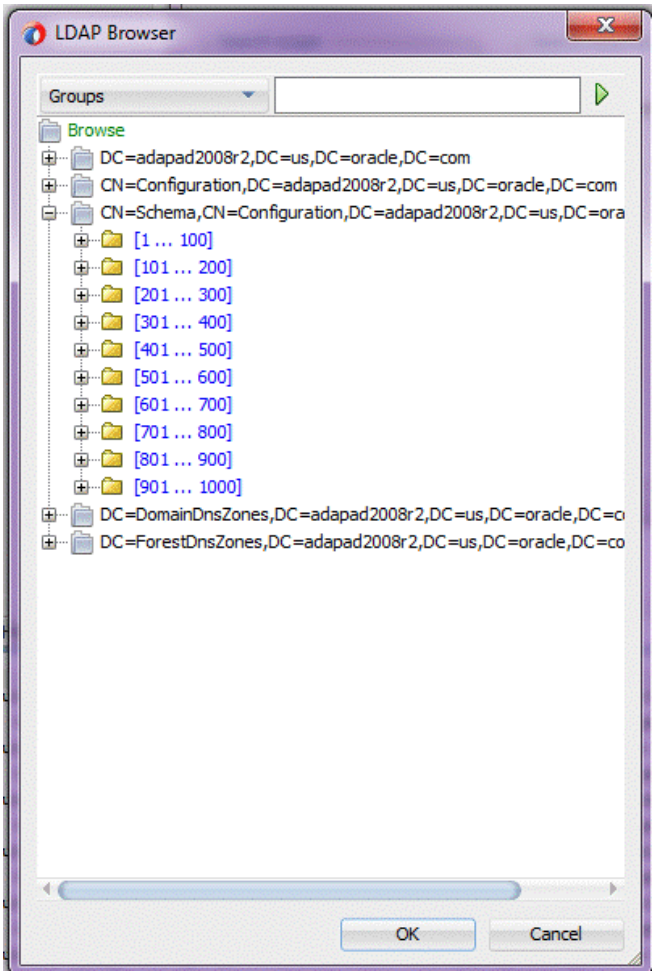


Folding

The directory information tree of the LDAP directory is displayed in its natural hierarchical structure. The first hierarchy level contains the base entries. When expanding an entry its direct children are fetched from directory. While browsing the directory, the LDAP Browser attempts to find if a fetched entry has children. Entries without children cannot be expanded.

By default, large browse results are folded into virtual folders each with 100 entries. The displayed tree is thus smaller. The number of fetched items is limited by 1000 items for performance improvements. To operate with folders with over 1000 records you can use the Search operation. See Figure 12-5.

Figure 12-5 The LDAP Adapter Configuration Wizard LDAP Browser Directory Folding



Searching

The Search operation is an alternative approach that you can use when browser does not load the expected entries.You can search the DIT by specifying a single search attribute and value. The search is processed over the whole subtree. There is no limitation on the quantity of search results.

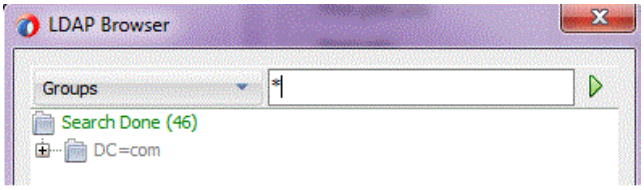
In the left combo-box. you can select the Search attribute from the list of commonly used attributes: `cn`, `givenName`, `mail`, `member`, `objectClass`, `sn`, `telephoneNumber`, `uniqueMember`.

Use the option `Groups` to specify that the Search is processed among the entries that are groups.

Use the option `Users` to indicate that the search are processed among the entries that are persons, accounts, or users.

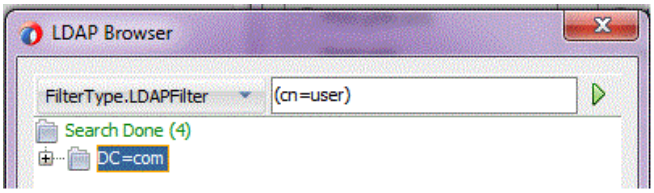
Enter the search value in the right input field. Use '*' as wildcard as needed.

Figure 12-6 LDAP Configuration Wizard Browser Showing Supplying a Wildcard to the Search Operation within the LDAP Browser



To start the search operation, select the green triangle button on the right side.In the left combobox, select `FilterType.LDAPFilter` to set the LDAP search filter in the right field manually.

Figure 12-7 LDAP Configuration Wizard Browser Showing Supplying a Filter to the Search Operation within the LDAP Browser



Oracle LDAP Adapter Features

The LDAP protocol and the LDAP adapter provide fast read-access to directory information. The LDAP Adapter supports both inbound and outbound messages and includes the following features, as detailed in this section:.

- Configuring the LDAP Adapter
- Outbound Operations
- Inbound LDAP Adapter Features
- LDAP Browser

Note:

The LDAP Adapter does not provide any transaction support on outbound messages.

Configuring the LDAP Adapter

You use the LDAP Configuration Wizard to configure the LDAP Adapter, to model LDAP services through contextual options and browsing on the LDAP directory schema. The LDAP Adapter wizard requires a connection to the target LDAP directory server which can you configure in the following manner.

1. Drag and drop the LDAP Adapter component from the **Components** window to the **Exposed Services** or to the **External References** swimlane. The LDAP configuration wizard **Welcome and Service** name screen appears.
2. The **Welcome and Service Name** screen appears. Enter the name of a service. Click **Next**. The **Service Connection** screen appears

If a connection is not available, use **Create New Connection** to create a new Connection. Figure 12-22 LDAP Adapter Configuration Wizard Service Connection Screen,

3. The **Create a New Connection** dialog allows a user to connect to the target LDAP server.

   On this screen, you can supply the Network parameters required to reach the target directory server. Enter the following:

   - **Connection name**: String value to bind this connection with which to bind this information.
   - **Connection type**: Currently only LdapV3 is supported (which is the default.)
   - **Host name**: name or the IP address of the machine where the LDAP server is running.
   - **Port**: Where the LDAP service is running.Use the SSL port if the encryption method is SSL.
   - **Encryption method**:
     - **No encryption**: Unencrypted channel.
     - **use ssl**: SSL connection.
     - **use startTLS**: Secure channel over an unencrypted connection.
     - **Truststore**: Path of the trust store that are used for server authentication.
     - **Truststore password**: Password to access the truststore.

     Authentication parameters to bind to the given directory server.

     - **Anonymous**: Bind anonymously to the directory server to which the network parameters point. Note, you must enable anonymous bind on the target directory server, which is a non-default configuration on most LDAP servers.
     - **Simple Authentication**: Username/password authentication.

     Test connection. Test the connection configuration in the LDAP Adapter Configuration wizard. Test connection establishes a connection to the directory server with the configured network parameters and binds to the directory server using the configured authentication parameters.

JNDI Connection Pool Properties for the LDAP Adapter

Table 12-2 provides JNDI connection pool properties for the LDAP Adapter.

Table 12-2 JNDI Connection Pool Properties for the LDAP Adapter

Property Name	Description	Default	Required
bindDN	Distinguished Name required to bind to directory server. Example, cn=orcladmin	cn=Directory Manager	Yes (Needed when using an authentication method other than anonymous.)
hostName	Host name of the directory server.	localhost	Yes
inboundDataSource	JNDI string location pointing to a valid XA data source. Example: jdbc/SOADataSource	None	Yes (if the LDAP Adapter has to be used on the inbound direction.)
operationTimeout	Sets the operation timeout. If the response is not received from the directory server in the timeout period, the operation is abandoned and an exception is raised. Example: 1000	False	No
password	Password of the directory server.	welcome	Yes (Needed when using an authentication method other than anonymous.)
port	Port on which directory server is running..	389	yes
hostName	Host name of the directory server.	Localhost	Yes
port	Port where the LDAP service is running.	389	Yes
securityProvider	SecurityProvider is the provider url which the SSL context should use. Example: ldaps://myhost.example.com:636	None	No
SSLContextProtocol	Key management protocol. Example: SSLl	None	No, required only if useSSL or startTLS is selected.

Table 12-2 (Cont.) JNDI Connection Pool Properties for the LDAP Adapter

Property Name	Description	Default	Required
`trustAll`	If set to true, the client blindly trusts any certificated that the server might present.	ture	No, required only if useSSL or startTLS is selected. Set this to false if server certificate has to be validated with truststore.
trustStorePassword	Specifies the password needed to access the trust store contents.	None	No
`trustStorePath`	Specifies file system path to the trust store file that the client uses for server authentication.	false	No. required only if SSL or startTLS is used.
`trustStoreProvider Name`	The Java provider name. Example: JKS	None	No, required only if SSL or startTLS is used.
`UseSSL`	Specifies if a secure channel must be established	False	No
`useStartTLS`	Specifies if startTLS might be turned on for secure communication later on within a plain non-ssl connection.	False	No, do not set this to true when useSSL is set to true. This need to be enabled when non-ssl plain port is used for server connection.

Note: Replace the default values with the actual values that you got from the supported target directory server installations and redeploy LdapAdapter with the updated deployment plan.

Outbound Operations

The LDAP Adapter supports the following operations outbound from the LDAP Adapter.

- Add
- Delete
- Modify
- ModifyDN
- Compare

- Search
- DSML

Note: The LDAP Adapter does not provide any transaction support for outbound operations.

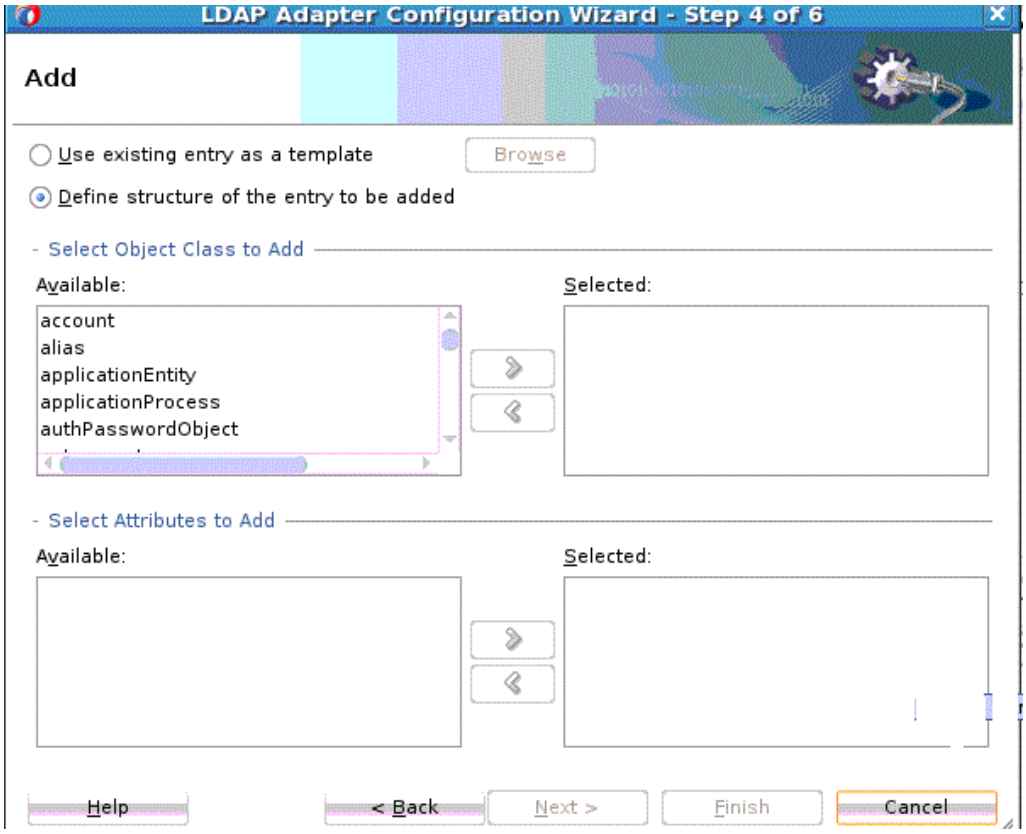
Add Operation

The Add operation adds an entry with a given DN and a set of attributes. It returns an error if the entry to add already exists, or if the given operation could not be completed.

To use the LDAP Configuration Wizard Add operation:

1. Select the Add Operation from the Operation Type Screen. The **LDAP Adapter Configuration Wizard Add Screen** appears.

Figure 12-8 The LDAP Adapter Configuration Wizard Add Screen



2. Select either **Use existing entry as a template**, or **Define structure of the entry to be added** (the default). If you choose **Use existing entry as a template**, click the rectangle to the right to browse for an existing template. If you select **Use existing entry as a template**, all object classes and attributes of the existing entry are used to define the add request. You can add/remove objectclasses/attributes to modify the add request format.

3. In the **Select Object Class to Add** section, the list on the left displays all the Object Classes to add. Select an object class to add in the column on the left and move to the right hand column. You can select one or more object classes. On **Add**, the

selected object classes are displayed in the right. You can select object classes you want to remove by selecting and clicking the `Remove` arrow button for that class.

4. You can select more attributes from the **Attributes** list. You can add mandatory attributes of the selected object classes to the selected attributes automatically. When you select of an item from the selected object classes, attributes belonging to that object class is displayed in the available attributes item list.

   Attributes that do not have any of the corresponding object classes included are automatically removed. You can remove attributes using the remove arrow button.

5. The **Add Suppress Entry Screen** appears. Use the **Add Suppress Entry Screen** to ensure that if an entry exists, and if you are trying to add the entry again, you can suppress such errors.

6. Click **Next**.

7. The **Finish** screen displays the names of the LDAP Service WSDL, xsd and jca files.

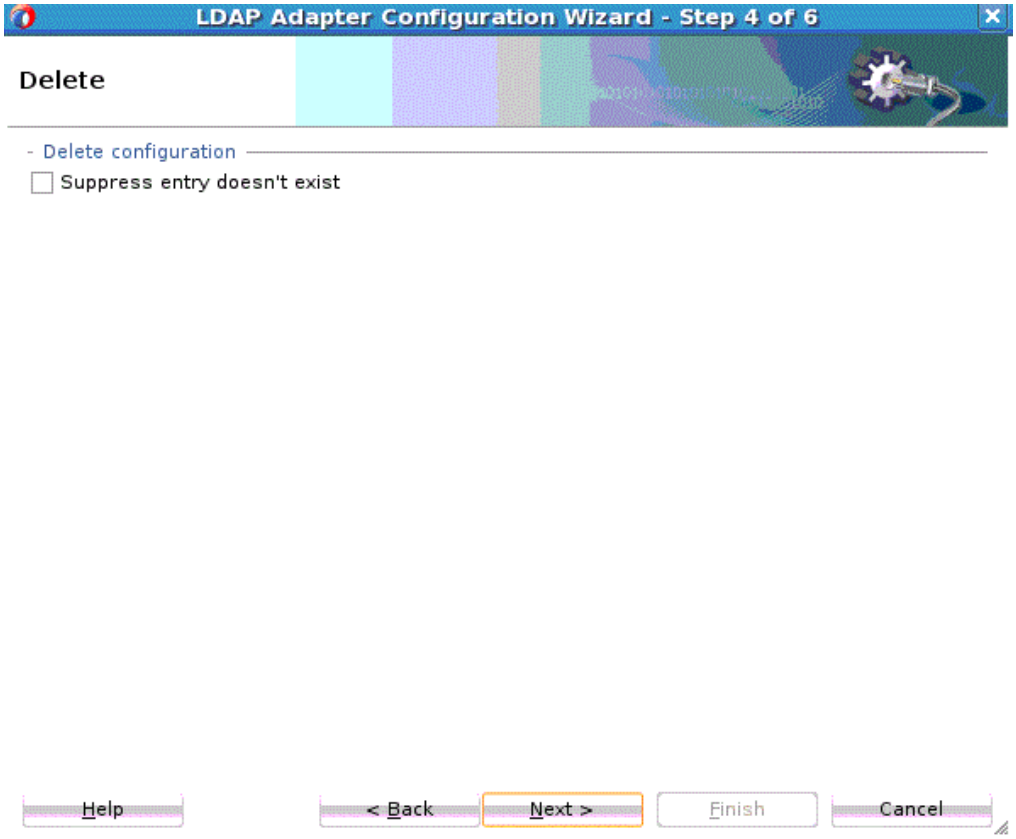
Delete Operation

The LDAP **Delete** operation deletes an entry with a given DN.You can optionally add controls along with the LDAP Delete request. The Delete operation returns an error if the given entry cannot be deleted or does not exist. It also returns an error if a critical control attached with the request message is not supported by the target Directory Server.

To use the **Delete** operation:

1. Select the **Delete** Operation from the **Operation Type Screen**. The **Delete (Suppress Entry) screen** appears.

Figure 12-9 The LDAP Adapter Configuration Wizard Delete Screen

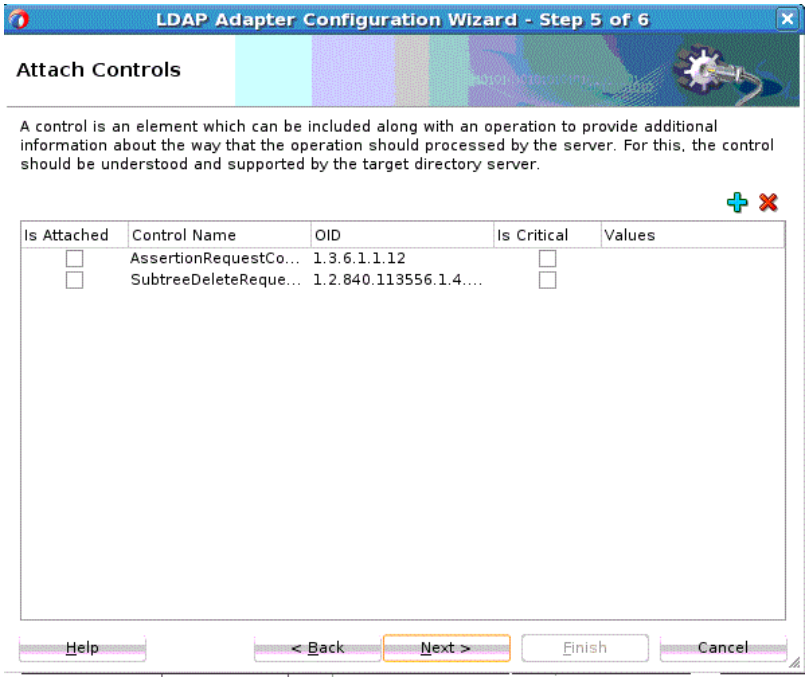


2. The **Delete Screen** appears. Select the **Suppress entry doesn't exist** checkbox if you do not want errors to appear if the entry to delete does not exist. Click **Next**.

3. The **Attach Controls** screen appears. See Figure 12-10.

   The **Attach Controls** Screen provides you the ability to attach controls to provide additional constraints to the **Delete** operation. You can either Attach Controls or Click Next. If you want to attach controls, that are supported by LDAP Adapter and the target directory server, select the **Is Attached** check box, to include the control along with the delete request operation. If you need more information about controls, refer to the controls section of LDAP Adapter configurations.

Figure 12-10 The LDAP Configuration Wizard Delete Attach Controls Screen Showing Delete Controls Added



4. Click **Next**. The **Finish** screen displays the names of the LDAP Service WSDL, xsd and jca files.

Modify Operation

The LDAP **Modify** operation alters an existing entry in the Directory Server.

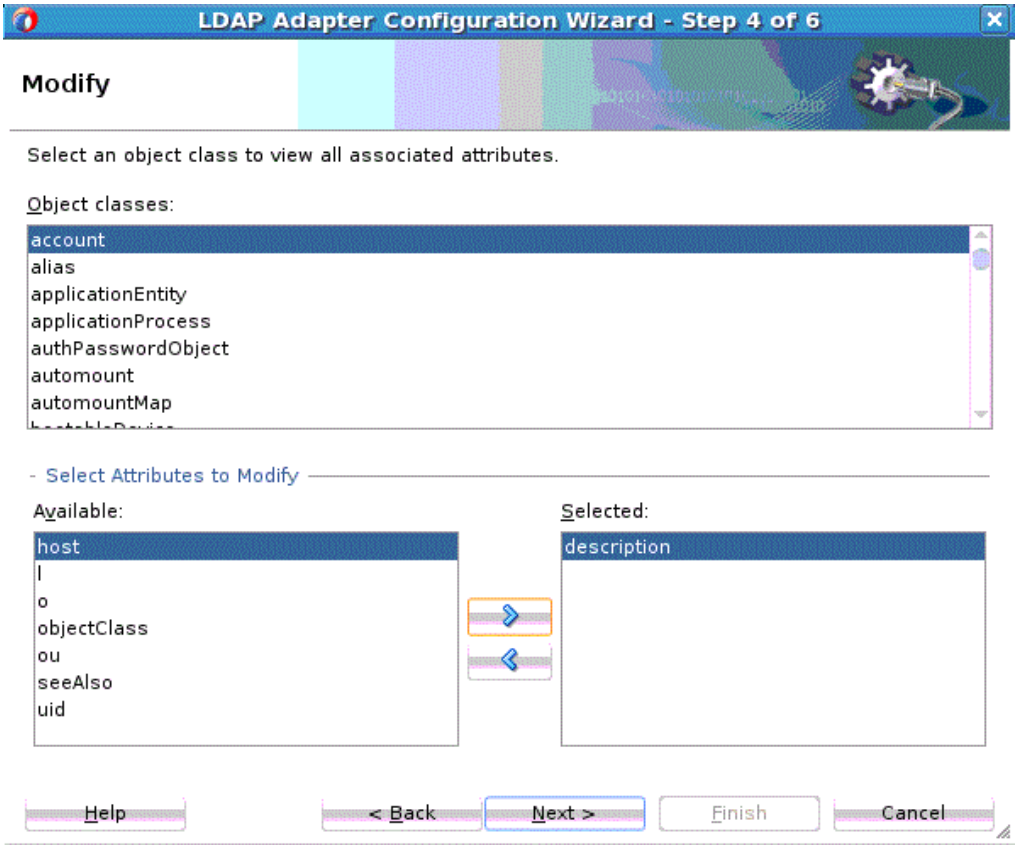
Types of alterations performed with **Modify** include: **Add**, **Delete**, and **Replace**.

- **Add**: Creates the attribute if it does not exist, and adds the specified values. If the attribute already exists, the specified value is added to the attribute values. The Add operation must contain at least one value, and all values of the attribute must be unique.
- **Delete**: Deletes specified values from the attribute. If no values are specified, or if all existing values of the attribute are specified, the attribute is removed. Mandatory attributes cannot be removed.
- **Replace**: Creates the attribute if necessary, and replaces all existing values of the attribute with the specified values. If you want to keep any existing values of a multi-valued attribute, you must include these values in the replace operation. A replace operation with no value removes the entire attribute if it exists, and is ignored if the attribute does not exist.No changes are made to the directory unless all the operations succeed. If the connection fails during a modification, it is unpredictable if the modification occurred.

To use the LDAP Configuration Wizard **Modify** operation:

1. Select the **Modify** Operation from the **Operation Type Screen**. The Modify Operation screen appears.

Figure 12-11 The LDAP Configuration Wizard Modify Select Object Classes Page



2. Select from the list of **Object classes** on this page. From the list of **Available attributes to Modify**, select attributes that can be modified, and move them to **Selected**. You can also remove them from selected by using the left arrow key. Click **Next**.

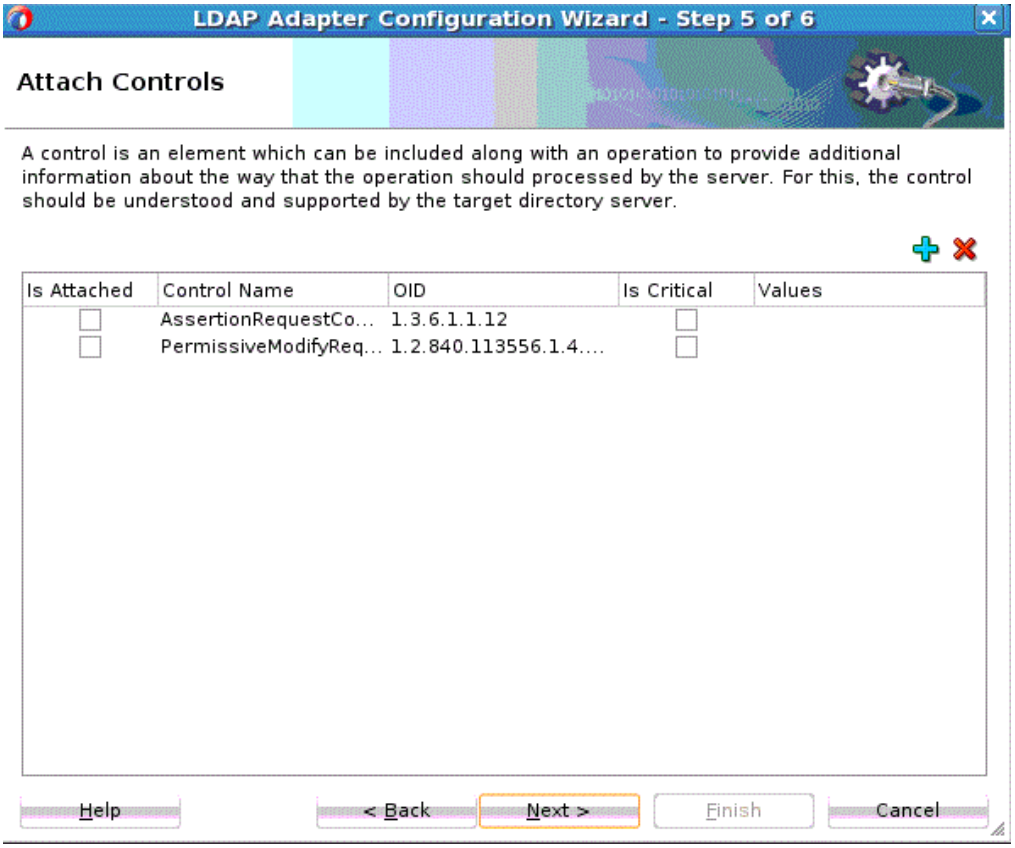
3. The **Attach Controls** Page appears. See Figure 12-12.

   Use the green plus to add custom controls, or controls provided by the target Directory Server. An example is shown below. Select controls needed, and provide values by entering in the **Values** column.

Note:

Consult your target directory server documentation for supported controls. Check the relevant operation in this chapter before attaching a control. The LDAP Adapter might not support all controls supported by the target directory server.

Figure 12-12 The LDAP Configuration Wizard Modify Attach Controls Page



4. Click **Next**. The **Modify Finish** screen displays the names of the LDAP Service WSDL, xsd and jca files.

ModifyDN Operation

The LDAP **ModifyDN** operation can be used to change the distinguished name of an entry in the Directory Server. It can also alter the RDN of the entry and/or it can move the entry below a new parent. If the target entry has subordinate entries, then it may be used to move or rename that subtree.

To use the **ModifyDN** operation:

1. Select the **ModifyDN** Operation from the **Operation Type** screen.The **LDAP Adapter Configuration Wizard ModifyDN Screen** appears. See Figure 12-13.

Figure 12-13 The LDAP Adapter Configuration Wizard ModifyDN Screen

2. Select the **Delete old RDN** checkbox if you want to delete the old RDN. If you want to modify the DN, but do not want to delete the old RDN, do not check this box. Click **Next** after either choice.

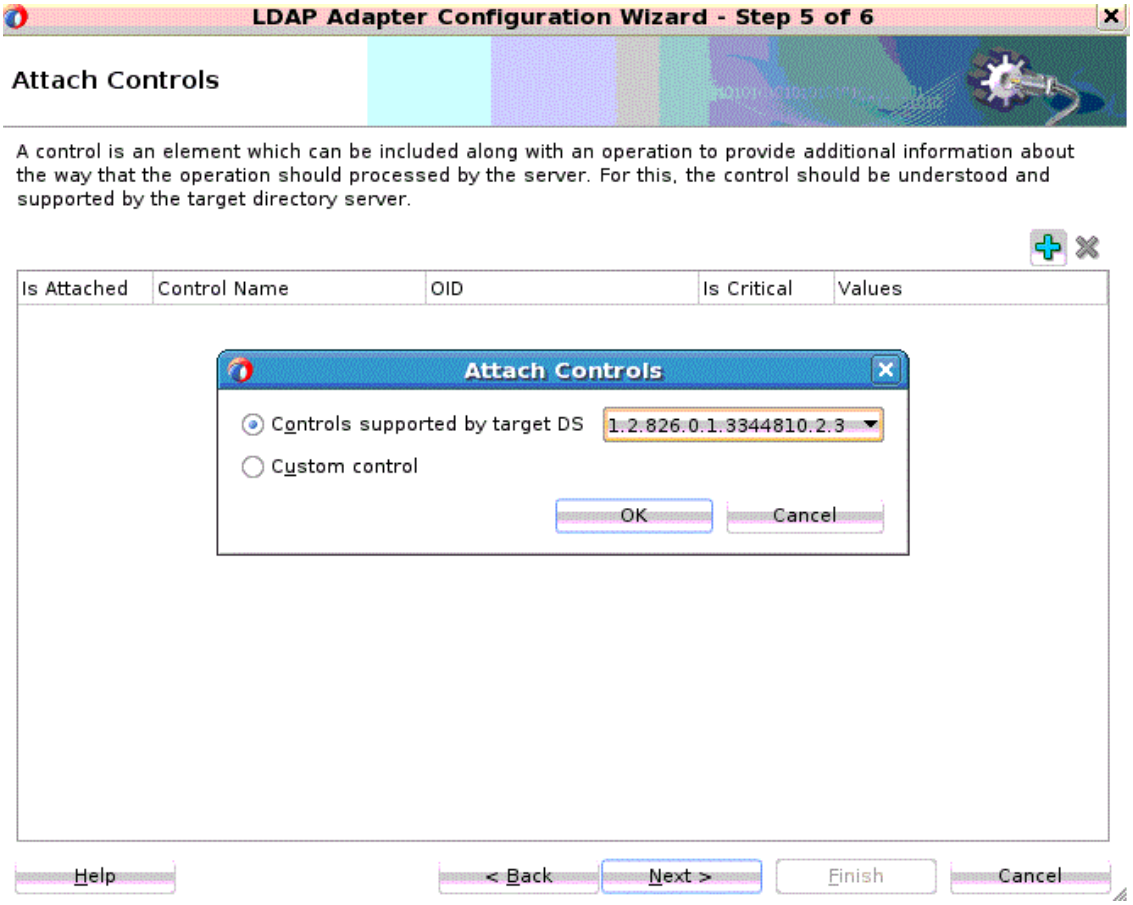
3. The **Attach Controls** page appears if you had chosen either choice in Step 2. See Figure 12-14

   The **Attach Controls** screen provides you the ability to choose and attach controls to provide additional constraints to modifying the DN.

   Available controls are specific to your Directory Server. If you choose to attach controls, select the green plus symbol to add controls. Select controls you need, and provide values by entering them in the Values column. Consult your target Directory Server documentation for supported controls. Check the relevant operation in the "controls section of LDAP Adapter configurations" before attaching a control. The LDAP Adapter might not support all controls supported by the target directory server.

4. When this chooser appears, you can either add controls that are supported by your target Directory Server, or you can add custom controls.

Figure 12-14 LDAP Adapter Configuration Wizard Attach Controls Chooser Screen

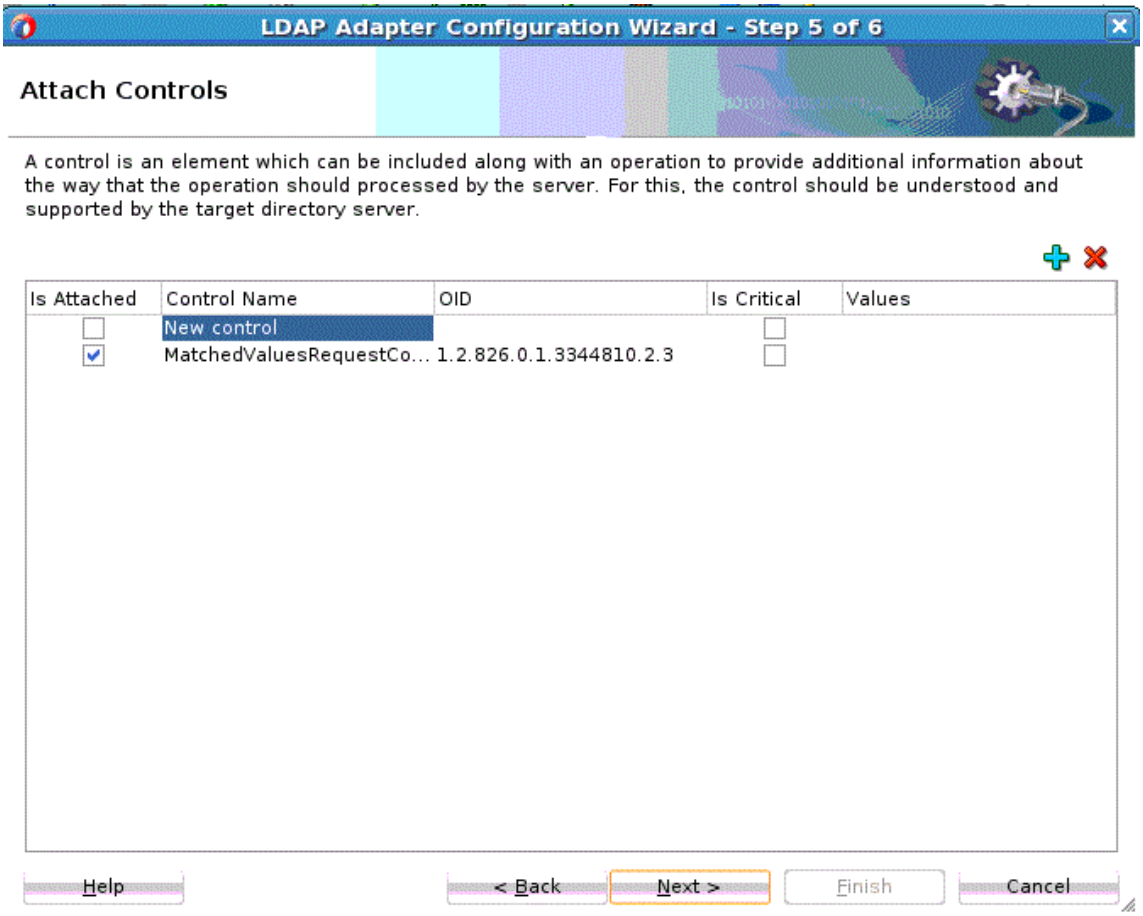


5. Choose either controls supported by the target Directory Server or a Custom Control.

   If you have chosen to attach controls, a page similar to the following appears.

   See Figure 12-15 which shows both the custom new control and a control that matches those available on the Directory Server ("MatchedValues...") added as controls.

Figure 12-15 LDAP Configuration Wizard Attach Controls Page



6. Click **Next**.
7. The **Finish** screen displays the names of the LDAP Service WSDL, xsd and jca files

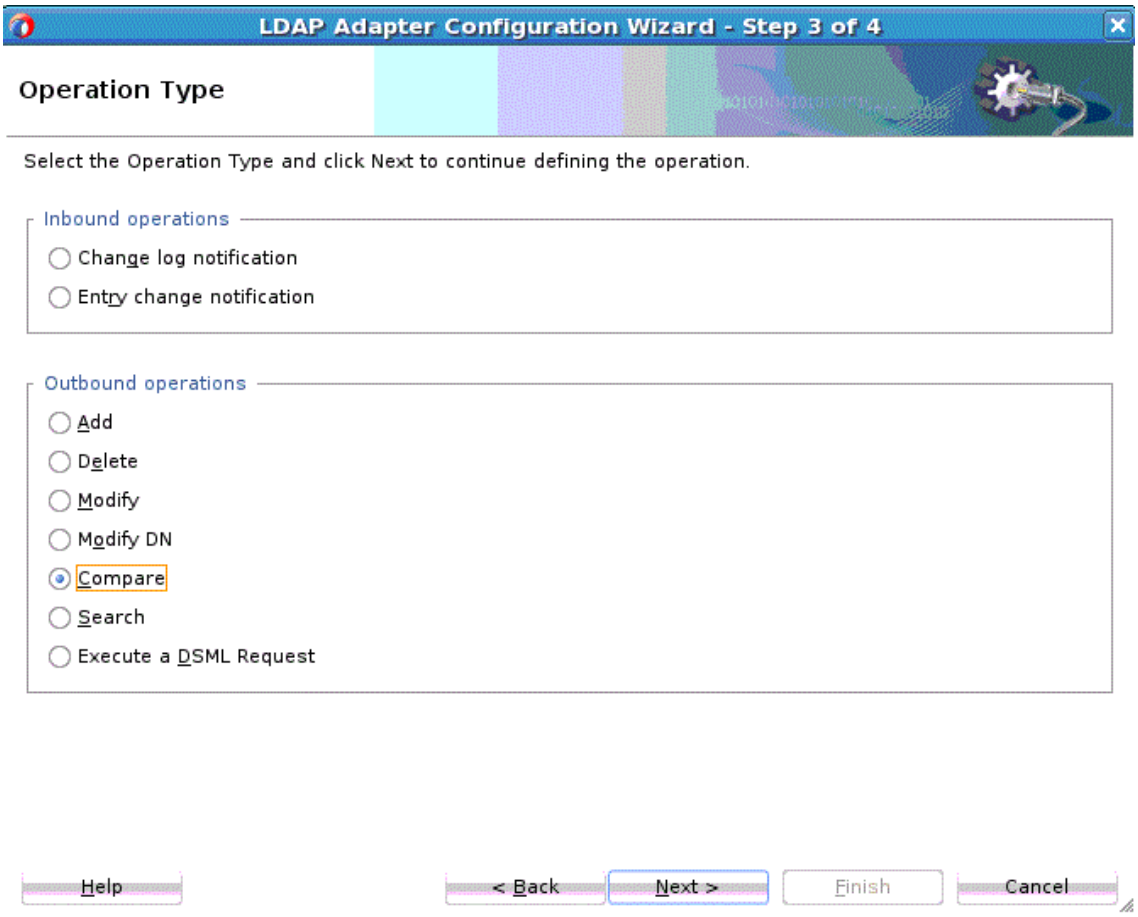
Compare Operation

The LDAP **Compare** operation determines whether a specified entry contains a given attribute value.

To use the LDAP Adapter **Compare** Operation:

1. Select the **Compare Operation** from the LDAP Adapter **Operation Type** screen.

Figure 12-16 LDAP Configuration Wizard Operation Type Screen, showing Compare Selected



2. Click **Next**.
3. The **Finish** screen displays the names of the LDAP Service WSDL, xsd and jca files.

Search Operation

The LDAP Adapter Search operation can be used to identify entries in the Directory Server that match a given set of criteria. It may return zero or more entries, and also zero or more referrals. The Search operation includes the following criteria.

- The base DN, which specifies the location in the DIT in which to perform the search.
- The search scope, which specifies the scope of entries at or below the base DN to consider when processing the search.
- The dereference policy to use if any aliases are encountered during processing.
- The size limit, which specifies the maximum number of entries that should be returned from the search (or –1 for no limits on the search results).
- The time limit, which specifies the maximum length of time in seconds that the server should spend processing the search (or zero if there should not be a maximum number of entries).

- The types only flag, which indicates whether the entries returned should include attribute types only or both types and values.
- The search filter, which specifies the criteria to use to identify matching entries.
- The search attributes that indicate which attributes should be included in matching entries
- Controls you can attach along with the search request, and return an error if a critical control attached with the request message is not supported by the target Directory Server.

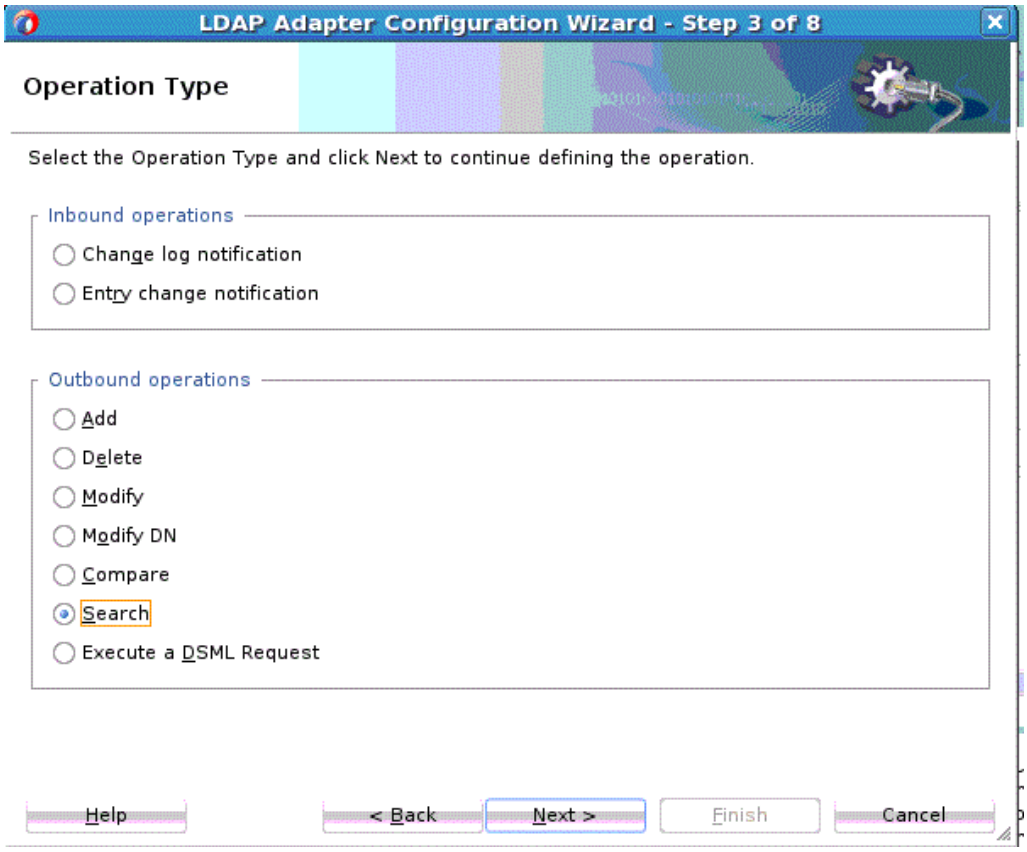
Note:

In the current version, binary types are not supported and attribute values are always encoded as utf8 Strings.

To use the outbound Search operation:

1. Select the **Search** operation from the **Operation Type** screen. See Figure 12-17.

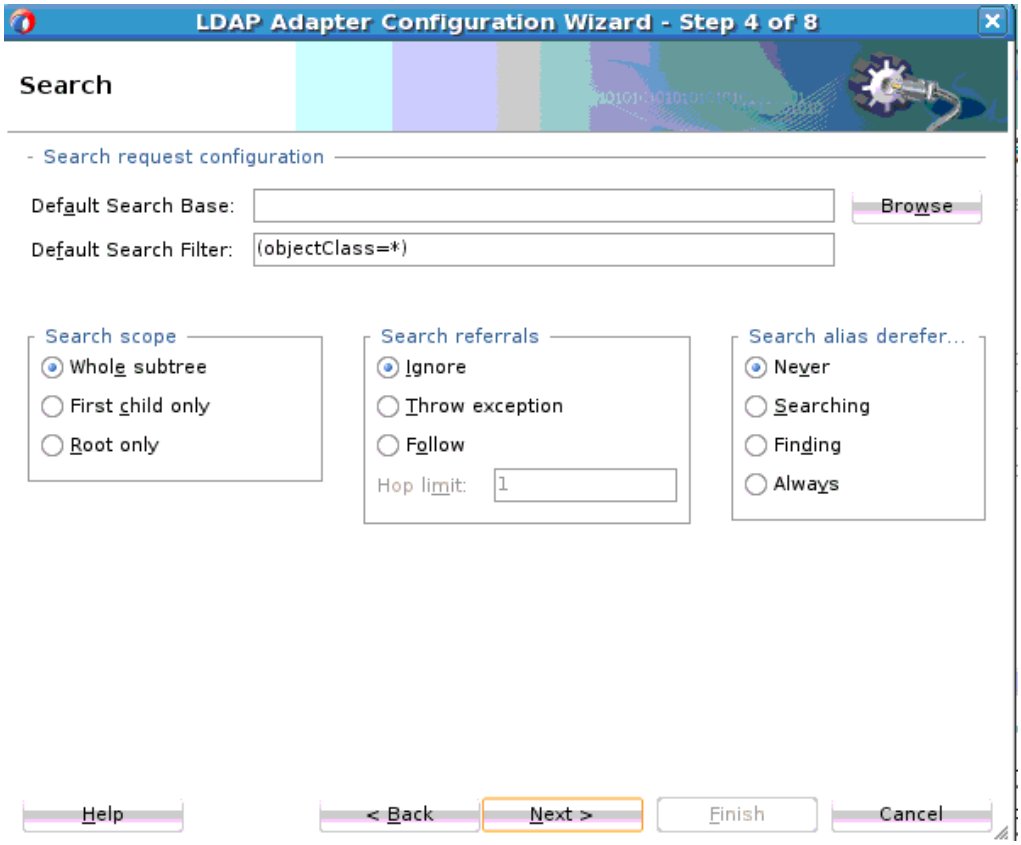
Figure 12-17 LDAP Configuration Wizard Operation Type Page with Search Selected



2. The **LDAP Configuration Wizard Search Configuration Page** appears. See Figure 12-18.

   Use the **Search (Request Configuration) Page** to provide search information relating to search scope, search referrals, and search alias dereferencing.

Figure 12-18 The LDAP Adapter Configuration Search Configuration Wizard Search Request Configuration Page



3. At the top of the screen, above the other sections, indicate the default search base and default search filter. Provide the search base in the **Default Search base** text box. Note that Default Search BaseDN and Default Search Filter are considered only when these attributes are not defined in the `LdapSearchRequest`.

   The following is added in Search Operation schema xsd.

```
<element name="searchRequest">
    <complexType>
        <sequence>
            <element name="baseDN" type="string" default="dc=oracle,dc=com"/>
            <element name="searchFilter" type="string" default="(objectClass=*)"/>
        </sequence>
    </complexType>
</element>
```

   Select the **Browse** button to open the directory information tree browser dialog, which can help you specify a search base. See LDAP Browser for more information.

   This is the base DN that defines the source of events. Selecting this DN means you want to notify events under it.

4. In the **Default Search filter** text box, supply a **Search** filter to filter the entries that can be returned for a search operation. The filter can be a string representation of a search filter. For more information on LDAP search filters, see LDAP Search Filters in "Oracle Fusion Middleware Reference for Oracle Directory Server Enterprise Edition" documentation.

5. **Search scope** defines the Search scope within the tree. This has to be defined at service deployment to control any malignant search request from controlling server resources.

   Check with your Directory Server before using this option. Scope can be:

   - **Whole subtree**-Select this to indicate a scope of all entries within the subtree.
   - **First child only**-Select this to indicate a scope of one level below the search base only.
   - **Root only**-Select this to indicate one event only at the search base.

6. The next section is **Search Referrals**. A referral is a response returned to the LDAP Adapter that instructs the Adapter to contact another LDAP server to perform the search operation the LDAP Adapter requested.

   In the Referrals section, select one of the following:

   - **Ignore**-Select to have the LDAP Adapter ignore the referral and returns whatever search operation on the local sever returned.
   - **Throw Exception**-Select to have the LDAP server not chase the referral, nor does it complete the search by returning results from the local server. Instead, the Server throws an exception on which an action can be initiated.
   - **Follow**-Select to follow a referral server. These are alternate locations where additional entries may be searched/matched as part of a search request. By default, the LDAP Adapter ignores all referral suggestions unless explicitly asked to follow or throw an exception. (This search can potentially take longer duration and should be a non-blocking activity.)
   - **Hop Limit** - Select to indicate number of alternate locations participating in search referral. This should be a positive integer. The default value is 1.

7. The next section, which deals with the LDAP server behavior for handling alias entries while processing the search, specifically with dereferencing alias entries, is **Search alias dereferencing.** Select one of the following:

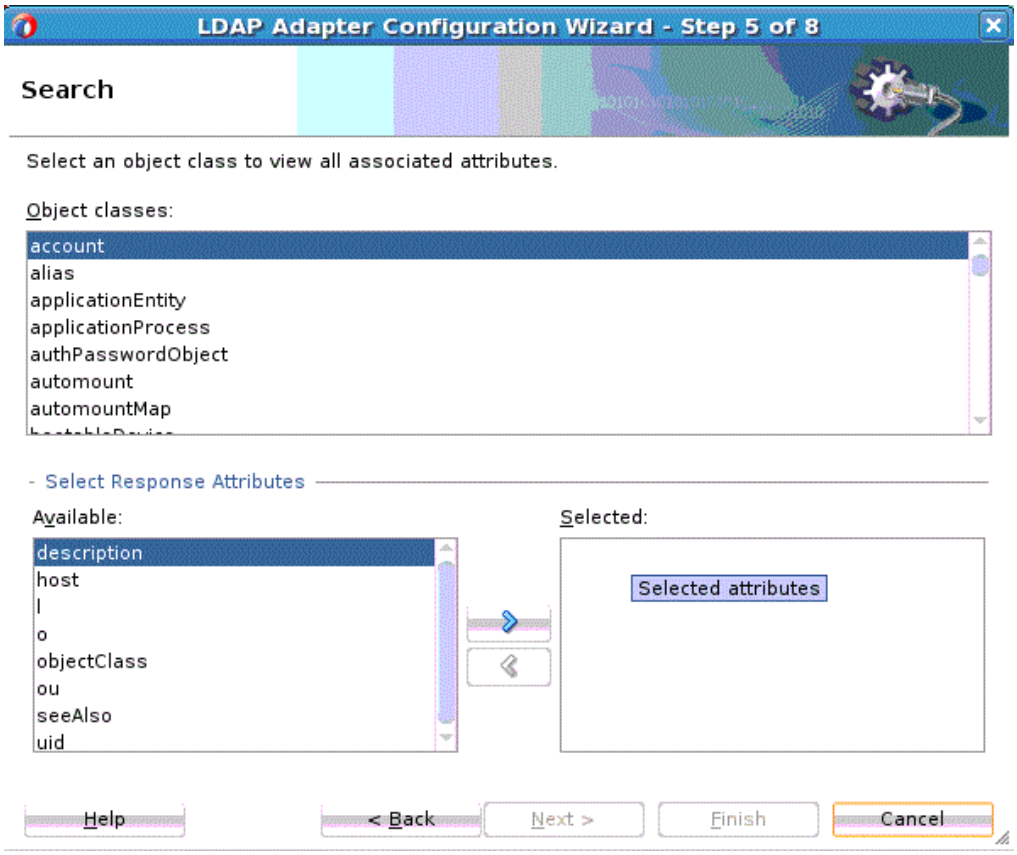
   - **Never**-Select this to indicate the server should not dereference any aliases that it encounters.
   - **Searching**- Select this to indicate that the server should dereference any aliases that it might encounter while examining candidate entries, but it should not dereference the base entry if it happens to be an alias entry.
   - **Finding**-Select this to indicate that the server should dereference the base entry if it happens to be an alias entry, but it should not dereference any alias entries that may be encountered while examining candidate entries.
   - **Always**-Select this to indicate that the server should dereference the base entry if the entry happens to be an alias entry, and should dereference any entries that might be encountered while examining candidates.

   Click **Next** to proceed to the next Page or **Finish** to finish the Search Operation configuration.

8. If you clicked **Next**, the **LDAP Search (Response Attributes) Page** is displayed. See Figure 12-19.

Use the **Search (Response Attributes) Page** to configure the Search operation response parameters, using the object classes available on the target LDAP Directory Server.

Figure 12-19 The LDAP Adapter Configuration Wizard Search (Response Attributes) Page



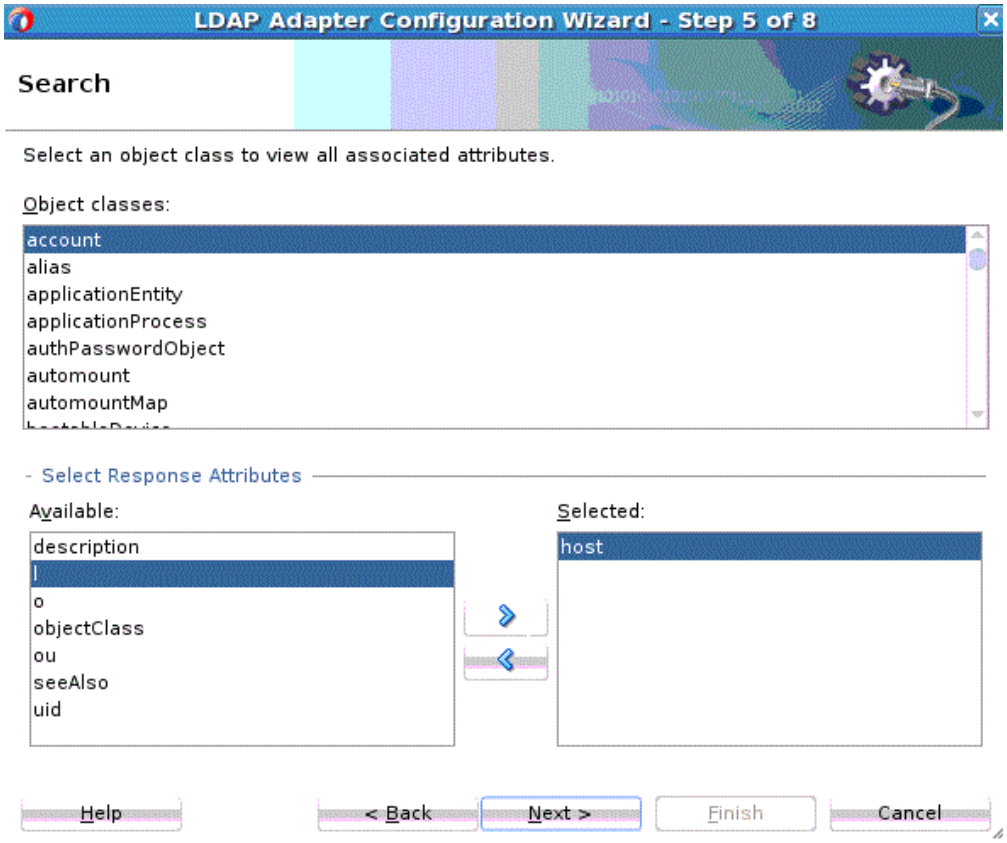
9. Select an object class or classes from the **Available** object classes list box, which displays all object classes available in the schema of the connected target Directory Server.

10. All attributes belonging to the selected object class or classes are displayed in the **Available** attributes list box. Select from Available attributes and click the arrow to move an available attribute or attributes to **Selected**.

    If you selected more than one object class, the union of all the attributes are displayed; common attributes are not displayed twice.

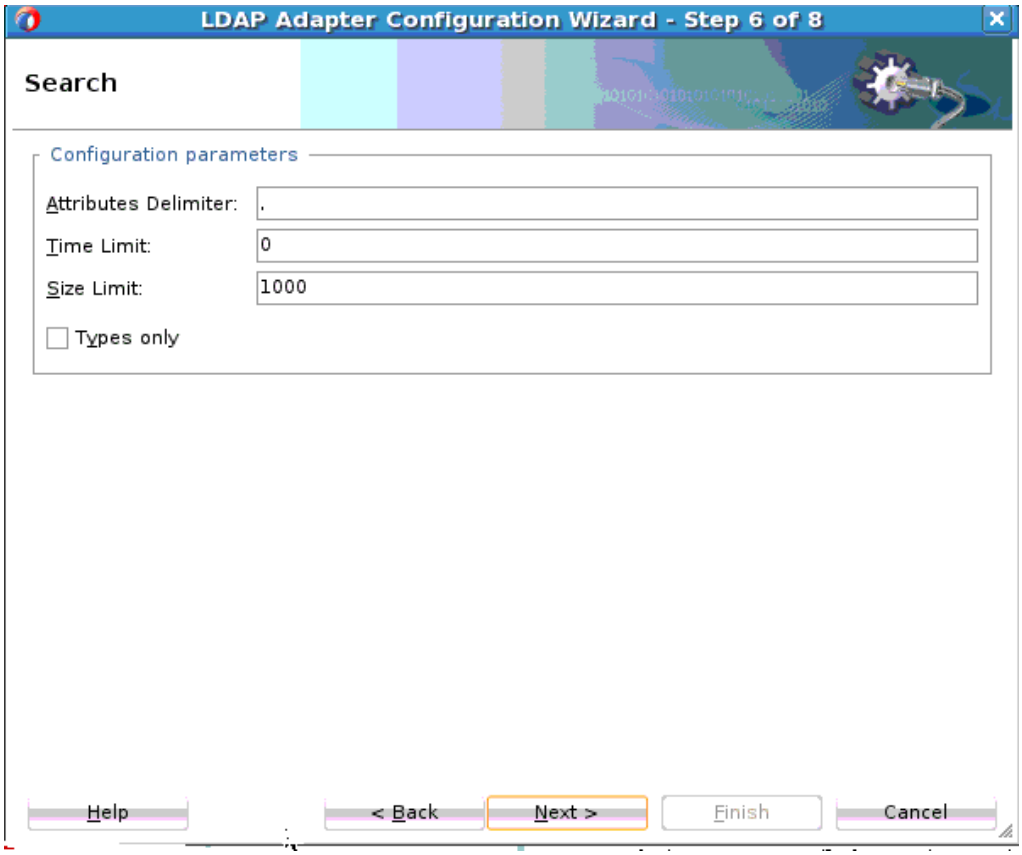
    Figure 12-20 shows an example.

Figure 12-20 LDAP Adapter Configuration Wizard Search (Response Attributes) Page Example Use



11. Click **Next**. The **Search Configuration Parameters Page** appears. See Figure 12-21.

Figure 12-21 The LDAP Adapter Configuration Wizard Search Configuration Parameters Page



12. Indicate the following configuration parameters on this page:

    - **Attributes Delimiter**-Provide a string literal value to delimit the attributes selected by the user.
    - **Time Limit**-Maximum time the server should wait before returning the results. This is an integer value in seconds with default value as 0 for no time limit.
    - **Size Limit**-Maximum number of events or entries that could be returned in search results. This is an integer value with the default value as 1000. Set –1 for no limits on the number of events or entries in search results.
    - **Types only**-Select this checkbox if you want to return only the name of the attributes and not both names and values of attributes.

    Click **Next** to proceed to the next page.

13. The **Attach Controls Page** is displayed. Here you can attach controls to provide further constraints to the LDAP Search Operation. Use the plus to add custom controls, or the controls provided by the target Directory Server. Attach controls that are needed, and provide values by entering in the **Values** column.

> **Note:** Consult your target directory server documentation for supported controls. Check the relevant operation in the "controls section of LDAP Adapter configurations" before attaching a control. The LDAP Adapter might not support all controls supported by the target directory server.

14. The **Finish** screen displays the names of the LDAP Service WSDL, xsd and jca files.

DSML Operation

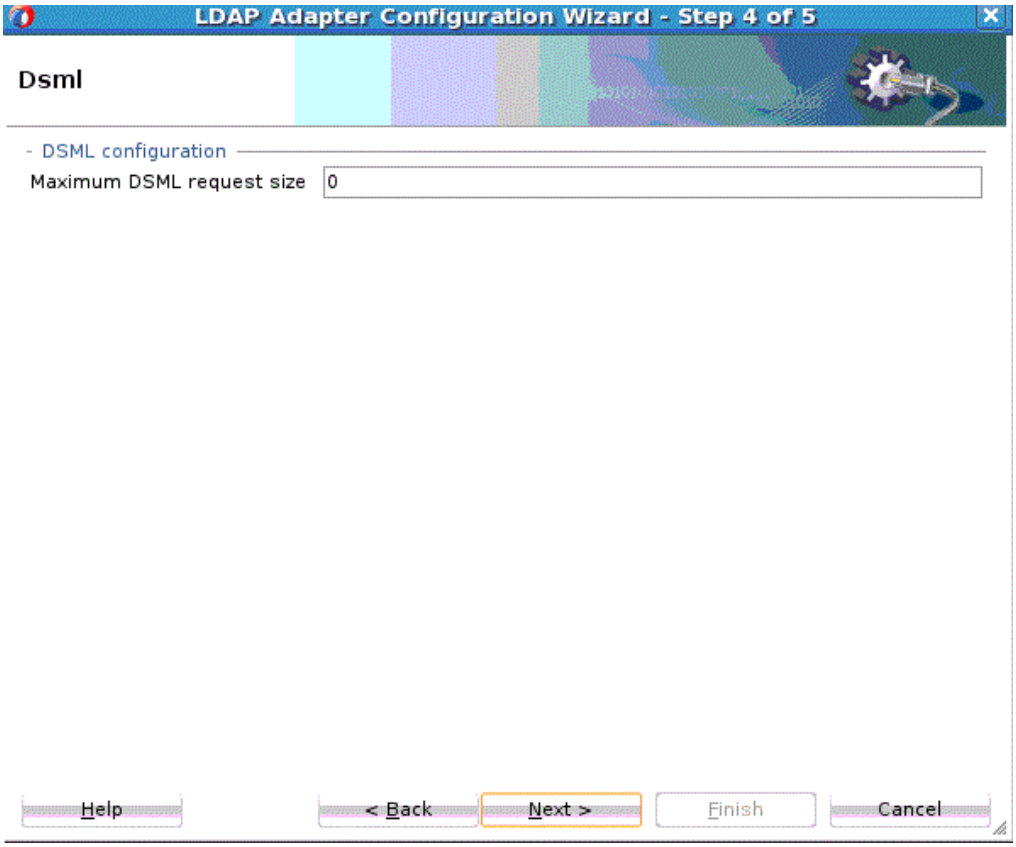
DSML operations express LDAP requests and responses as XML document fragments. Use the LDAP Adapter **DSML** screen to specify an execute a DSML request and to optionally enter a Maximum request size for that DSML request on the LDAP Operations Screen.

To use the DSML Execute Request:

1. Specify **DSML** on the LDAP Operation Type Screen.
2. The **DSML** Screen appears. See Figure 12-22.

   On this screen, optionally enter the **Maximum DSML Request Size**. Set this property if you want to control the maximum number of requests that can participate in a DSML batch request. The incoming DSML request is executed if and only if the total number of contained requests is less than this number.

Figure 12-22 The LDAP Adapter Configuration DSML Configuration Page



3. Click **Next**. The **DSML Operation Finish** screen appears, showing the LDAP WSDL, jca and xsd files.

Inbound LDAP Adapter Features

The LDAP Adapter supports publishing events from a Directory Server. Specifically, it supports one way inbound interactions, asynchronous event publishing from the directory server through the asynchronous notification model.

The LDAP Adapter listens for events such as addition, deletion, or modification of an entry in the directory information tree and publishes such events. The LDAP event consumer singleton connects and listens for changes on the directory server according to a chosen strategy and processes/publishes such events to LDAP Adapter clients.

By default, the LDAP Adapter publishes events in the order they are received from the directory server.

The LDAP Adapter provides two type of inbound activities:

- LDAP Adapter Entry Change Notification
- LDAP Adapter Change Log Notification

LDAP Adapter Entry Change Notification

Entry Change Notification enables you to obtain entries that are created or modified on a source directory server. The result contains the modified entry along with information about the type of modification.

This mechanism is based on a timestamp-based search; it does not require any external configuration and is supported by most Directory Servers. With time-based search, a search is done for all entries after a base time stamp. The base time stamp is updated for subsequent timestamp-based searches.

In search, the LDAP Adapter returns results up unto the latest time stamp, it then updates the base time stamp to the latest time stamp. The time interval you configured indicates when the next iteration of the timestamp-based search for all events next occurs.

Time stamp-based search only provides notifications of added or modified entries. With Entry Change Notification, there is no provision to find out deleted entries. This mechanism only notifies about the changed entry that is, it indicates if an entry has been changed, and does not divulge details of the changes that occurred.

There is no way to differentiate between a modify and modifyDN operation as there is no provision to find out deleted entries.

Specifically, Entry Change Notification does not provide notification when an entry enters or leaves the scope of a result set due to a modifyDN operation or when a Modify operation acts on a attribute value that is used in a search filter.

Time-stamp based search does not work with multi-master setups of the Directory Server because the time stamp is generated using the clock on each of the servers and can therefore appear in a non-linear order on a given replica.

Because Entry Change Notification uses Timestamp-based search, which is based on the operational attributes `createTimestamp` and `modifyTimestamp`, these attributes should be available, indexed and functional in the directory server for timestamp based search to work properly.

You might see the `Unavailable Critical Extension` exception in the LDAP Adapter client logs, if any of these attributes are not indexed and functioning correctly.

Refer to your directory server documentation to help ensure these attributes are indexed properly, so they are available during the time stamp search.

Note:

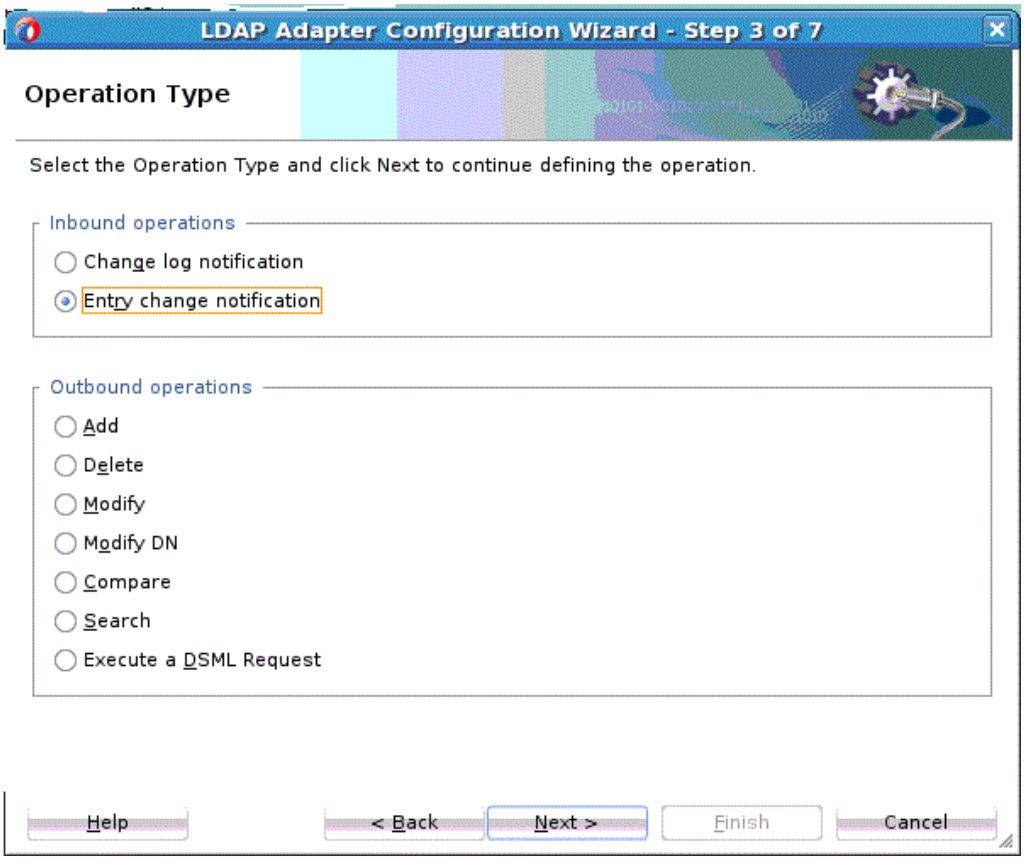
PageSize configuration does not work when connecting to the Oracle Internet Directory server and should not be used with that server. If configured, the number of events happening over the pageSize limit are not processed.

LDAP Adapter Entry Change Notification Configuration Wizard Flow

Follow these steps to configure Entry Change Notification.

1. Once you create a connection (see Configuring the LDAP Adapter,) click the **Next** button. The **Operation Type** screen appears. See Figure 12-5. Select **Entry Change Notification**.

Figure 12-23 LDAP Adapter Configuration Wizard Operation Type Screen



2. Click **Next**. The **Entry Change Notification** screen appears. See Figure 12-6.

   Here, you can define the scope of events to which the adapter must listen.

3. On the **Entry Change Notification** screen, click the **Browse** button to open the directory information tree browser dialog. See LDAP Browser for more information.

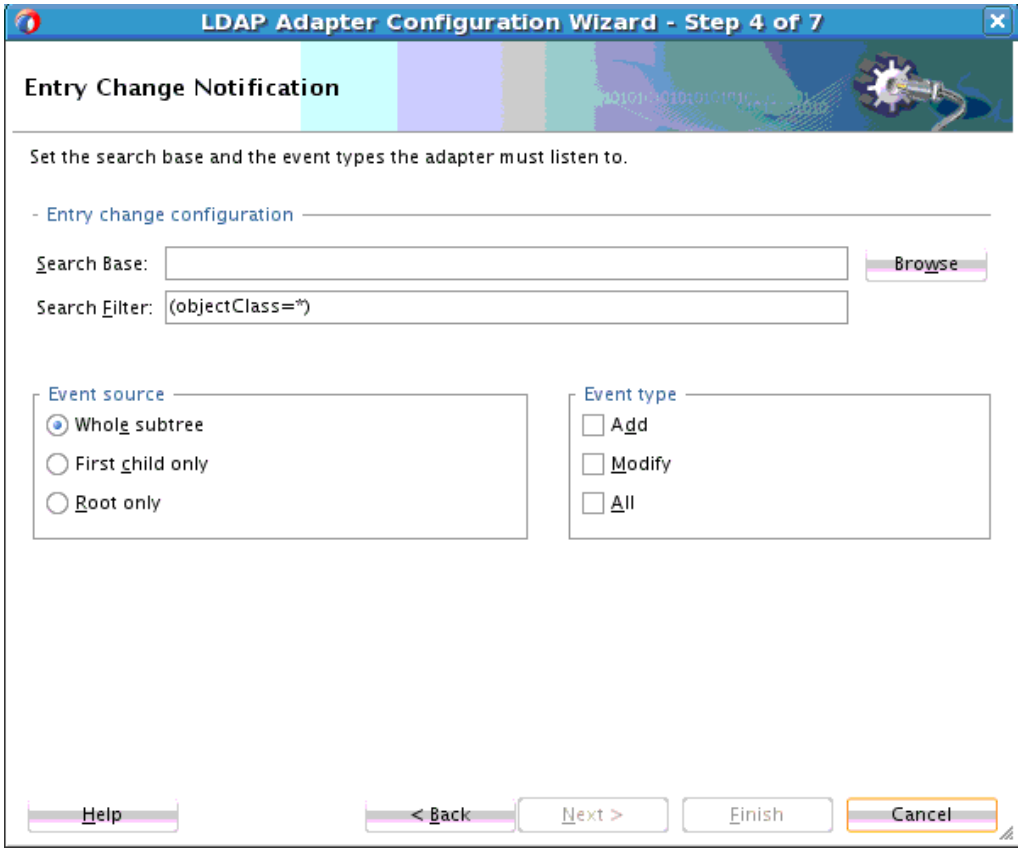
4. The **Search Field** area enables you to supply a search filter in string format for what is to be searched. For more information on LDAP search filters, see Oracle Fusion Middleware Reference for Oracle Directory Server Enterprise Edition.

5. Use the **Event Source** radio buttons to choose among the following selections (A search scope defines how deep to search within the search base_

   - **Whole subtree**-indicates a search of the base object and the entire subtree of which the base object distinguished name is the topmost object.
   - **Subtree only**--search only this subtree
   - **First child only**--only search from the root to the first child.
   - **Root only**--search the root of the LDAP hierarchy only.

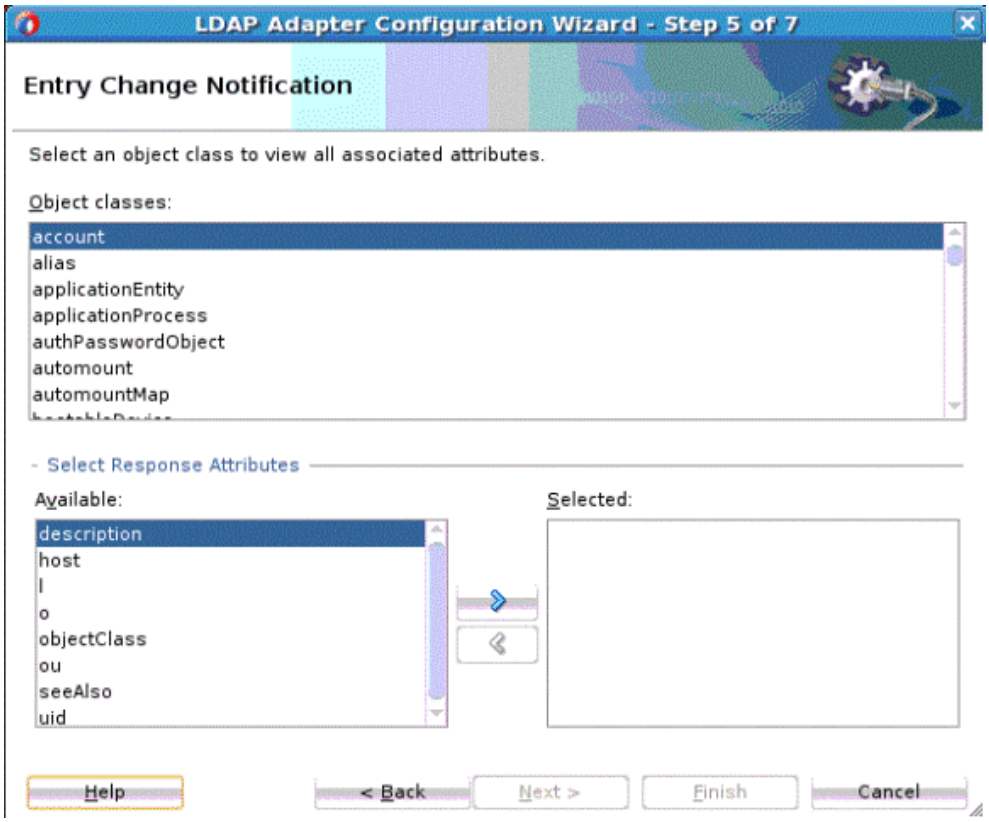
6. Choose an **Event Type**: **Add**, **Modify** or **All**.

Figure 12-24 LDAP Adapter Configuration Wizard Entry Change Notification Screen



7. Click **Next**. On the next screen, you can select the attributes to return as part of the changed entries. See Figure 12-7.

Figure 12-25 LDAP Adapter Configuration Wizard Entry Change Notification Screen



8. The **Object classes** list box displays all object classes starting with the characters specified in the available object classes' text field. Choose an object class from the list. (Select multiple object classes by CTRL+select).

   All attributes belonging to the object class are displayed in the **Available** Attributes list box below. Same attributes belonging to multiple object classes are displayed only once.

9. Select one or more attributes and use the move buttons (> <) to select/unselect return attributes. (using CTRL+select). The Response Attributes, once selected, do not contain any duplicates. The wizard ignores adding an already-added element.

   Click **Next**. The following screen enables you to provide the attributes' delimiter, polling interval, timeLimit, sizeLimit pageSize and other options.

10. In the **Attributes Delimiter** field provide a regular expression value that can be used to delimit the attributes you select if you cannot use the ',' as delimiter..

11. In the **Polling Interval** field, an integer with default value set to 10 seconds, you can provide the polling interval to indicate the interval before conducting subsequent searches for new events.

12. In the **Time Limit** field provide an integer value in seconds with default value as 0 seconds. This is the maximum time the LDAP server should wait before returning results.

13. In the **Size Limit** field, provide an integer value. This value represents the maximum number of entries returned as part of a search operation. You can also

configure this attribute on the Directory Server as well. The lower of the two values takes effect. The `SizeLimit` value is enforced within a single page. The default value is 1000. A value of 0 or less implies that the server does not enforce a sizeLimit.

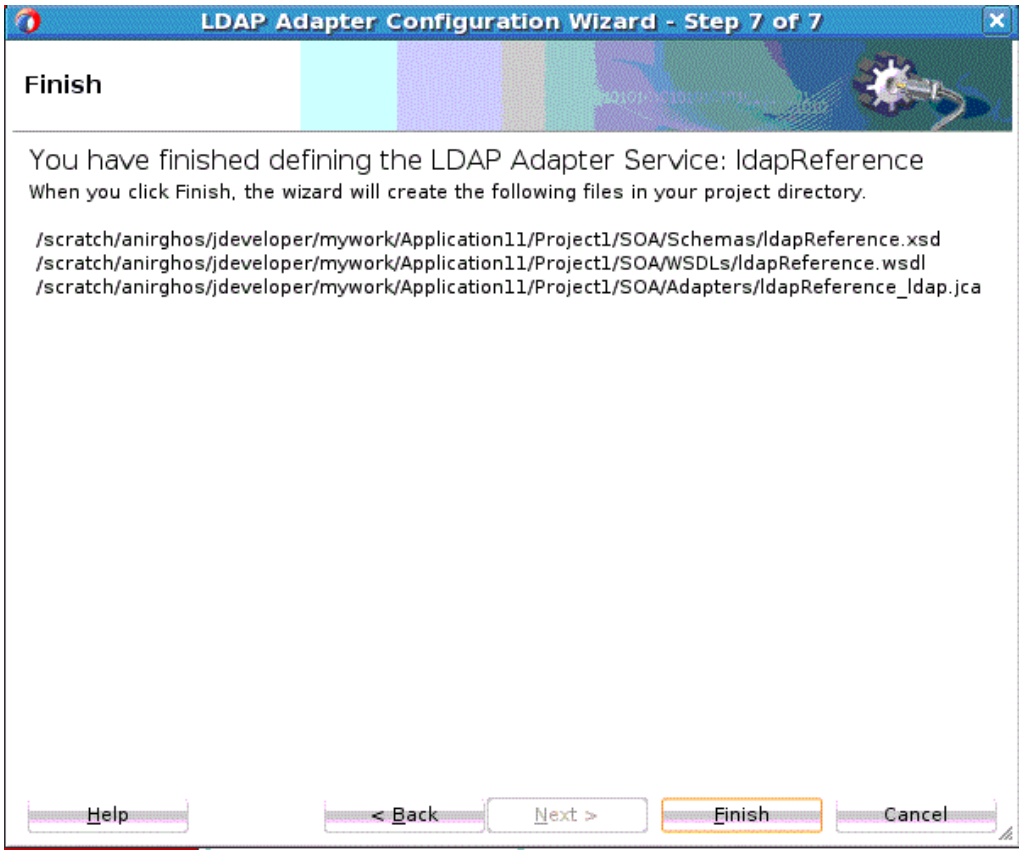
14. Enter in the **PageSize** field the maximum number of events that should be published in a page. The `sizeLimit` attribute implies events within a page and not the total number of entries returned across all the pages in a search. The default value for this value is -1, implying an unlimited `pageSize`.

    If a search is expected to return 12,000 entries and **Size Limit** is set to 1000, a total of only 1000 entries will be returned. In this scenario, if **Size Limit** is set to zero and **Page Size** is set to 500, the search returns all 12,000 entries in pages of 500 entries each, with the last page containing only 200 entries.

15. **Types only** if not checked, the LDAP Adapter returns the names and values of the attributes; otherwise, it returns only the names of the attributes.

16. Click **Next**. The **Finish** screen appears.

Figure 12-26 LDAP Adapter Configuration Wizard Change Notification Finnish Screen



LDAP Adapter Change Log Notification

The Change Log Notification event provides a detailed log of modifications that were done on entries in Directory Server.

The event includes change details and reconstructs the request message that was executed on the source server that led to this change. This event is particularly useful

if the same request is to be replayed on a target server. The Change Log Notification is a very powerful feature in that it provides details on activities.

You can use Change Log Notification to incrementally synchronize a source LDAP directory with other sub-systems. As this mechanism is not part of the LDAP specification, you should check your Directory Server's implementation and the Oracle certification matrix before using this feature, as it might require specific configuration on the source Directory Server before you can use it.

Change Log Notification is not available on a server instance that is configured as either a dedicated directory server or a dedicated replication server.

Change Log Notification LDAP Adapter Configuration Wizard Flow

Follow these general steps to configure Change Log Notification. The steps assume you have performed prior steps in the general flow, and you are now at the **Operations Type** Screen.

1. Select **Change Log Notification** on the **Operations Type Screen**. Click **Next**.
2. If the Directory Server does not support the Change Log notification mechanism, a warning dialog is displayed.
3. The **Change Log Notification Screen** appears.
   - Select the **Notification Strategy** from the options. Notification strategy indicates the change log plugin that is used to query for the change log. This selection defaults to the currently connected Directory Server.
   - If you do not select an existing **Notification Strategy**, select the Custom Plugin option.. When you select this, it opens a textbox where you can provide the class name of their custom changelog plugin, Polling interval, SizeLimit and TimeLimit.
   - Select from the following **Change Type** options:
     - **Add**: Returns the changeNumber , changeType(Add) and DSML add requests that were executed on the source directory server in a specified poll interval.
     - **Delete**: Returns the changeNumber, changeType(Delete) and DSML delete requests that were executed on the source directory server in a specified poll interval.
     - **Modify**: Returns the changeNumber, changeType(Modify) and DSML modify requests that were executed on the source directory server in a specified poll interval.
     - **Modify DN**: Returns the changeNumber, changeType(moddn) and DSML moddn requests that were executed on the source directory server in a specified poll interval.
     - **All**: Returns the changeNumber, changeType(Add,Delete,Modify,moddn) and DSML requests of Add, Delete, Modify, ModifyDN events that were executed on the source directory server in a specified poll interval.
   - **Polling Interval**: Polling time interval in seconds before conducting subsequent searches for new events.

- **Time Limit**: Maximum time in seconds the server should wait before returning results.
- **Size Limit**: Maximum number of events that should be published in one attempt. Default integer value is 1000.
- **Page Size**: Maximum number of events that should be published in a page. Default Integer value is 0 (no paging).

4. Click **Next**. The Finish screen appears.

Change Log Notification Errors

The following table lists Change Log Notification Errors.

Error Name	Description	Status
`InboundConnectionDown`	Inbound adapter service's connection to the Directory Server is down. This is an `LdapConnectionException`	Remote Fault with appropriate error message. Adapter should retry the connection to the Directory Server.
`ChangeLogNotSupported`	Change log mechanism is not supported by the target Directory Server. This is an `LdapValidationException`	Fatal exception and inbound service stops.
`InvalidChangeNumber`	The change number is invalid or not recognized by the Directory Server. Contact your Directory Server administrator. Provide a valid change number. The change number may be invalid or purged and no longer recognized.You can manually provide a new change number in the database. This is an `LdapValidationException`	This leads to a fatal exception because the inbound service does not identify the initial change number and hence cannot start processing.
EndpointActivationError	Validation fails while activating the endpoint due to one or more activation spec errors. This is an LdapValidationException	Binding fault.
SizeLimitExceededException	Specified Size Limit Was exceeded.	.-
TimeLimitExceededException	Specified Time Limit was exceeded.	-

Note:

During runtime, error/status codes that are raised by an LDAP server does not get sent back to the client or displayed in the server.

Entry Change Notification Error Conditions

The following table lists possible errors related to entry change notification through the LDAP Adapter. Included are the error name, a description of the error name and the status the configuration is left in after the error is reported, and retryability.

Table 12-3 Entry Change Notification Error Conditions

Error Name	Description	Status
InboundConnectionDown	Inbound adapter service's connection to the DS is down. This is an LdapConnectionException	Remote Fault with appropriate error message. The LDAP Adapter should retry the connection to the DS.
MalformedDN	Malformed baseDN in event scope. This is an LdapValidationException	Binding fault.
MalformedFilter	Search filter is malformed according to string representation of ldap filters. See http://www.ietf.org/rfc/rfc2254.txt LdapValidationException	Binding fault.
InvalidAdapterConfiguration	Adapter inbound service configuration is invalid due to one or more reasons. LdapValidationException	Binding fault with the appropriate message indicating the violation.
EndpointActivationError	Validation fails while activating the endpoint due to one or more activation spec errors. LdapValidationException	Binding fault.
EventProcessingError	Occurs if there was an error while processing an event received from the directory server. LdapRetriableResourceException	The fault is retried according to configured retry parameters and rejected once the appropriate retries are done.

Logging

The following logs are available for the LDAP Adapter. You can view these Runtime Loggers from the WebLogic Console. The logging level for each is inherited from the parent.

- `oracle.soa.adapter.ldap.inbound`
- `oracle.soa.adapter.ldap.outbound`
- `oracle.soa.adapter.ldap.connection`

- `oracle.soa.adapter.ldap.transaction`

Security

The LDAP Adapter must sign-on with valid security credentials. To enable this, the Oracle WebLogic Server supports both container-managed and application-managed sign-on.

Container managed sign-on enables a user to sign-on to Oracle WebLogic server and to access the Enterprise Information System through the resource adapter without having to sign-in separately to the Enterprise Information System.

Because the Oracle WebLogic server and the LDAP Enterprise Information System maintain independent security realms, this is achieved by using credentials mapping. Oracle WebLogic Server security principals are mapped to the corresponding credentials required to access the Enterprise Information System. The Adapter deployment descriptor does not need to store sensitive credential information.

Creating Outbound Credential Mappings

Outbound credential mappings enable you to map WebLogic Server user names to user names in the Enterprise Information System (EIS) to which you want to connect using a resource adapter. You can use default outbound credential mappings for all outbound connection pools in the resource adapter or you can specify particular outbound credential mappings for individual connection pools.

As an alternative to providing the bindDN and password in the ConnectionFactory properties, you can do the following to create credential mappings:

1. Open the **Connection Pool** in WebLogic console.
2. Navigate to**Deployments**, **LdapAdapter**, **Configuration**, **Outbound Connection Pools**, expand **javax.resource.cci.ConnectionFactory**, and select the pool that you created.
3. Click on the**Authentication** tab and ensure that **Resource Authentication Source** is set to **Container**.

   If it is not **Container** and you set it now, click **Save**.
4. Navigate to**Deployments**, **LdapAdapter**, **Security**, and **Outbound Credential Mappings** .
5. Click **New**, and check the **ConnectionFactory JNDI** for which you want to configure the credentials.
6. Select Default User and select **Next**.
7. Enter the **Enterprise Information System User name** as the `bindDN` information, for example, `cn=Directory Manager` and enter the password for the `bindDN`. for example, `welcome`.
8. Click **Finish**.
9. Update the **LdapAdapter Deployment**.

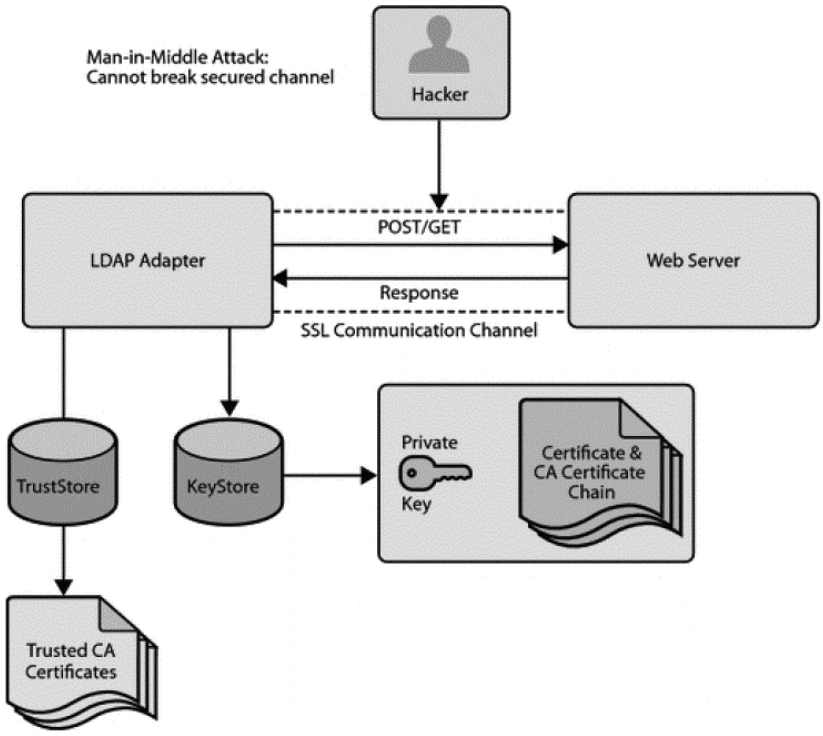
The Credential Mappings you supply take precedence over the bindDN and password properties you have configured in the ConnectionFactory, if you configured any. Use this connection factory as you would use any LDAP connection factory.

After you completed the last step, you have configured the Enterprise Information System User Name and Password that you wanted to use to map the WebLogic Server User.

LDAP over SSL

Secured Sockets Layer, or SSL, provides a secure and encrypted channel for communication between the client and the server. See Figure 12-27 for an overall view of SSL in the LDAP over SSL environment.

Figure 12-27 LDAP Over SSL



In environments in which sensitive data is transferred to remote servers (for example, sending credit card information to HTTP servers), the issue of security is very important. Security in these cases primarily refers to two requirements:

- Trust in the remote server with which you are exchanging data.
- Protection from third parties trying to intercept the data

The LDAP Adapter's use of SSL certificates and encryption satisfy these two security requirements.

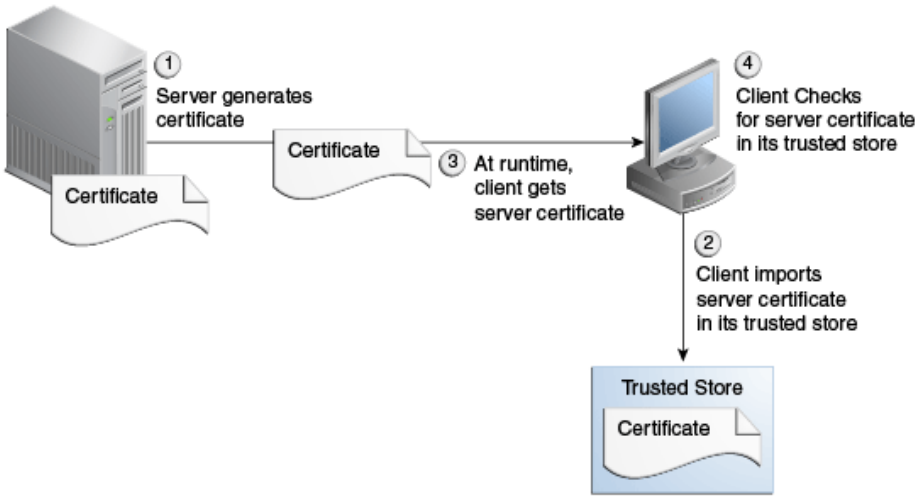
To gain the trust of clients in SSL environments, servers obtain certificates (typically, X.509 certificates) from recognized certificate authorities. Every client trusts a few parties. If the server is one of these trusted parties, or if the server's certificate was issued by one of these trusted parties, there is trust, even indirectly. This is also known as server authentication. The LDAP Adapter provides this type of server-authentication SSL handshake.

Since there is no notion of a rejected message in the LDAP Adapter, the LDAP Adapter has to raise a non-retriable exception.

Note: LDAP Adapter only supports **server authentication** or **server-side authentication**. It doesn't support **dual authentication**, where LDAP server authenticates client certificate with its truststore.

See Figure 12-28 for a representation of the use of certificates and a trusted store.

Figure 12-28 Using Certificates and Trusted Store with SSL



Payload Size Threshold

You can set the Payload size threshold in the `composite.xml` as a binding property on the adapter service.

By default, there is no limit on the payload size threshold. However, the LDAP adapter enforces the rule that any message published to Fusion Middleware must be less than the payload size threshold limit. Violating messages are rejected. Because there is no notion of a rejected message in the LDAP Adapter, the Adapter raises a non-retriable exception instead. Results are not going to alter in a re-try scenario so there is no point in retrying the `payloadSizeThreshold`.

High Availability

The LDAP adapter is highly available and propagates a change once and only once without message loss. The LDAP Adapter is deployed as a singleton in an active cluster. This implies that at any given time, only a single instance of the LDAP Adapter receives notifications from directory servers. The High Availability Support is applicable for inbound operations only. The LDAP Adapter does not provide any transaction support for outbound operations.

LDAP Adapter Exception Handling

LDAP Adapter Exceptions consist of the following types of exceptions:

- Inbound Retriable Exceptions
- Non-Retriable Inbound Exceptions
- Outbound Retriable Exceptions
- Outbound Non-retriable Exceptions

Inbound Retriable Exceptions

Inbound retryable exceptions are usually connection-related. The LDAP Adapter retries until a connection or the retriable exception condition is corrected.

The following binding properties apply to such exceptions. If retry properties are not specified, retry occurs indefinitely.

Table 12-4 Inbound Retryable JCA LDAP Adapter Properties

Property Name	Allowed	Description
Jca.retry.count	-	Indicates maximum number of retries before rejection.
Jca.retry.interval	Measured in Seconds	Indicates time interval between retries.
Jca.retry.backoff	Positive Integer	Indicates the retry interval growth factor.
Jca.retry.maxInterval	-	Indicates maximum value of retry interval
Jca.retry.maxPeriod	-	Is the upper limit of the entire accumulated duration of all retries; that is, a hard limit.

Inbound Non-Retriable Exceptions

Rejection Handlers define the course of action in this case.

Outbound Retriable Exceptions

The LDAP Adapter performs retries according to configured binding properties.If these binding properties are not specified, the retry is carried out by fault policies, if they are included as part of the composite.

The following properties govern outbound retriable exceptions.

Table 12-5 Outbound Retriable LDAP Adapter JCA Properties

Property Name	Allowed	Description
`Jca.retry.count`	-	Indicates the maximum number of retries before rejection.
`Jca.retry.interval`	Measured in Seconds	Indicates the time interval between retries
`Jca.retry.backoff`	Positive Integer	Indicates the retry interval growth factor
`Jca.retry.maxInterval`	-	Indicates maximum value of retry interval
Jca.retry.maxPeriod	-	Is the upper limit of the entire accumulated duration of all retries; that is, a hard limit.

Table 12-5 (Cont.) Outbound Retriable LDAP Adapter JCA Properties

Property Name	Allowed	Description
Jca.retry.count	-	Indicates maximum number of retries before rejection.
Jca.retry.interval	Measured in Seconds	Indicates time interval between retries

Outbound Non-Retriable Exceptions

Fault policies are executed if a non-retriable outbound fault occurs.

LDAP Adapter Samples

For LDAP Adapter Samples, see the SOA Samples site. The table below indicates the samples that are available for the LDAP Adapter.

Table 12-6 LDAP Adapter Samples

Sample	Description
Ldap_Search	Search a directory server for entries with results sorted by a given sort key.
Ldap_Modify	Modify, Delete or Rename entries using LDAP adapter
Ldap_Incremental_Sync	Capture events on a source directory server and replicate those events on a target directory server
Ldap_DSML_Service	DSML gateway service using LDAP Adapter.

13

Oracle JCA Adapter for Microsoft Message Queueing

The chapter describes Oracle JCA Adapter for Microsoft MQ concepts, features, configuration and use cases.

This chapter describes how to use the Oracle JCA Adapter for Microsoft Message Queueing, which provides access to MSMQ functionality and works with the Oracle BPEL Process Manager (Oracle BPEL PM) and Oracle Service Bus (OSB). The Adapter also supports processing of various message formats from MSMQ through the Native Format Translation framework (nXSD).

This chapter consists of the following sections:

Oracle JCA Adapter for MSMQ Concepts and Features

Microsoft Message Queueing (MSMQ) is a message infrastructure and a development platform for creating distributed, loosely-coupled messaging applications for the Microsoft Windows operating system.

Message queuing applications use message queuing to communicate across heterogeneous networks with computers that might be offline. Microsoft message queuing provides guaranteed message delivery, routing, security, transaction support and priority based routing.

Message Queues are logical containers that MSMQ uses to store and later forward those messages, thus providing the basis for loosely-coupled aspects of Message Queuing. Queuing applications send messages to the queue without needing to know when the messages are processed and which receiving application actually processes the message.

Applications that use MSMQ create/locate a queue, connect to the queue, navigate the queue, send/receive messages from a queue and use the MSMQ queue properties to define the behavior of the queue where applicable and needed.

This section has the following subsections:

- MSMQ Properties Supported

MSMQ Terminology

In addition to having familiarity with basic MSMQ concepts, it is important to understand basic MSMQ terminology as background to using the MSMQ Adapter. While you should consult the relevant Microsoft documentation for a thorough understanding of the MSMQ technology, the following definitions help you understand the MSMQ product at a level that complement your use of the MSMQ Adapter.

- **Public Queues** – A queue registered in the directory service that can be located by any Message Queuing application. This enables the MSMQ application to locate and open a queue anywhere within its domain. Public queues enable multi-hop scenarios, where messages are replicated throughout the Active Directory Service.
- **Private Queues**–A queue registered on the local computer (and not in the directory service) that typically cannot be located by other applications.
- **MSMQ queue** – A temporary storage location from where messages can be sent and received reliably, as and when conditions permit.
- **MSMQ user message queues** – Queues that are either private or public.
- **Remote Queue** – A queue manager is a Message Queuing service that delivers, receives, authenticates, and routes messages, and maintains information in the directory service. For an application, a remote queue is a queue that is hosted by a queue manager other than the one with which the application communicates.
- **Distribution Lists** – Distribution lists are public lists of destinations that are stored in Active Directory Domain Services (ADDS).
- **ADDS** – Active Directory Domain Services is a directory service implemented by Microsoft for Windows domain networks. It is included in most Windows Server operating systems.
- **Transactional Queues** – A queue that contains transactional messages. Transactional queues can only contain transactional messages, which are messages sent within a transaction. You can use transactional messages to pair the sending or receiving of any message with an action in another operation. Using transactional messages ensures that the unit of work is carried out as an atomic operation.
- **NonTransactional Queues** – A queue that contains only non-transactional messages. Message Queuing does not allow transactional messages in non-transactional queues.
- **Foreign Queue** – A queue that resides on a computer that does not run Message Queuing (a foreign computer).

jCOM and the MSMQ Adapter

The MSMQ Adapter uses Oracle WebLogic jCOM to enable interaction with MSMQ v5.0. jCOM assists in providing Java-to-COM integration.

Background

Specifically, WebLogic jCOM provides a runtime component that implements both COM/DCOM over Distributed Computing Environment Remote Procedure Call, and

Remote Method Invocation (RMI) over the Java Remote Method protocol/Internet Inter-ORB Protocol distributed components infrastructures. This makes the objects on the other side of an interaction appear as if they were native objects for each environment.

The DCOM (Distributed Component Object Model) mode uses the Component Object Model (COM) to support communication among objects on different computers.

In a WebLogic jCOM application running in DCOM mode, the COM client communicates with WebLogic Server DCOM protocol.

In native mode, COM clients make native calls to WebLogic Servers (COM-to-WLS) and WebLogic Servers make native calls to COM applications.

For both COM-to-WLS and WLS-to-COM applications, because native mode uses native code dynamically loaded libraries (DLLs)—compiled and optimized specifically for the local operating system and CPU—using Native mode results in better performance.

Implications for the MSMQ Adapter

The implications for use with the MSMQ Adapter are that you can use Native Mode when the MSMQ server and the SOA server are installed on the same machine. The SOA server installed must be a Windows Platform and not another platform, such as a Linux Platform.If the MSMQ Adapter and MSMQ are installed on the same system, you can enable Native Mode at both the jCOM protocol level and at the MSMQ Adapter level. When you use Native Mode, the MSMQ Adapter can directly interact with the MSMQ server because they are on the same system, rather than your having to use DCOM protocols to communicate (which are used when the MSMQ adapter and the MSMQ server are on two different machines). Ensuring that MSMQ Adapter does not use the DCOM protocol to interact with MSMQ COM components provides you with performance benefits.

Security

When the MSMQ Adapter needs to make an outbound connection to the MSMQ server, it must sign on with valid security credentials. In accordance with the J2CA 1.5 Specification, the WebLogic Server supports both container-managed and application-managed sign-on for outbound connections. The MSMQ Adapter can leverage either of these methods to sign on to the Enterprise Information System.

Component-managed Sign-On

With component-managed sign-on, the component itself supplies the necessary security credentials when making the call to obtain a connection to an Enterprise Information System.The application server invokes the `createManagedConnection` method of `ManagedConnectionFactory` by passing a null Subject instance.

Container-Managed Sign-On

Container managed sign-on enables a user to sign-on to Oracle WeblogicServer and also to access the Enterprise Information System through the resource adapter without having to sign-in separately to the Enterprise Information System.

Because the Oracle WebLogic server and MSMQ maintain independent security realms, this is achieved by using credentials mapping. Oracle WebLogic Server security principals are mapped to the corresponding credentials required to access the Enterprise Information System.

Logging and Diagnosability

The MSMQ Adapter employs the MSMQ Adapter logging framework provided by the Adapter Framework component to capture any runtime logs.

The MSMQ adapter produces the following logs:

- `oracle.soa.adapter.msmq`
- `oracle.soa.adapter.msmq.transaction`
- `oracle.soa.adapter.msmq.connection`
- `oracle.soa.adapter.msmq.inbound`
- `oracle.soa.adapter.msmq.outbound`

Each logger can be set to `TRACE:32` to enable debug logging for that area or the `oracle.soa.adapter.msmq` logger can be set to `TRACE:32` to enable complete logging for the MSMQ Adapter. Any exception emanating from the MSMQ Adapter has a corresponding error code and error message

MSMQ Adapter and High Availability

The MSMQ Adapter is deployable in an active-active topology. As part of its implementation, the MSMQ Adapter enables each poller thread to poll the queue for the next available message. Each poller thread uses the receive API; on a successful read the message is removed from the queue. This ensures there is no message duplication when the MSMQ Adapter is deployed in an active/active topology.

Set Up MSMQ on Windows Server 2008

To use MSMQ on a Windows Server 2008 installation, enable the following features for a Windows Server on which MSMQ is to be installed. Consult the relevant Microsoft documentation for more setup and configuration information.

- Message Queuing Server. For more information on installation, see the Microsoft document Installing Message Queuing (MSMQ) at http://msdn.microsoft.com/en-us/library/aa967729.aspx
- Directory Service Integration (for Public Queues and Distribution Lists). The prerequisite specifically requires Active Directory Domain Services (AD_DS) configured on a Windows 2008 Server system. Active Directory Domain Services is required to access MSMQ public queues, otherwise only private queues are available for MSMQ Adapter use cases.
- Message Queuing DCOM Proxy

After successful installation, Message Queuing appears under the Features link in the Microsoft Server Manager console window.

Setup Oracle Weblogic Server for COM

To set up the Oracle WebLogic server for use with COM:

- Enable jCOM for the server that deploys the SOA MSMQ Adapter. For complete information on enabling jCOM, see the online version of the help at Oracle Fusion Middleware Oracle WebLogic Server Administration Console Online Help 12c Release 1 (12.1.3).

Note that if jCOM is enabled on the SOA server, while activating the MSMQ Adapter, the target on which it should be deployed must be the SOA server itself. The jCOM must be enabled on the managed server to which the adapter is targeted.

- When Weblogic is installed on a Windows machine which is running Microsoft Message Queuing, you can configure native mode though the configuration option 'Enable Native Mode' as outlined in this documentation: http://docs.oracle.com/cd/E24329_01/apirefs.1211/e24401/pagehelp/Corecoreserverserverprotocolsjcomtitle.html

  Enable Native Mode for jCOM. For more information on enabling Native Mode for jCOM, see http://docs.oracle.com/cd/E15051_01/wls/docs103/jcom/comtowls.html#wp1074435

- Finally, enabling jCOM requires a restart of the corresponding server that deploys the SOA MSMQ Adapter; restart the WebLogic Server.

Transaction Management and Error Handling

For more information on Transaction Management and Error handling as it applies to Adapters in general, see Adapter Framework.

Transaction Management

MSMQ Adapter transaction support defaults to LocalTransaction. The MSMQ Adapter does not support XA Transactions. Transactional queues are supported when used in local transaction semantics.

For both inbound and outbound error cases, the MSMQ Adapter starts an internal MSMQ transaction if the `TransactionMode` property of the JCA-managed connection factory is set to `Single`.

If the `TransactionMode` property of the JCA-managed connection factory is set to `None`, the MSMQ Adapter does not start a transaction on the MSMQ side; the messages are produced and consumed in a non-transactional way.

The `TransactionMode` property value of None is required when receiving and sending messages to or from a Non-transactional queue.

Fault Handling

The MSMQ Adapter can handle faults encountered when producing or consuming a message to or from an MSMQ queue

If a fault occurs, the message is delivered to a BPEL recovery queue and/or is retried based on the nature of the fault, and depending if the error is retriable or non-retriable. For more information on handling Adapter Faults, see Error Handling.

Because the MSMQ Adapter does not support XA transactions, XA-retriable errors are not supported.

XA retriable errors refer to the errors that occur in context of an XA transaction. The adapters can throw PCRetriable or XARetriable exceptions. However, because there is no XA when used with MSMQ Adapter, the MSMQ Adapter does not allow for XA errors.

Outbound Retriable Errors

The MSMQ Adapter performs retries according to configured binding properties. If these binding properties are not specified, the retry is carried out by fault policies, if they are included as part of the composite application.

For more information on MSMQ binding properties, see the Adapter Properties chapter, Table 13-3.

Outbound Non-Retriable Errors

Fault policies are executed if an outbound non-retriable fault occurs.

MSMQ Adapter Features

The Oracle MSMQ Adapter provides the following features:

- **Sending Messages to MSMQ Private Queues**
  - The MSMQ Adapter enables sending a message to a local private queue. When using the MSMQ Adapter Configuration Wizard to model an MSMQ Adapter reference, you can model an enqueue operation that is used to send (or Put) a message to an MSMQ queue.
  - The MSMQ Adapter enables sending a message to a local transactional private queue. When using the MSMQ Adapter Configuration Wizard to model an MSMQ Adapter reference, you can model an enqueue operation that is used to send a message to a local transactional MSMQ queue.

Note:

It is important to understand the relevance of the MSMQ `TransactionMode` property, which indicates if the connection participates in a transaction when sending and receiving a message. Values for this property are [Single | None].

If the value is Single, the local and remote MSMQ queues to which the message is sent should be transactional. If the value is Single, the local MSMQ queue from which the message is retrieved should be Transactional.

If the value is None and the queues to which the message is sent are transactional, a ResourceException occurs.

- **Sending messages to MSMQ Public Queues**
  - The MSMQ Adapter enables sending a message to a local public queue. When using the MSMQ Adapter Configuration Wizard to model an MSMQ Adapter reference, you can model an enqueue operation. You can use this enqueue operation to send a message to a public MSMQ queue.
  - The MSMQ Adapter enables you to send a message to a local transactional public queue. When using the MSMQ Adapter Configuration Wizard to model an MSMQ Adapter reference, you can model an enqueue operation. You can use this enqueue operation to send a message to a MSMQ queue.
- **Sending Messages to MSMQ Distribution Lists**
  - The MSMQ Adapter enables you to send a message to a Distribution List. When using the MSMQ Adapter Configuration Wizard to model an MSMQ Adapter

reference, you can configure an enqueue operation to send a message to a MSMQ distribution list.

- **Consuming or Receiving Messages from a Private MSMQ Queue**
  - The MSMQ Adapter enables you to consume or receive of a message from a local private MSMQ queue. The default behavior is for the next available message on the queue to be available for consumption.

    To do this, use the MSMQ Adapter Configuration wizard to model a Dequeue operation that enables consumption of a message from an MSMQ private queue.The MSMQ Adapter enables message consumption from a local private transactional MSMQ queue and supports an operation called dequeue.

    To do this, use the Configuration Wizard to create the dequeue that consumes or receives a message from a transactional MSMQ queue.
- **Receiving Messages from a Public MSMQ Queue**
  - The MSMQ Adapter enables reception of a message from a public MSMQ queue. The MSMQ Adapter supports an operation called dequeue that you configure when modeling an adapter service and is used to consume or receive a message from a MSMQ queue.
  - The MSMQ Adapter enables reception of a message from a public transactional MSMQ queue. You can use the MSMQ Adapter Configuration Wizard operation called dequeue that is used to receive a message from a transactional MSMQ queue.

MSMQ Properties Supported

The MSMQ Adapter includes several JCA properties. They are provided here and in the properties appendix for your convenience.

See Table 13-1 for a list of JCA properties.

Table 13-1 MSMQ Adapter JCA Properties

Property	Description	Default Value	Required

`DestinationType`	Indicates if the message is sent to a public queue, private queue or a group of queues as identified by the distribution list name. The values are;
	`PUBLIC_QUEUE`
	`PRIVATE_QUEUE`
	`DISTRIBUTION_LIST`
		None	Yes

`DestinationName`	Name of the MSMQ queue.	None	Yes, if UseActiveDirectory Path is False

Table 13-1 (Cont.) MSMQ Adapter JCA Properties

Property	Description	Default Value	Required
`DestinationPath`	The string that identifies a DistributionList or Public queue as represented in ActiveDirectory. An example string is listed below; LDAP://MyLDAPServer/CN=MyQueue,CN=msmq,CN=My Computer,CN=Computers,DC=My Domain,DC=MyCompany,DC=COM	None	Yes, if UseActiveDirectory Path is True
`UseActiveDirectoryPath`	Boolean that allows for Active Directory Path to be used to identify a public queue instead of queue name. This property is applicable when DestinationType is '`DISTRIBUTION_LIST`' or '`PUBLIC_QUEUE`'	False	-
`UseDirectFormatName`	Boolean that allows for Direct Format name to be used for public and private queues.	False	-
`Priority`	Priority can be set to any integer value between 7 and 0 (the default is 3). 7 means higher priority. 0 means lowest priority. Highest priority implies faster processing	3	-
`TimeToLive`	This property specifies a time limit (in seconds) for the message to be retrieved from the target queue. The value is assigned to `MaxTimeToReceive` property for a given MSMQ message.	-1 (infinite	-
`Delivery`	The property is used to specify express (non-persistent) or recoverable (persistent) messaging. Express messaging provides faster throughput. Recoverable messaging guarantees that the message is delivered even if a computer crashes while the message is en route to the queue. The values are: Persistent, Non-Persistent.	Persistent	-
`OperationType`	The operation to be carried out. The supported values for OperationType are `enqueue` and `dequeue`.	None	Yes
`BodyType`	Values are `String` (default) and `ByteArray`. When opaque (schema) processing option is selected that would imply BodyType value of ByteArray.	String	Yes

Table 13-1 (Cont.) MSMQ Adapter JCA Properties

Property	Description	Default Value	Required
`EnableStreaming`	Boolean that enables payload to be streamed	False	-

See Table 13-2 for a list of Normalized Properties related to the MSMQ Adapter. Also indicated is the Adapter processing direction for each property.

Table 13-2 MSMQ Adapter Normalized Properties

Property Name	Description	Direction
`jca.msmq.message.SentTime`	The property indicates when a message is sent.	Inbound
`jca.msmq.message.Priority`	The property specifies a message's priority. This overrides `Priority` in MSMQInteractionSpec.	Inbound/Outbound
`jca.msmq.message.TimeToLive`	The property specifies a time limit (in seconds) for the message to be retrieved from the target queue. This overrides `TimeToLive` in the MSMQInteractionSpec.	Inbound/Outbound
`jca.msmq.message.MaxTimeToReachQueue`	The property specifies a time limit (in seconds) for the message to reach the queue. Message Queuing uses the enterprise-wide setting for the time-to-reach-queue interval if none is specified here.	Inbound/Outbound
`jca.msmq.message.Id`	Identifies the message using an MSMQ-generated message identifier.	Inbound
`jca.msmq.message.Delivery`	The property specifies how Message Queuing delivers the message. This overrides `Delivery` in MSMQInteractionSpec	Inbound/outbound
`jca.msmq.message.BodyLength`	The property provides the length of the message body in bytes.	Inbound

Table 13-2 (Cont.) MSMQ Adapter Normalized Properties

Property Name	Description	Direction
`jca.msmq.message.ArrivedTime`	The property indicates when the message arrived at the queue.	Inbound

See Table 13-3 for a list of Binding Properties that apply to the MSMQ Adapter.

Table 13-3 MSMQ Adapter Binding Properties

Property Name	Description
adapter.msmq.receive.timeout	The time (in milliseconds) that Message Queuing waits for a message to arrive before starting another poll-cycle. The default value is one second (1).
adapter.msmq.dequeue.threads	Number of poller threads that is initialized when endpoint activation occurs. This enables the adapter to wait for a specified time to receive a message before next poll cycle is initiated. When specified, the value of adapter.msmq.dequeue.threads is used to spawn multiple inbound poller threads; multiple inbound threads can be used to improve performance The default value is 1.

Use the Connection Factory Configuration properties when you configure Connection Factories for the MSMQ Adapter. See Table 13-4. Also indicated is the processing direction for each property.

Table 13-4 MSMQ Adapter Connection Factory Configuration

Property Name	Description	Default
`Host`	IP Address of the MSMQ host.	-
AccessMode	Identifies if the connection factory allows for native access or not. Values are `[Native | DCOM]`. If Native, the Oracle WebLogic Server should be installed on the same host as MSMQ.	DCOM
`TransactionMode`	Indicates if the connection participates in a transaction when sending and receiving a message. Values are [Single| None].If Single, the local and remote MSMQ queues to which the message is sent should be transactional. If Single, the local MSMQ queue from which the message is retrieved should be Transactional.If None and the queues to which the message is sent are transactional, there is a ResourceException.	-
User	Identifies a user.	-

Table 13-4 (Cont.) MSMQ Adapter Connection Factory Configuration

Property Name	Description	Default
Password	Password for the specified user.	-
`Domain`	Domain of the MSMQ host.	-

MSMQ Adapter Configuration Wizard Flow

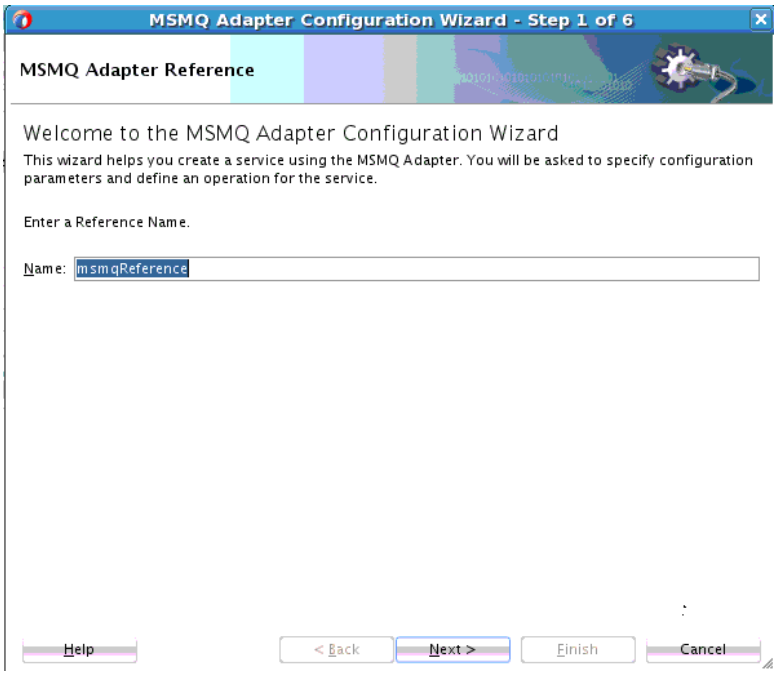
Use the MSMQ Adapter Configuration Wizard to create and configure an MSMQ Adapter.

Creating an Enqueue Operation

In this example walkthrough, you create an Enqueue Operation.

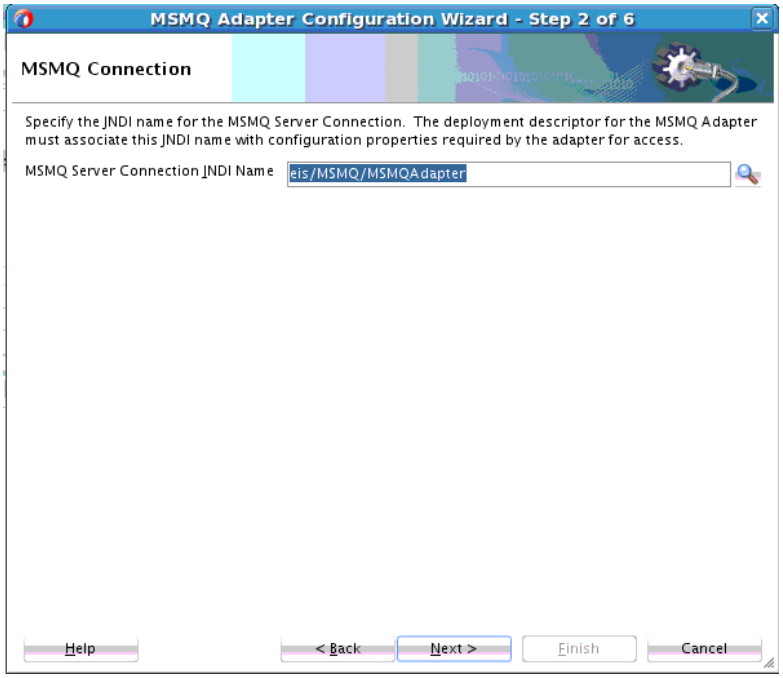
1. When you drag and drop MSMQ Adapter from the Component Palette of JDeveloper BPEL Designer, the Adapter Configuration Wizard starts with a Adapter Configuration Wizard Welcome page. Click **Next**.
2. The MSMQ Adapter Configuration Wizard prompts you to enter a service or reference name, as shown in Figure 13-1

Figure 13-1 MSMQ Configuration Wizard Service or Reference Name Screen



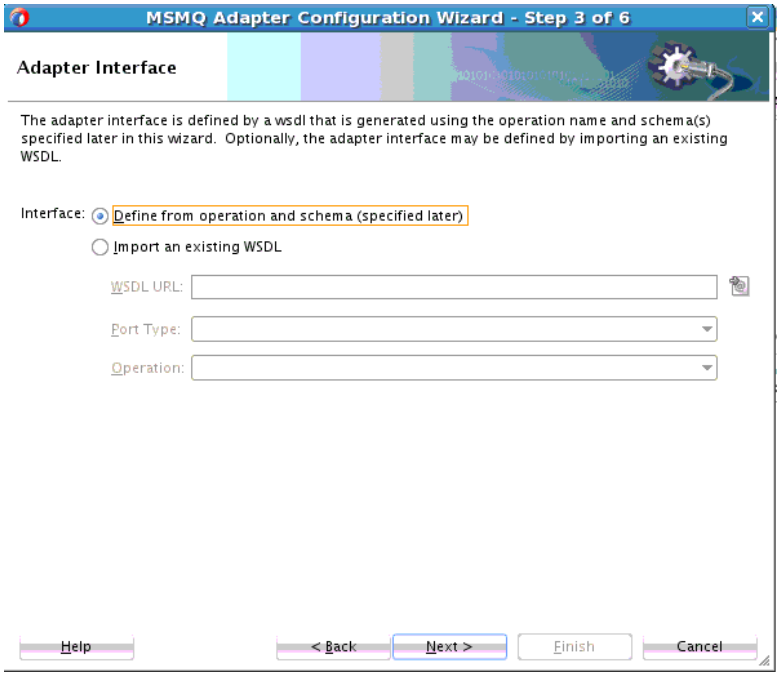
3. Next, specify a the JNDI name for the MSMQ Service connection, as shown in Figure 13-2.

Figure 13-2 MSMQ Adapter Configuration Wizard MSMQ Connection Screen



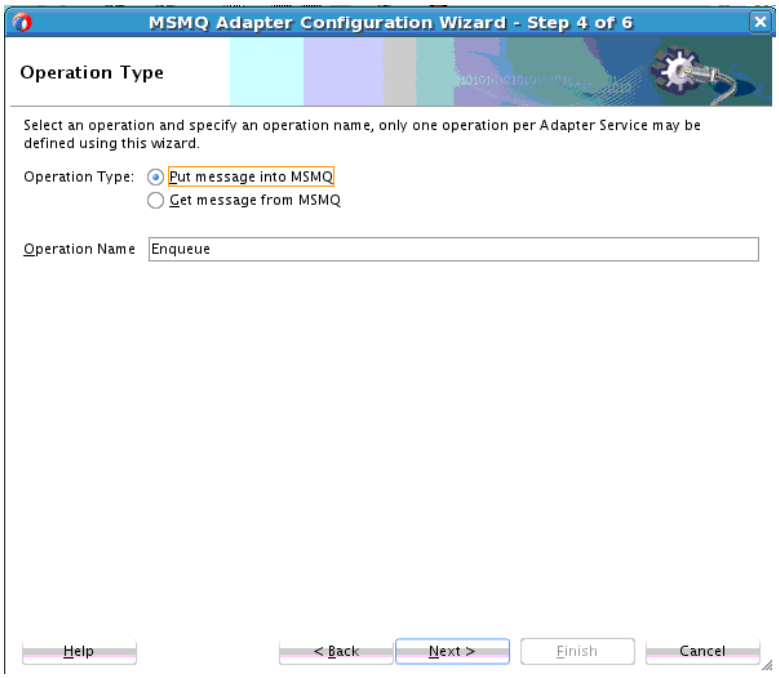
4. On the MSMQ Configuration Wizard **Adapter Interface** page, you can either create a new MSMQ adapter WSDL file using the selected operation, or import an existing WSDL that already specifies the operation.

Figure 13-3 MSMQ Adapter Configuration Wizard Adapter Interface Screen



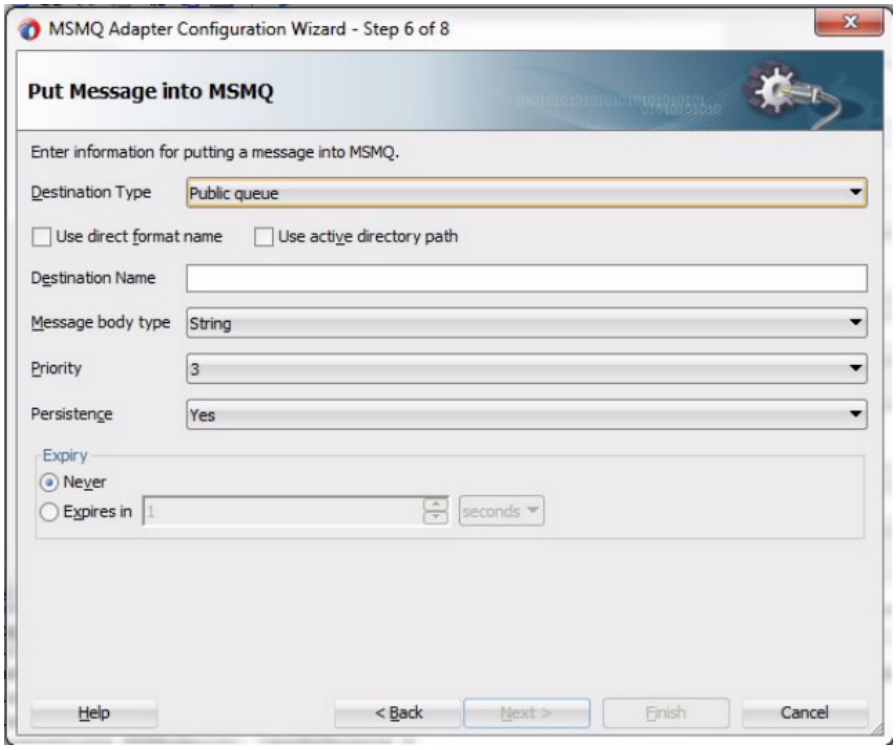
5. On the MSMQ Adapter Configuration Wizard **Operation type** page, you can select a valid operation for the MSMQ adapter configuration. If you are updating the operation, the operation is pre-selected on this page and the operation name is pre-populated. If you have imported an existing WSDL, the operation name on this page is pre-populated.

Figure 13-4 MSMQ Adapter Operation Type Screen



6. The next step is to create an Enqueue Operation. This page captures the configuration parameters for the Enqueue operation.

Figure 13-5 Enqueue Operation



Enter the following:

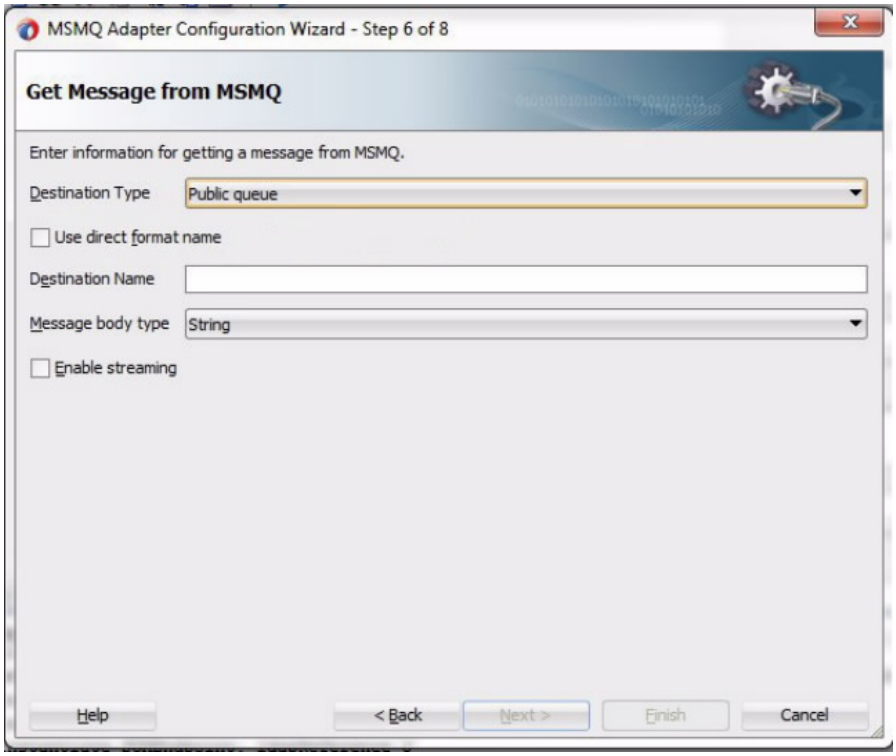
- In the **Destination Type** field, specify `Public Queue`, `Private Queue` or `Distribution List`.

- If `Public queue` or `Distribution List` is selected, the **Use active directory path** checkbox is enabled, and **Private queue** is disabled.
- If **Use active directory path** checkbox is selected, you can specify the **Destination Path** of the queue, otherwise you can specify the **Destination Name** of the queue.
- Values for **Message Body Type** are `String` (default) and `ByteArray`.
- Values for **Priority** are from 0 (lowest priority) to 7 (highest priority). The default priority is 3.
- Values for **Persistence**: are `Yes` (default), or `No`.
- **Expiry** units are seconds, minutes, hours and day. The MSMQ Adapter Configuration Wizard converts Units other than seconds into seconds, while setting the value for the `TimeToLive` interaction spec property value.

7. If you had chosen the `Dequeue`, or `Get Message`, Operation, the **Dequeue message** appears. On this screen, you can enter the parameters to dequeue message.

   There is a checkbox for direct format name. Direct format names are used to reference public or private queues without accessing the directory service. Direct format names are used when sending messages directly to a computer, to computers over the Internet, sending messages to any queue while operating in domain, workgroup, or offline mode, reading messages while operating in domain, workgroup, or offline mode, or sending messages across forest boundaries.

Figure 13-6 Dequeue Operation (Get Message from MSMQ) Screen



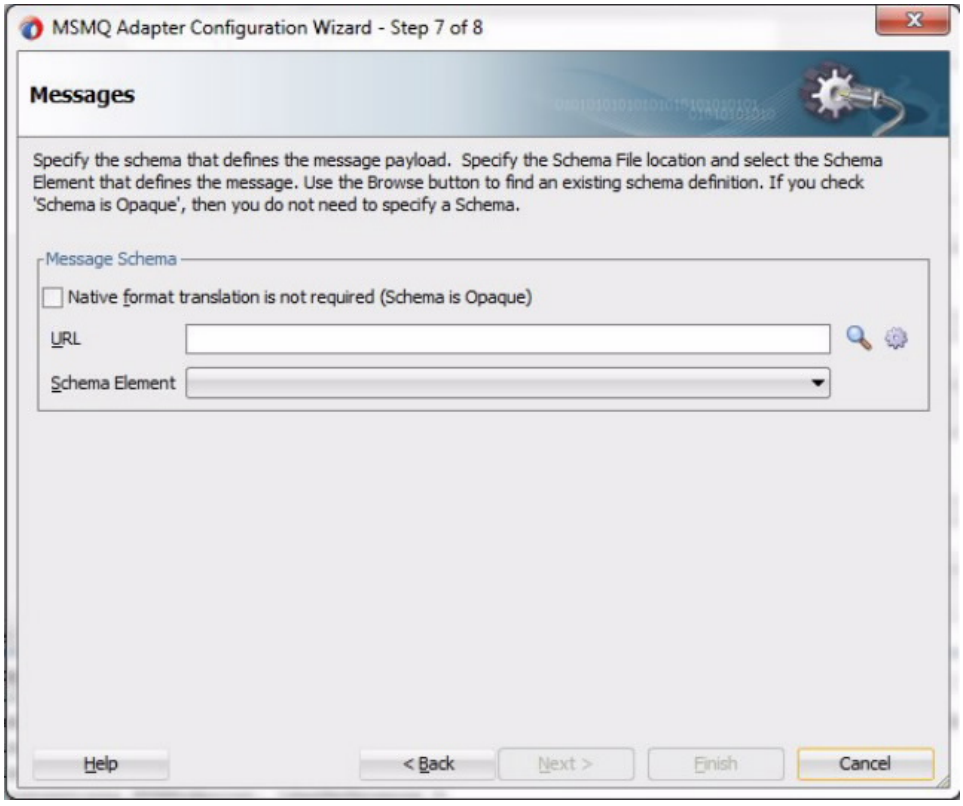
On this screen, you can specify the following:

- Specify the **Destination Type**, `Public Queue` or `Private Queue`.

- Specify the **Destination Name** of the queue.
- Choose either `String` (the default) or `ByteArray` as the **Message Body Type**.
- Enable streaming by checking the **Enable streaming** checkbox. When you enable this feature, the payload is streamed to a database rather than being manipulated in SOA runtime as in a memory DOM. You use this feature while handling large payloads. When you select the **Enable streaming** check box, a corresponding Boolean property `StreamPayload` is appended to the Activation Spec properties defined in the respective `.jca` file

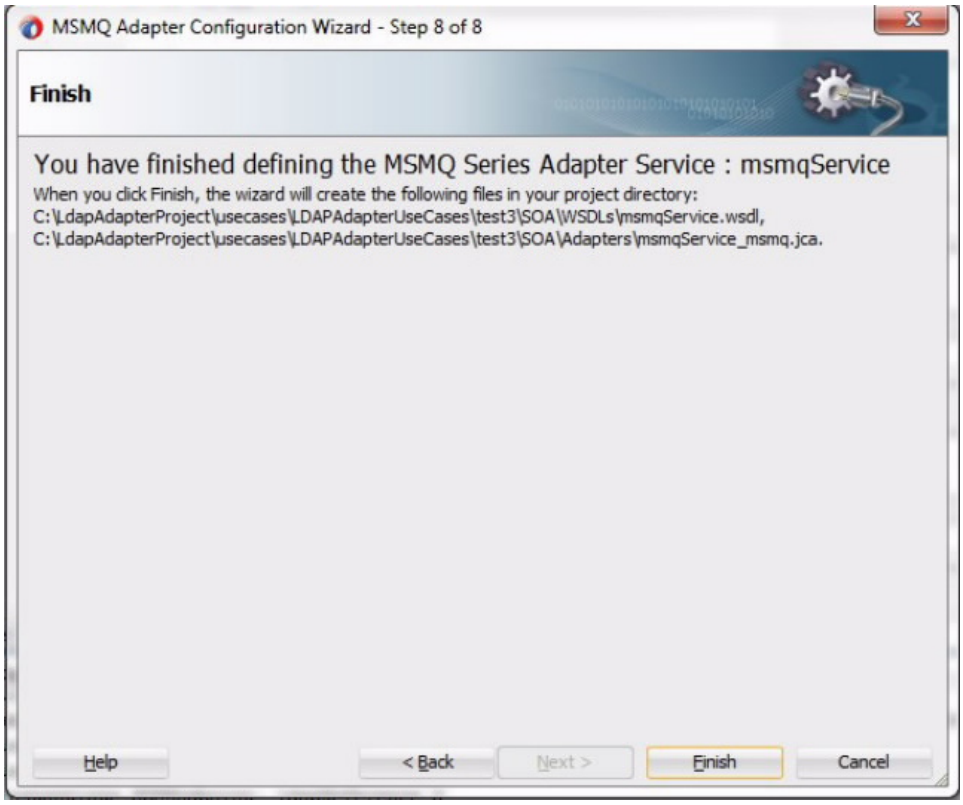
8. The Adapter Configuration Wizard displays the **Schema Page**. If you need the message payload translated using a certain schema, enter the schema details, which define the incoming notification body content. Use the Native Format builder to define the Schema for native format. If the message payload is opaque, you can choose the `Schema is Opaque` option.

Figure 13-7 MSMQ Adapter Configuration Wizard Messages Screen



9. Click **Next**. The MSMQ Adapters **Finish Page** appears. Click **Finish** to create the indicated files in your project directory.

Figure 13-8 The MSMQ Adapter Configuration Wizard Finish Page



Sample MSMQ Adapter Connection Factory Properties

The following are sample MSMQ Adapter connection factory properties, within Fusion Middleware Control, to see them proceed to **Home >Summary of Deployments >MSMQAdapter > Configuration> Outbound Connection Pools**. Select the **New** option and provide the JNDI a name, for example, `eis/MSMQ/MSMQAdapter_NonTrans1`

Properties include:

```
MSMQ_AccessMode=DCOM
MSMQ_Domain=adapter.test.msmq
MSMQ_Host=slc04lya.us.mydomain.com
MSMQ_Password=Welcome12
MSMQ_TransactionMode= NONE
MSMQ_User=Administrator
```

MSMQ Adapter Design-time Artifacts

The Adapter configuration Wizard generates the JCA, WSDL and the XML Schema artifacts based on the interaction and message definition.

Sample JCA File for an MSMQ Enqueue Operation

Following is the sample JCA file, specifying a WSDL for an MSMQ Enqueue Operation.

Example - Sample JCA File for an MSMQ Enqueue Operation

```
<adapter-config name="enqueueOp" adapter="msmq" wsdlLocation="../WSDLs/
enqueueOp.wsdl"
```

```
                xmlns="http://platform.integration.oracle/
                      blocks/adapter/fw/metadata">
  <connection-factory location="eis/MSMQ/MSMQAdapter"/>
  <endpoint-interaction portType="EnqueueOperation_ptt" operation="EnqueueOperation">
    <interaction-spec
                xmlns="http://platform.integration.oracle/
                      blocks/adapter/fw/metadata">
  <connection-factory location="eis/MSMQ/MSMQAdapter"/>
  <endpoint-interaction portType="EnqueueOperation_ptt" operation="EnqueueOperation">
    <interaction-spec
                className="oracle.tip.adapter.msmq.v2.jca.
                           MSMQInteractionSpec">
      <property name="BodyType" value="ByteArray"/>
      <property name="DestinationPath" value="dest"/>
      <property name="TimeToLive" value="-1"/>
      <property name="Delivery" value="Persistent"/>
      <property name="DestinationType" value="PUBLIC_QUEUE"/>
      <property name="UseDirectFormatName" value="true"/>
      <property name="OperationType" value="Enqueue"/>
      <property name="Priority" value="5"/>
      <property name="UseActiveDirectoryPath" value="true"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

Sample JCA for an MSMQ Dequeue Operation

The following sample JCA, specifying for a WSDL for a Dequeue Operation.

Example - Sample JCA for an MSMQ Dequeue Operation

```
<adapter-config name="dequeueOp" adapter="msmq"
   wsdlLocation="../WSDLs/dequeueOp.wsdl"
  xmlns="http://platform.integration.oracle/blocks/adapter/
    fw/metadata">
  <connection-factory location="eis/MSMQ/MSMQAdapter"/>
  <endpoint-activation portType="DequeueOperation_ptt"
    operation="DequeueOperation">
    <activation-spec
      className="oracle.tip.adapter.msmq.v2.
                 jca.MSMQActivationSpec">
      <property name="BodyType" value="String"/>
      <property name="DestinationType"
                 value="PUBLIC_QUEUE"/>
      <property name="UseDirectFormatName" value="false"/>
      <property name="DestinationName" value="dest"/>
      <property name="OperationType" value="Dequeue"/>
      <property name="EnableStreaming" value="false"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

Design-Time WSDL Artifacts

A WSDL file is generated when you click **Finish** in the MSMQ Adapter Configuration Wizard. The schema you specified within the Schema Page is imported in the generated WSDL.

WSDL for MSMQ Enqueue Operation

The following WSDL is generated for a Base64Binary Enqueue operation.

Example - MSMQ Adapter WSDL for Enqueue Operation

```
<wsdl:definitions
     name="msmqService"      targetNamespace="http://xmlns.oracle.com/
                           pcbpel/adapter/msmq/         MSMQAdapterUseCases/Project1/
```

```
msmqReference"
     xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/          msmq/
MSMQAdapterUseCases/Project1/msmqReference"
     xmlns:opaque="http://xmlns.oracle.com/
                   pcbpel/adapter/opaque/"
     xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/
                            partner-link/">
  <plt:partnerLinkType name="Enqueue_plt" >
    <plt:role name="Enqueue_role" >
      <plt:portType name="tns:Enqueue_ptt" />
    </plt:role>
  </plt:partnerLinkType>
    <wsdl:types>
      <schema targetNamespace="http://xmlns.oracle.com/
                            pcbpel/adapter/opaque/"
            xmlns="http://www.w3.org/2001/XMLSchema" >
        <element name="opaqueElement"
                        type="base64Binary" />
      </schema>
    </wsdl:types>
    <wsdl:message name="Enqueue_msg">
        <wsdl:part name="opaque" element=
                        "opaque:opaqueElement"/>
    </wsdl:message>
    <wsdl:portType name="Enqueue_ptt">
        <wsdl:operation name="Enqueue">
            <wsdl:input message="tns:Enqueue_msg"/>
        </wsdl:operation>
   </wsdl:portType>
</wsdl:definitions>
```

WSDL for MSMQ Adapter Dequeue Operation

The following example shows a WSDL for an MSMQ Adapter Dequeue Operation.

MSMQ Adapter Dequeue Operation WSDL

```
<wsdl:definitions
     name="mqService"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter
     /msmq/MSMQAdapterUseCases/Project1/msmqService"
     xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter           /msmq/
MSMQAdapterUseCases/
                   Project1/msmqService"
     xmlns:pc="http://xmlns.oracle.com/pcbpel/"
     xmlns:imp1="http://platform.integration.oracle/
                   blocks/adapter/fw/metadata/
                       msmqSchema"
     xmlns:plt="http://schemas.xmlsoap.org/ws/
                       2003/05/partner-link/">
       <plt:partnerLinkType name="Dequeue_plt" >
         <plt:role name="Dequeue_role" >
         <plt:portType name="tns:Dequeue_ptt" />
         </plt:role>
  </plt:partnerLinkType>
    <wsdl:types>
      <schema xmlns="http://www.w3.org/
```

```
                          2001/XMLSchema" >
        <import namespace="http://platform.integration.oracle/
                                        blocks/adapter            /fw/metadata/
msmqSchema"
                    schemaLocation="../Schemas/msmqSchema.xsd" />
      </schema>
    </wsdl:types>
    <wsdl:message name="Dequeue_msg">
        <wsdl:part name="body" element="imp1:message"/>
    </wsdl:message>
    <wsdl:portType name="Dequeue_ptt">
        <wsdl:operation name="Dequeue">
            <wsdl:input message="tns:Dequeue_msg"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

MSMQ Use Cases

The following use cases provide a description and walkthrough of various uses of the MSMQ Adapter. The use cases include:

- Enqueue/Dequeue Message from Public Queue
- Enqueue/Dequeue Message from Private Queue
- Enqueuing a Message to a Distribution List

Enqueue/Dequeue Message from Public Queue

This use case consists of the following steps

- Designing the SOA Composite
- Creating the Inbound Oracle MSMQ Adapter Service
- Creating the Outbound Oracle MSMQ Adapter Service
- Wiring Services and Activities
- Adding a Receive Activity
- Add an Invoke Activity
- Add an Assign Activity

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following steps:

1. In the Application Navigator of JDeveloper, click **New Application**. The select **SOA Application - Name your application** page is displayed.
2. Enter Enq_Deq_PublicApp in the **Application Name** field, and click **Next**. **Name your project page** is displayed.
3. Enter Enq_Deq_Public in the **Project Name** field, and click **Next**.

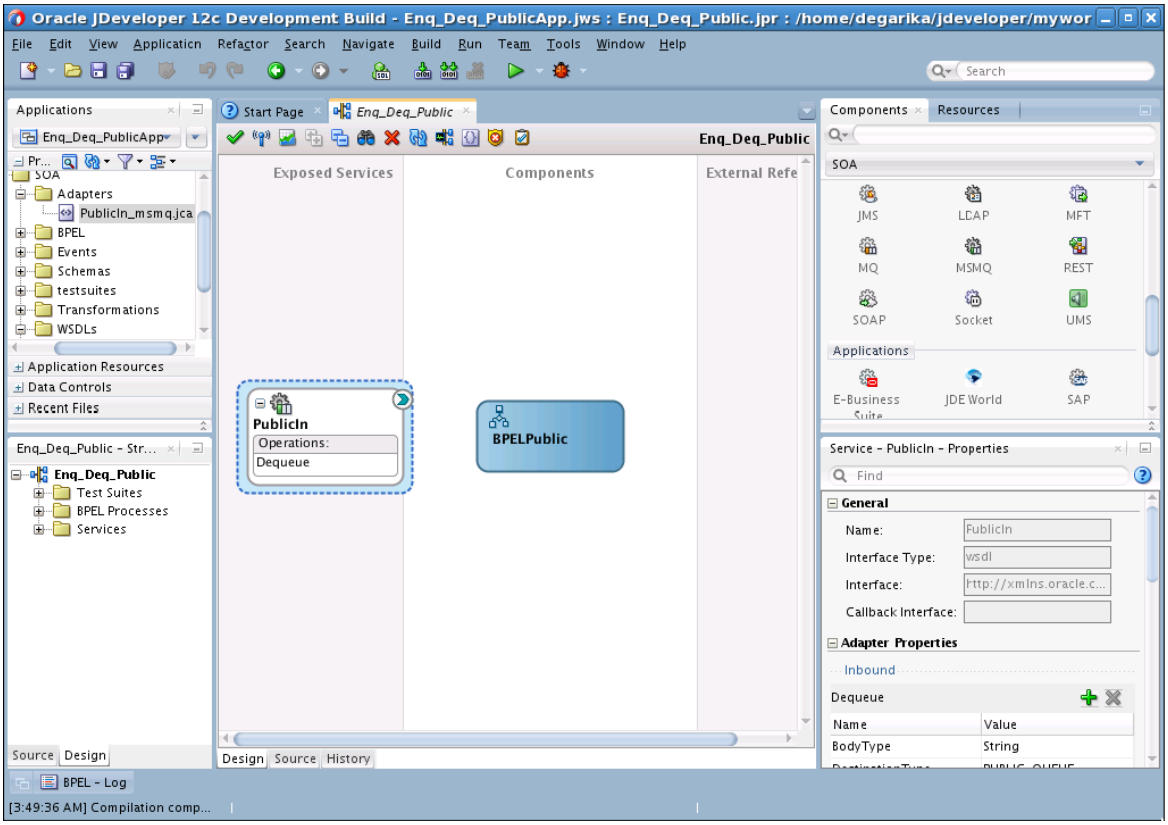
4. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The MSMQ Adapter Configuration Wizard displays the **Create BPEL Process - BPEL Process** page.
5. Enter `BPELPublic` in the **Name** field, select **Define Service Later** from the Template list.
6. Click **OK**. The `Enq_Deq_PublicApp` application and the `Enq_Deq_Public` project appear in the design area.

Creating the Inbound Oracle MSMQ Adapter Service

Perform the following steps to create an inbound Oracle MSMQ Adapter service to dequeue message from Microsoft messaging queue.

1. Drag and drop MSMQ Adapter from the Components to the Exposed Services swim lane. The **Adapter Configuration Wizard Welcome** page is displayed.
2. Enter `PublicIn` in the **Service Name** field.
3. Click **Next**. The **Adapter Connection** page is displayed.
4. Provide the JNDI connection name of the MSMQ Server. It can be used to either to connect for Transactional queues or for Non-Transactional queues. For example, in this use case, the JNDI used for the Non-Transactional Queue is `eis/MSMQ/MSMQAdapter_NonTrans`.
5. Click **Next**. The MSMQ Adapter Configuration Wizard displays the **Adapter Interface** page.
6. Click **Next**. The **Adapter Operation Type** page is displayed.
7. Select **Operation Type** as `Get message from MSMQ` and **Operation Name** as `Dequeue`.
8. Click **Next**. The MSMQ Adapter Configuration Wizard **Get Message from MSMQ** page is displayed.
9. Select **Destination Type as Public Queue**. Enter `Public_Queue_Deq` in the **Destination Name** field.
10. Click **Next**. The wizard displays the **Messages** page.
11. Select the **Native format translation is not required (Schema is Opaque)** checkbox. Note: If you have a schema, you can provide the schema for translation.
12. Click **Next**. The wizard displays the **Finish** page.
13. Click **Finish**. You have now configured the inbound Oracle MSMQ Adapter and the composite.xml appears, as shown in Figure 13-9.

Figure 13-9 The composite.xml for the Enqueue-Dequeue from Public Queue Use Case



Creating the Outbound Oracle MSMQ Adapter Service

Perform the following steps to create an outbound Oracle MSMQ Adapter service to enqueue the message from one Microsoft messaging queue to the other Microsoft messaging queue.

1. Drag and drop MSMQ Adapter from the Components to the External References swim lane. The Adapter Configuration Wizard **Welcome** page is displayed.
2. Enter `PublicOut` in the **Reference Name** field.
3. Click **Next**. The MSMQ Adapter Configuration Wizard indicates the **Connection**.
4. Provide the JNDI connection name of the MSMQ Server. It can be used to either connect for Transactional queues or Non-Transactional queues. For example, in this use case the JNDI is used for the Non-Transactional Queue is `eis/MSMQ/MSMQAdapter_NonTrans`.
5. Click **Next**. The **Adapter Interface** page is displayed.
6. Click **Next**. The **Adapter Operation Type** page is displayed.
7. Select `Put message into MSMQ` for **Operation Type** and `Enqueue` for **Operation Name**.
8. Click Next. The **Adapter Put Message into MSMQ** page is displayed.
9. Select **Destination Type as Public Queue**. Enter `Public_Queue_Enq` in the **Destination Name** field.

Note:

If you have the active directory path of the queue, you must select the **Use active directory** path checkbox and provide the appropriate value. Also, if you have the direct format name of the queue, you must provide the direct format name of the queue by selecting the checkbox **Use direct format name**.

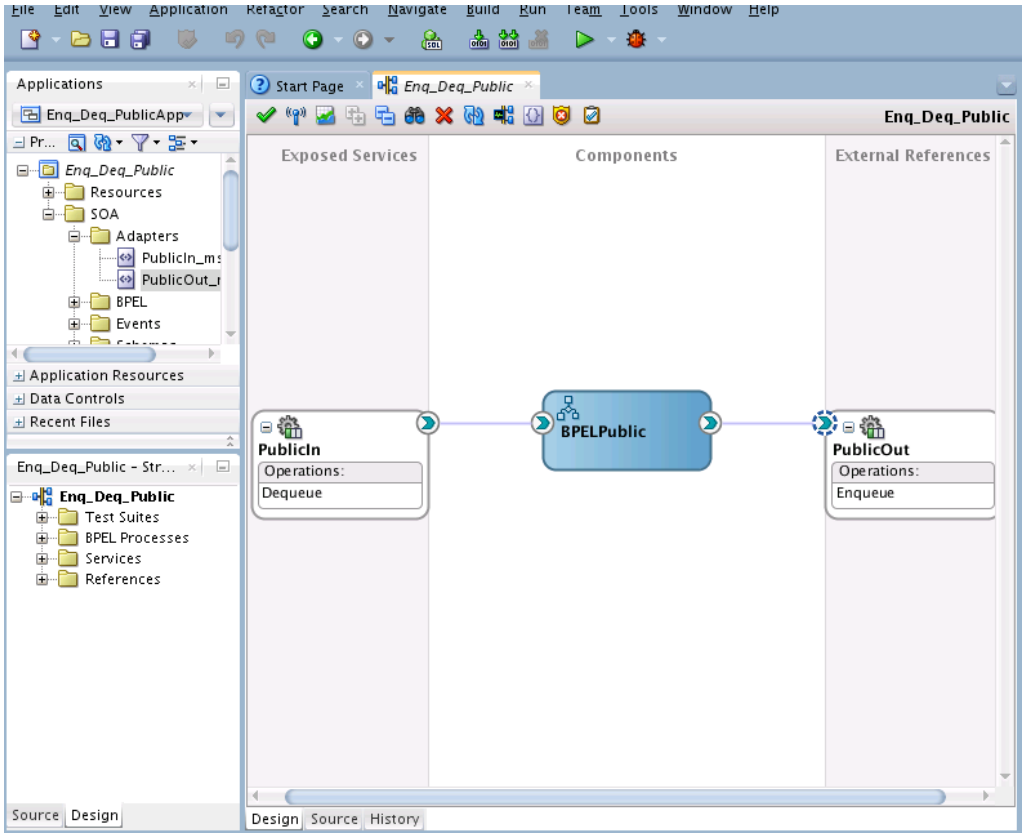
10. Click **Next**. The MSMQ Adapter **Messages** page is displayed
11. Select the **Native format translation is not required (Schema is Opaque)** checkbox. Note that if you have a schema, you can provide the schema for translation
12. Click **Next**. The Adapter Configuration Wizard displays the **Finish** page.
13. Click **Finish**. The outbound Oracle MSMQ Adapter is now configured and the `composite.xml` appears.

Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components.

1. Drag the small triangle in the PublicIn in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Choose **Delivery Type** as `async.persist`.
3. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the `PublicOut` in the External References area. The JDeveloper `composite.xml` appears, as shown in Figure 13-10.

Figure 13-10 Wiring the composite.xml



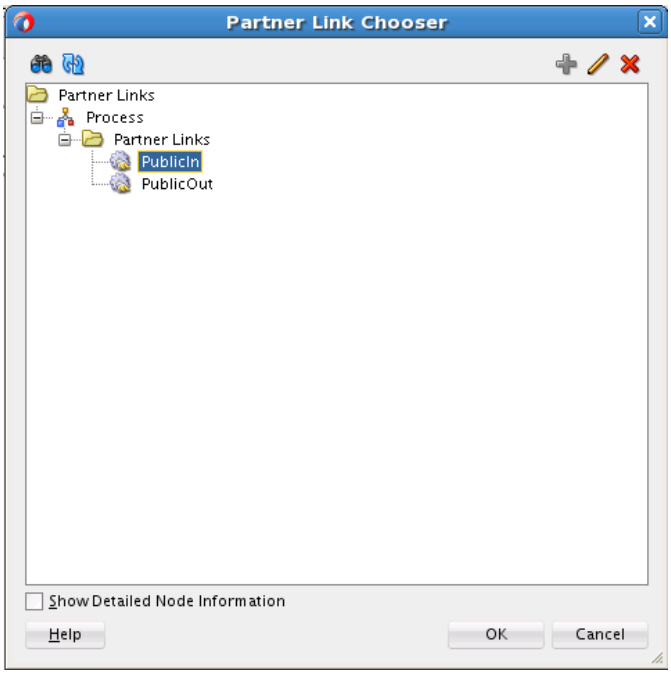
4. Click **File**, **Save All**.

Add a Receive Activity

Follow these steps to add a Receive Activity.

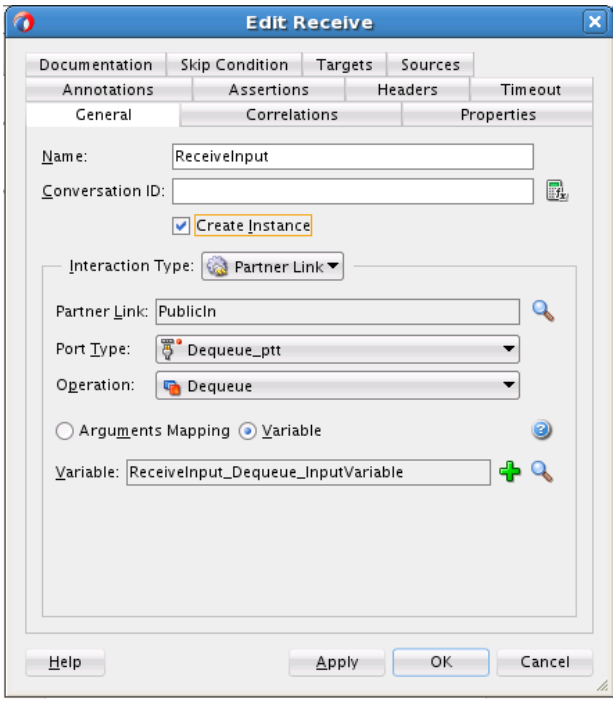
1. Double-click `BPELPublic`. The **BPELPublic.bpel** page is displayed.
2. Drag and drop a Receive activity from the Components area to the design area.
3. Double-click the Receive activity. The **Receive** dialog is displayed.
4. Enter `ReceiveInput` in the **Name** field.
5. Click **Browse Partner Links** at the end of the **Partner Link** field. The **Partner Link Chooser** dialog is displayed.
6. Select `PublicIn`, as shown in Figure 13-11 and click **OK**.

Figure 13-11 Selecting PublicIn in Partner Link Chooser



7. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog, as shown in Figure 13-12. The **Create Variable** dialog is displayed.

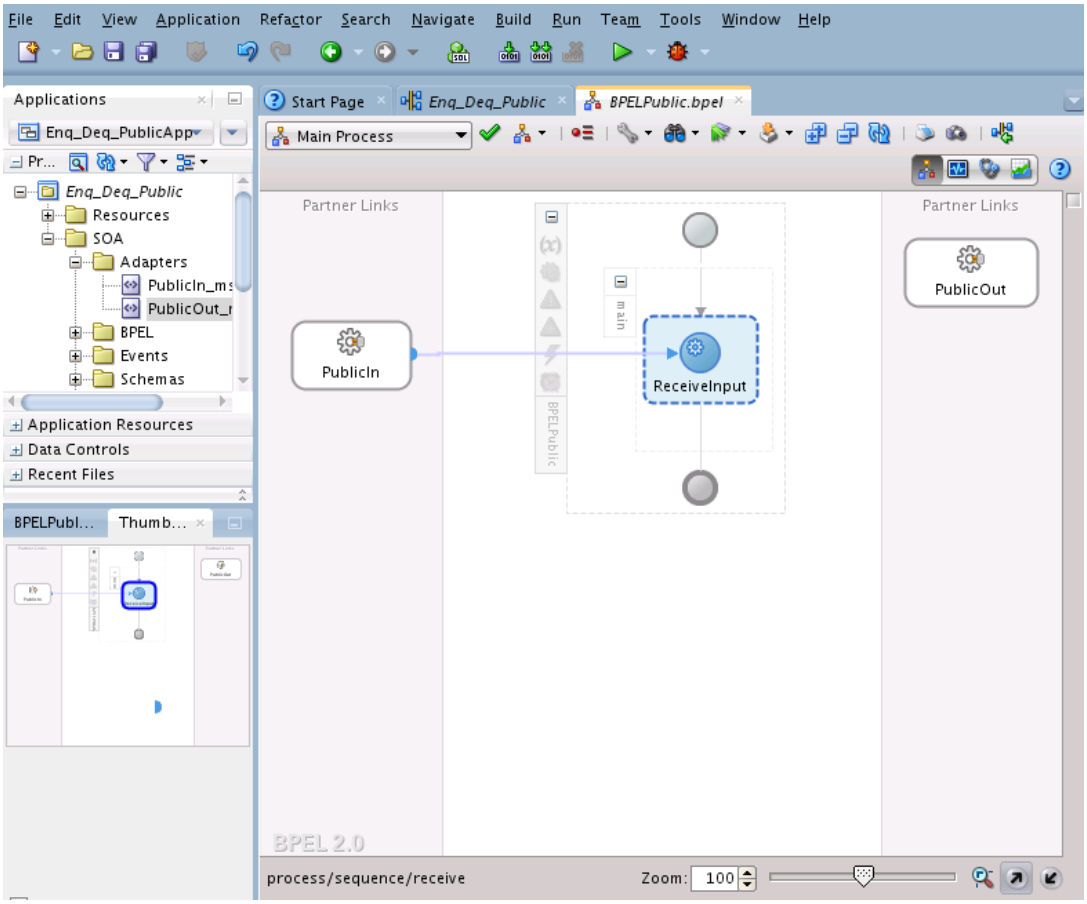
Figure 13-12 Clicking Auto-Create Variable Icon in Receive Dialog



8. Select the default variable name and click **OK**. The **Variable** field is populated with the default variable name.

9. Check **Create Instance**, and click **OK**. The JDeveloper **BPELPublic.bpel** page appears, as shown in Figure 13-13.

Figure 13-13 The JDeveloper BPELPublic.bpel Page

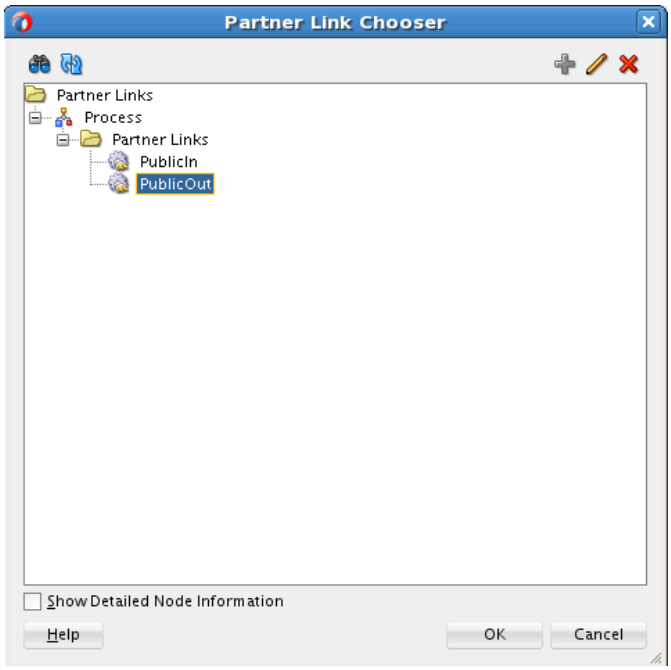


Add an Invoke Activity

Next step is to add an invoke activity.

1. Drag and drop an **Invoke** activity from the **Components** area to the design area.
2. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.
3. Enter `Enqueue_Message` in the **Name** field.
4. Click **Browse Partner Links** at the end of the **Partner Link** field. The **Partner Link Chooser** dialog is displayed.
5. Select `PublicOut`, as shown in Figure 13-14, and click OK.

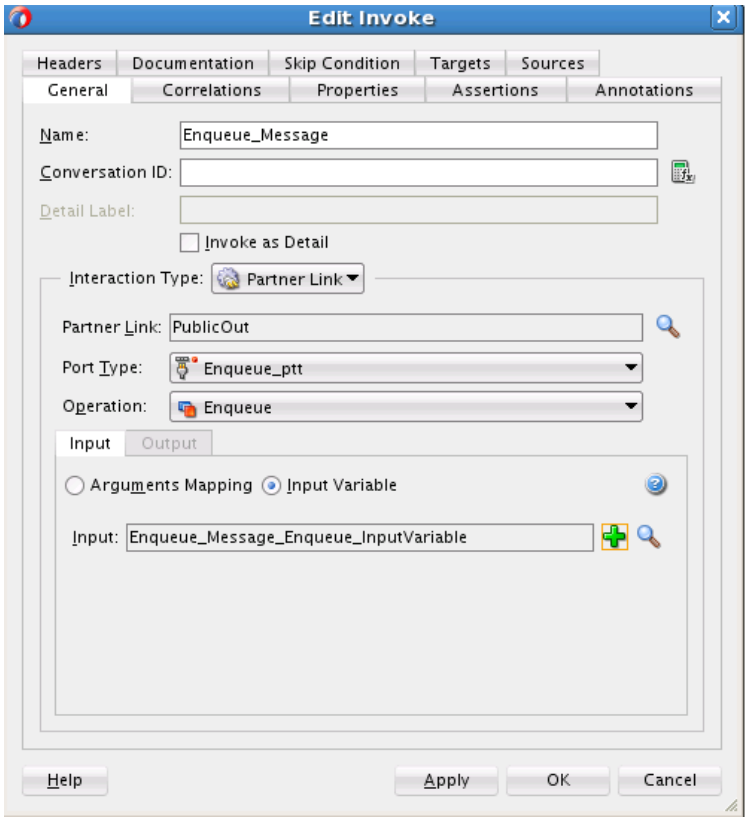
Figure 13-14 Partner Link Chooser Dialog with PublicOut Selected



6. Click the **Automatically Create Input Variable** icon to the right of the **Input** variable field in the Invoke dialog. The **Create Variable** dialog is displayed.

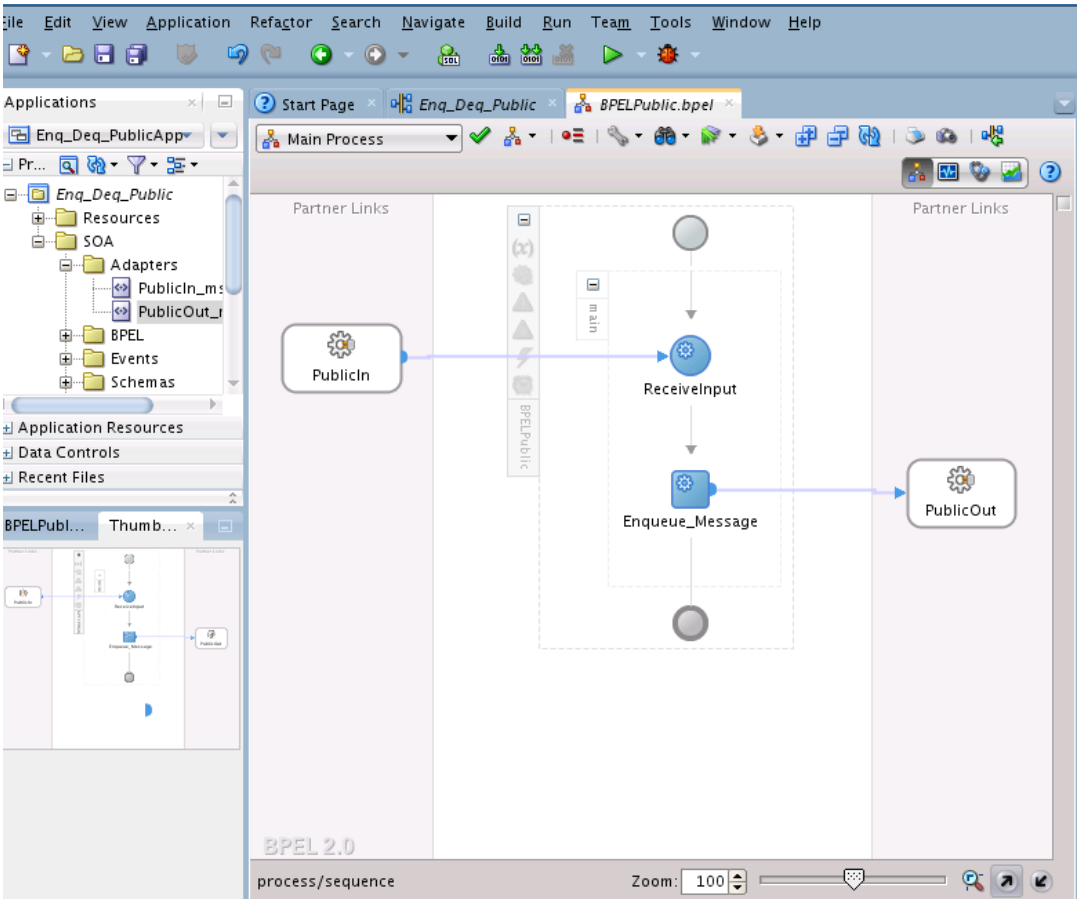
7. Select the default variable name and click OK. The **Variable** field is populated with the default variable name. The **Invoke** dialog is displayed, as shown in Figure 13-15.

Figure 13-15 Invoke Dialog for Enqueue Message



8. Click **OK**. The JDeveloper **BPELPublic.bpel** page appears. See Figure 13-16.

Figure 13-16 The BPELPublic.bpel Page after Invoke Has Been Added

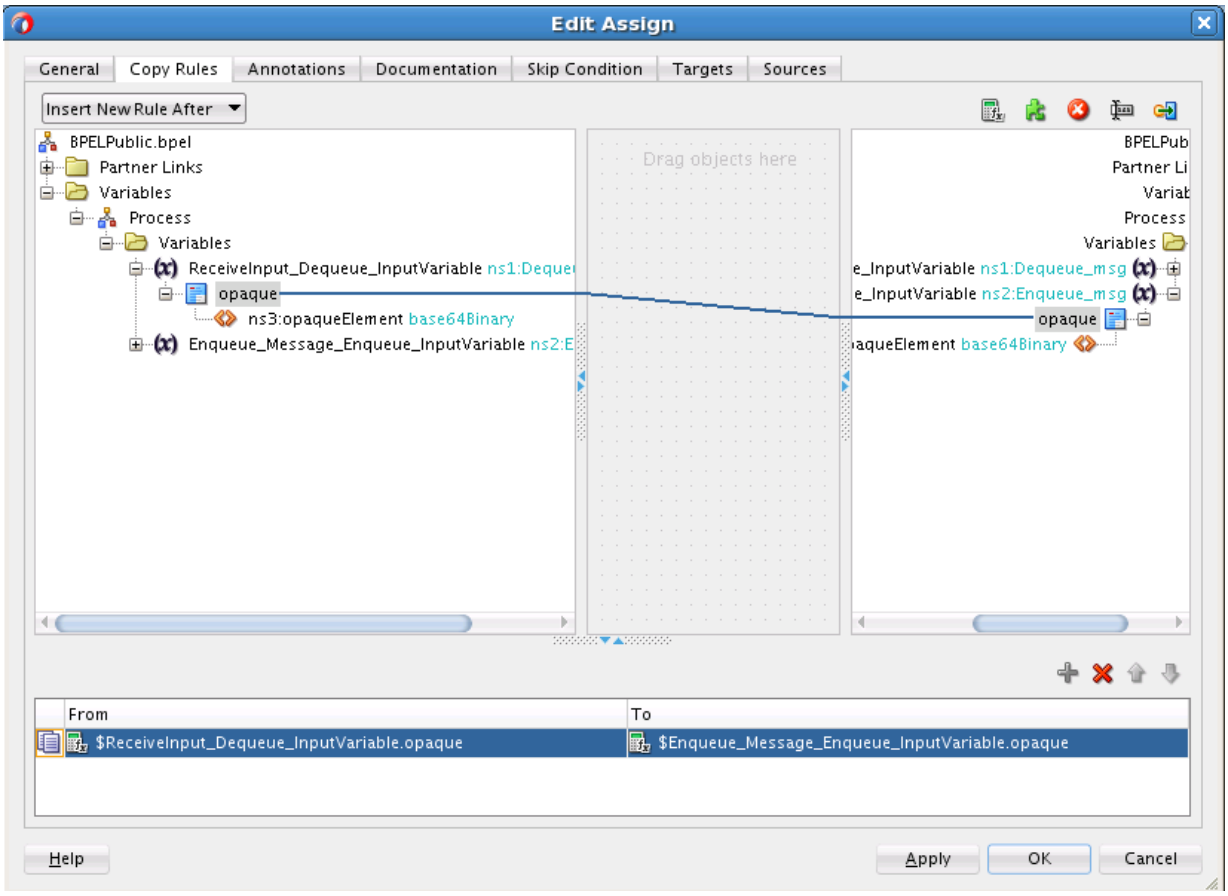


Add an Assign Activity

Next, you add an Assign activity. Follow these steps to do so.

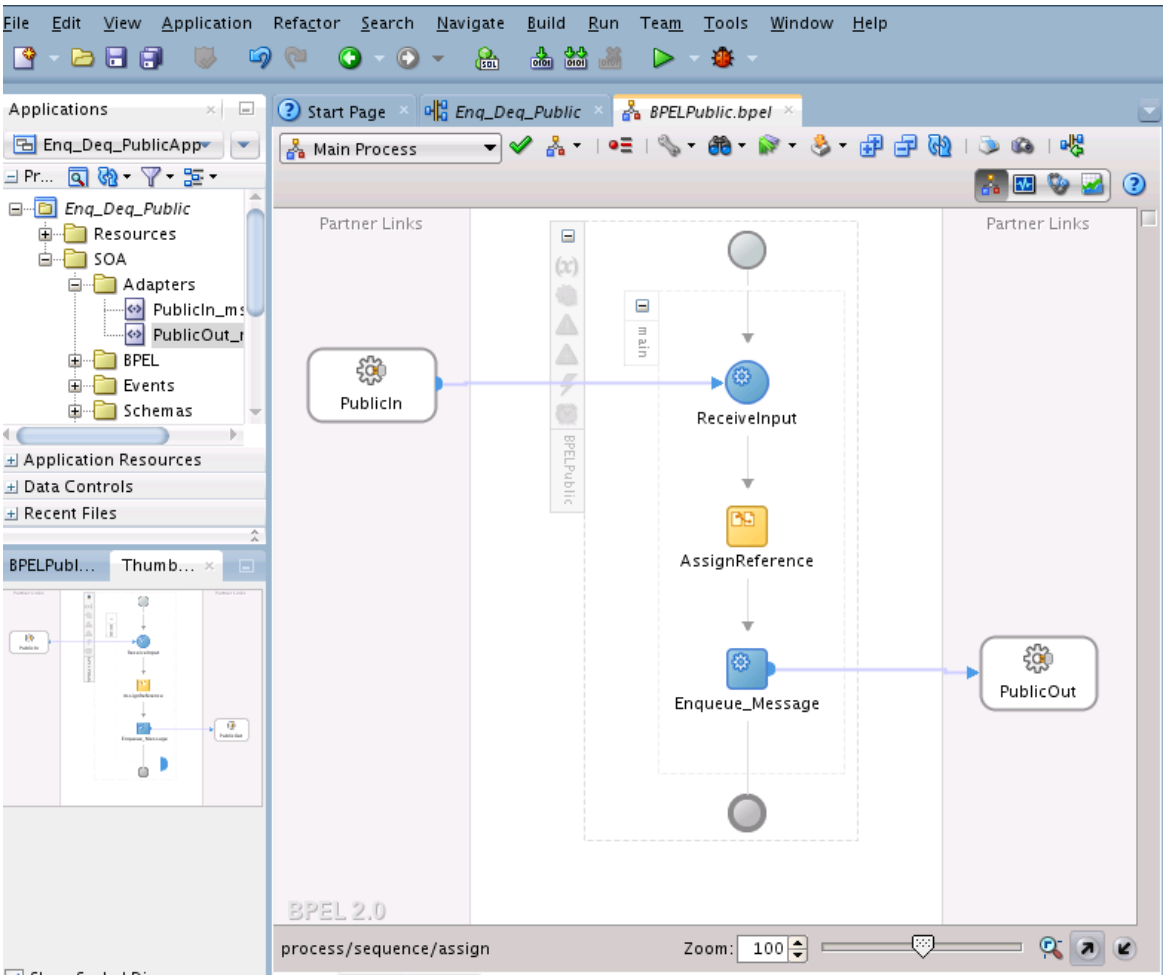
1. Drag and drop an **Assign** activity from the Components area in between the **Receive** and **Invoke** activities in the design area.
2. Double-click the **Assign** activity. The **Assign** dialog is displayed.
3. Click the **General** tab and enter `AssignReference` in the **Name** field.
4. Click the **Copy Rules** tab.
5. Wire between the opaque element of the variable `ReceiveInput_Dequeue_InputVariable` to opaque element of the variable `Enqueue_Message_Enqueue_InputVariable` as shown in Figure 13-17.

Figure 13-17 Using the Assign Dialog to Wire Elements



6. Click **OK**, the JDeveloper **BPELPublic.bpel** page is displayed, as shown in Figure 13-18.

Figure 13-18 The BPELPublic.bpel Page After the Assign Dialog has Wired Elements



Enqueue/Dequeue Message from Private Queue

In this use case, you create an application that enqueues and dequeues messages from an MSMQ private queue.,

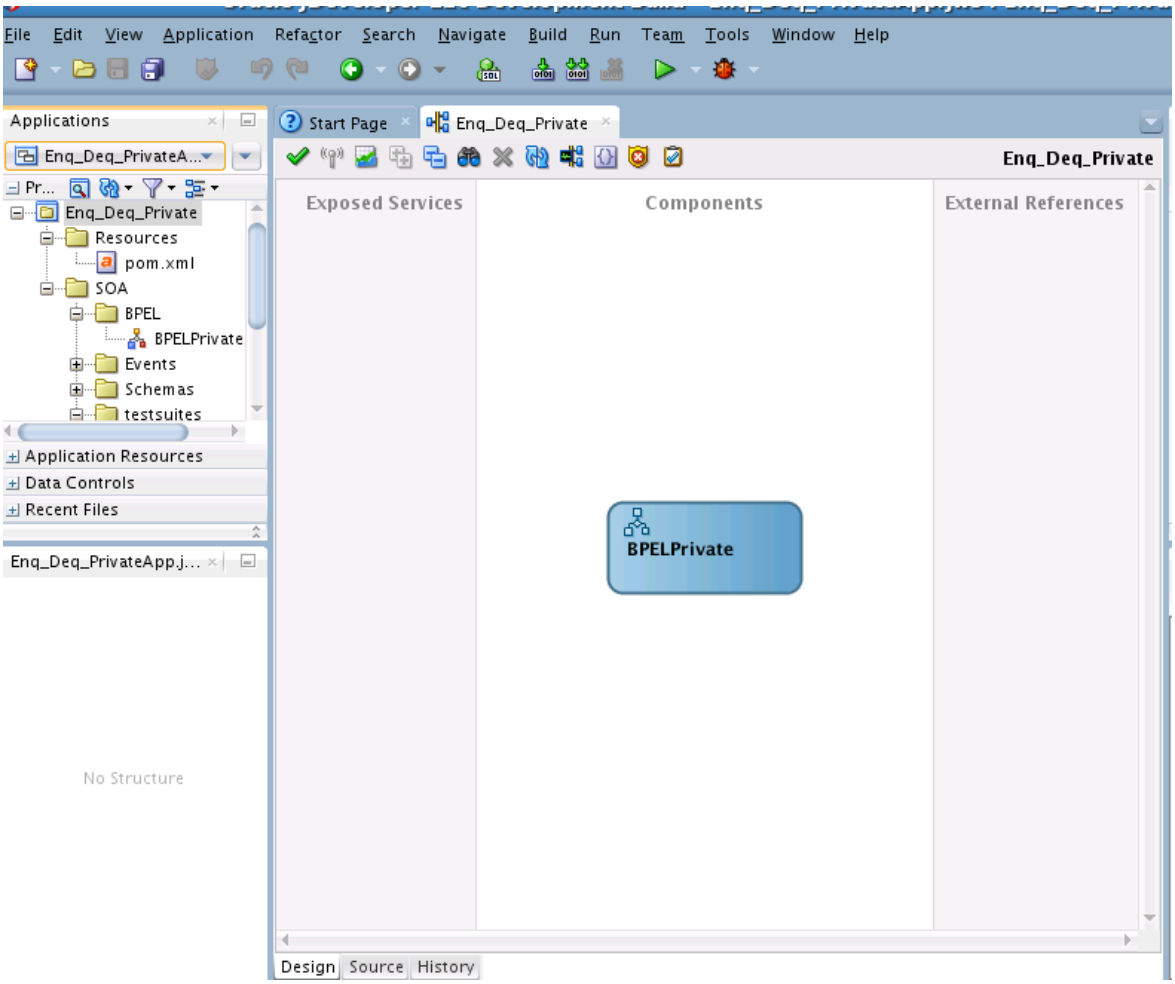
- Designing the SOA Composite
- Creating the Inbound Oracle MSMQ Adapter Service
- Creating the Outbound Oracle MSMQ Adapter Service
- Wiring Services and Activities
- Adding a Receive Activity
- Adding an Invoke Activity
- Adding an Assign Activity

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the Application Navigator of JDeveloper, click `New Application`. Then select **SOA Application**. The **Name your application** page is displayed. Enter `Enq_Deq_PrivateApp` in the **Application Name** field, and click **Next**.
2. **Name your project** page is displayed. Enter `Enq_Deq_Private` in the **Project Name** field, and click **Next**.
3. Select **Composite With BPEL** in the **Composite Template** box, and click **Finish**. The **Create BPEL Process - BPEL Process** page is displayed.
4. Enter **BPELPrivate in the Name** field, select **Define Service Later** from the **Template** list.
5. Click **OK**. The **Enq_Deq_PrivateApp** application and the **Enq_Deq_Private** project appear in the design area, as shown in Figure 13-19.

Figure 13-19 The Enq_Deq_PrivateApp and the Enq_Deq_Private Project in the Design Area



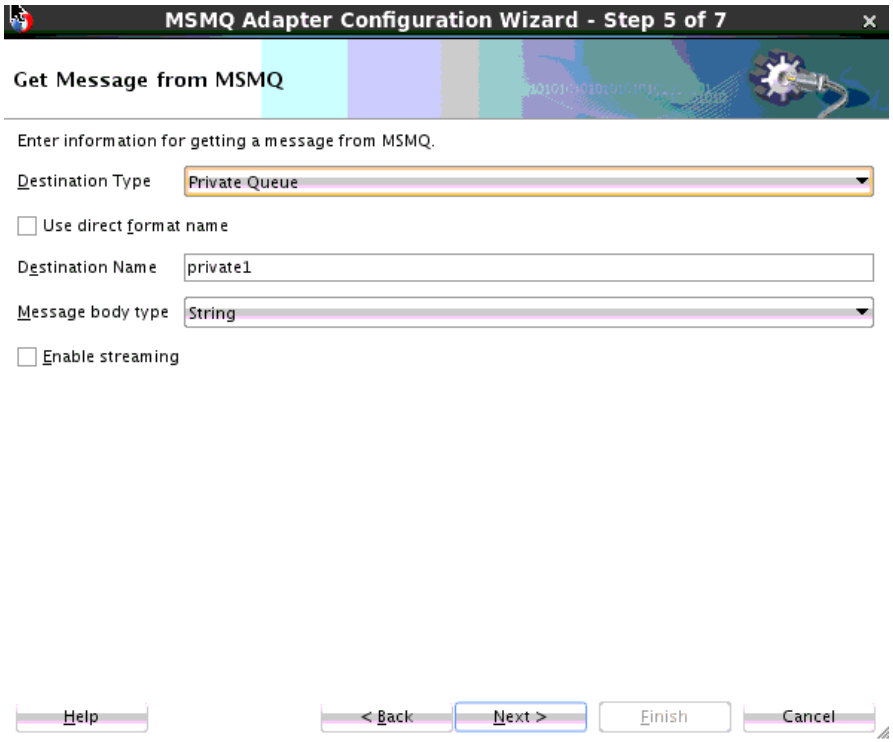
Creating the Inbound Oracle MSMQ Adapter Service

Perform the following steps to create an inbound Oracle MSMQ Adapter service to dequeue message from Microsoft messaging queue.

1. Drag and drop the MSMQ Adapter from the Components area to the Exposed Services swim lane. The Adapter Configuration Wizard **Welcome** page is displayed.

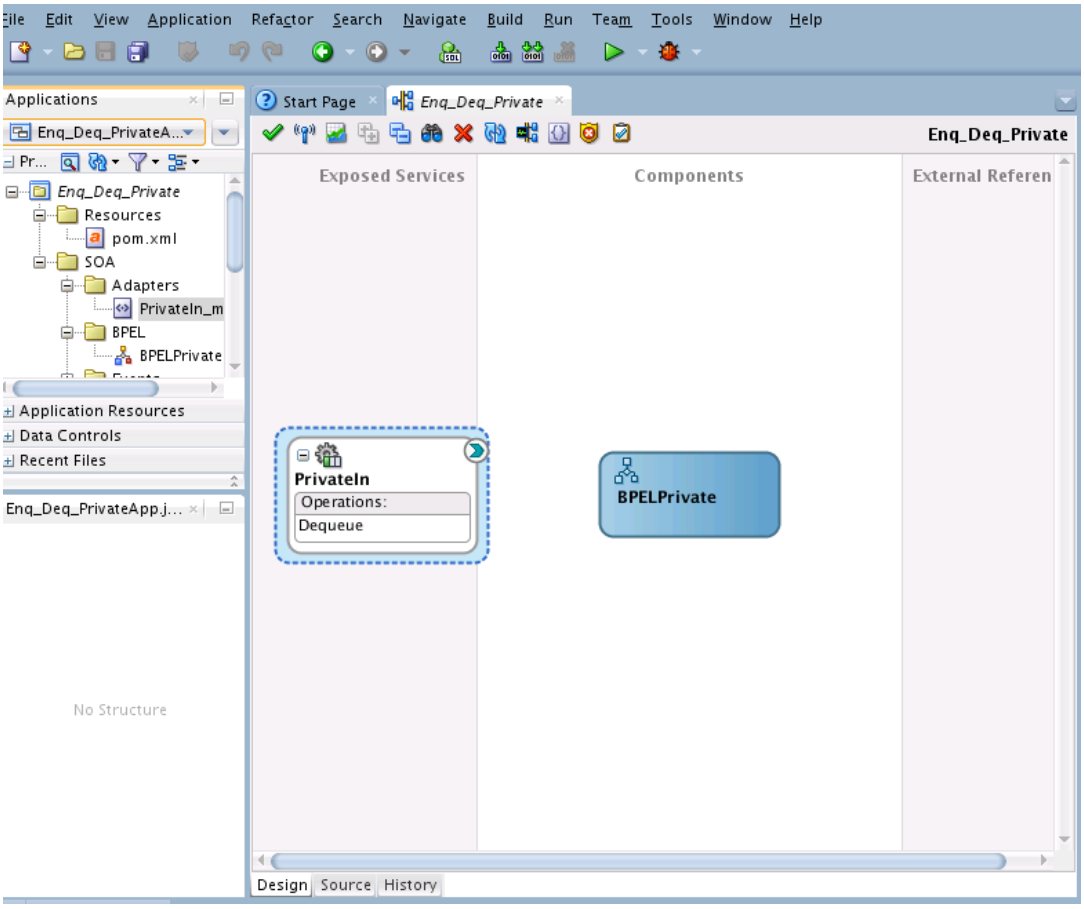
2. Enter `PrivateIn` in the **Service Name** field.
3. Click **Next**. The **Adapter Connection** page is displayed.
4. Provide the JNDI connection name of the MSMQ Server. It can be to connect for Transactional queues or for Non-Transactional queues. In this use case, the JNDI used for Non-Transactional Queue is `eis/MSMQ/MSMQAdapter_NonTrans.`
5. Click **Next**. The Adapter **Interface** page is displayed.
6. Click **Next**. The Adapter **Operation Type** page is displayed
7. Select **Operation Type as Get message from MSMQ** and **Operation Name as Dequeue**. See
8. Click **Next**. The Adapter **Get Message from MSMQ** page is displayed.

Figure 13-20 MSMQ Adapter Configuration Wizard Get Message Screen



9. Select **Destination Type** as `Private Queue`. Enter `Private_Queue_Deq` in the `Destination Name` field. (You might have indicated that the MSMQ adapter uses direct format name to access the private queue. If that option is preferred, you would have to check the **Use Direct Format Name** checkbox.
10. Click **Next**. The Adapter Configuration Wizard displays the Messages page.
11. Select the **Native format translation is not required (Schema is Opaque)** checkbox. Note: If you have a schema, you can provide the schema for translation.
12. Click **Next**. The MSMQ Adapter Configuration Wizard displays the **Finish** page.
13. Click **Finish**. You have now configured the inbound Oracle MSMQ Adapter and the `composite.xml` appears, as shown in Figure 13-21

Figure 13-21 Inbound MSMQ Adapter Configured with composite.xml



Creating the Outbound Oracle MSMQ Adapter Service

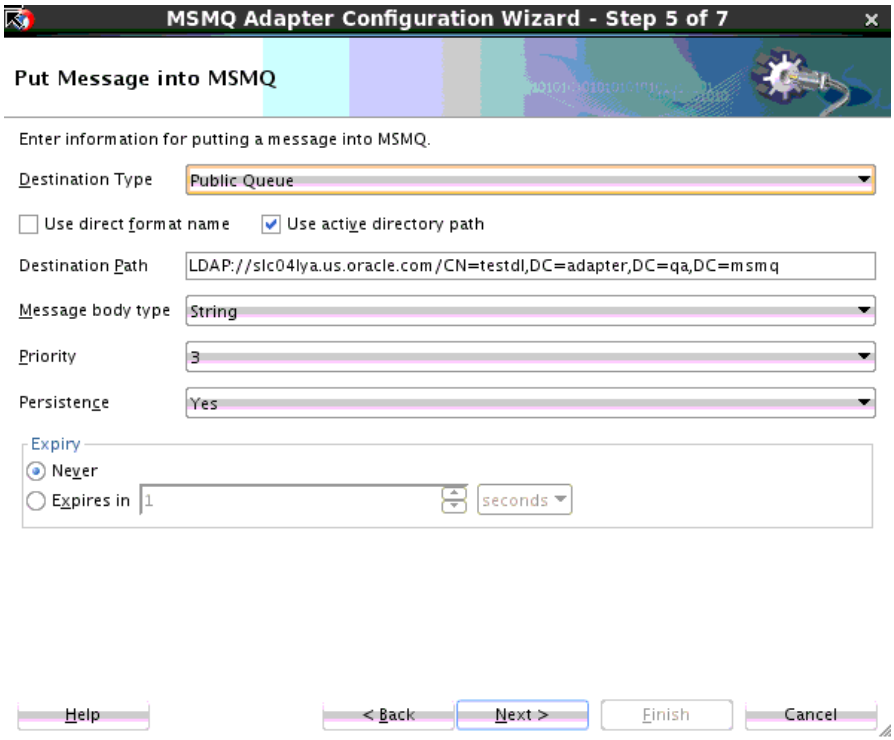
Perform the following steps to create an outbound Oracle MSMQ Adapter service to enqueue the message from one Microsoft messaging queue to other Microsoft messaging queue.

1. Drag and drop MSMQ Adapter from the Components area to the External References swim lane. The Adapter Configuration Wizard **Welcome** page is displayed.
2. Enter `PrivateOut` in the **Reference Name** field.
3. Click Next. The Adapter **Connection** page is displayed.
4. Provide the JNDI connection name of the MSMQ Server. It can be either to connect for Transactional queues or Non-Transactional queues. For example, in this use case the JNDI used for Non-Transactional Queue is `eis/MSMQ/MSMQAdapter_NonTrans`.
5. Click **Next**. The **Interface** page is displayed.
6. Click **Next**. The **Operation Type** page is displayed.
7. Select **Operation Type** as **Put message into MSMQ** and **Operation Name** as **Enqueue**.

8. Click **Next**. The **Adapter Put Message into MSMQ page** is displayed.

9. Select **Destination Type as Private Queue**. Enter `Private_Queue_Enq` in the **Destination Name**. Note: If you have the direct format name of the queue, provide direct format name of the queue by selecting the checkbox **Use direct format name**.

Figure 13-22 MSMQ Adapter Configuration Wizard Put Message Into MSMQ Screen



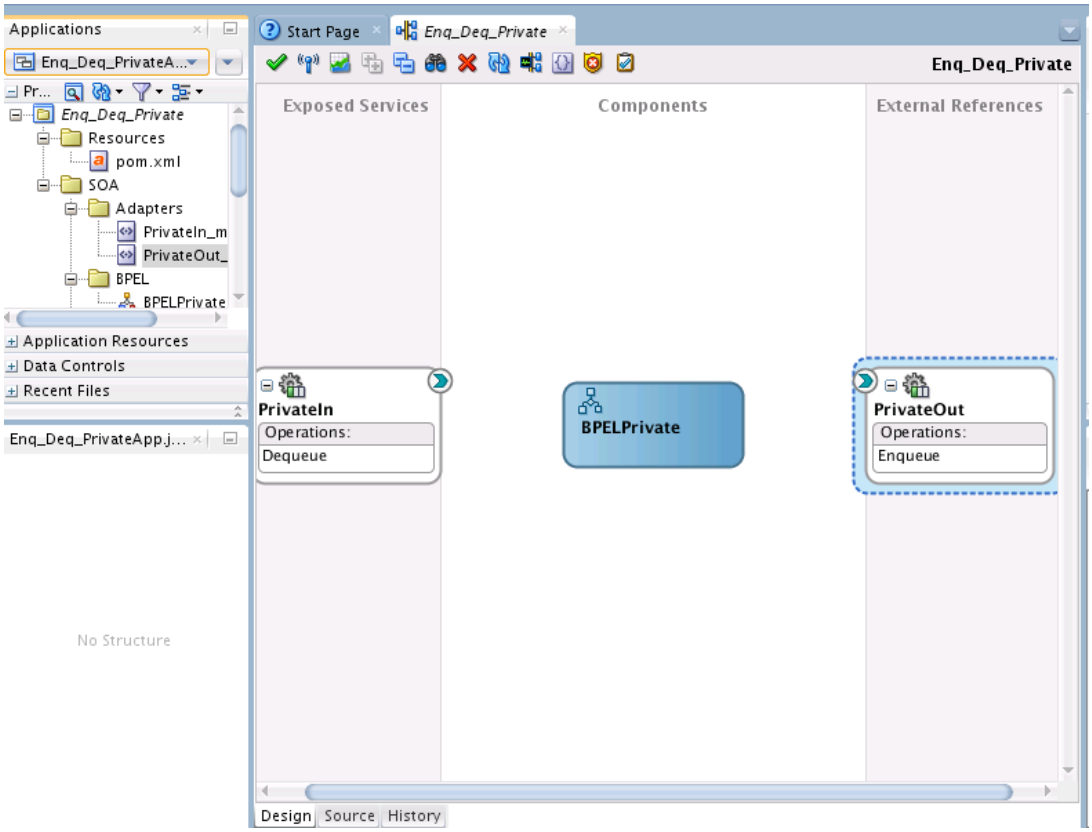
10. Click **Next**. **Messages** page is displayed.

11. Select **Native format translation is not required (Schema is Opaque)** checkbox. Note: If you have a schema, you can provide the schema for translation.

12. Click **Next**. The MSMQ Adapter Configuration Wizard displays the **Finish** page.

13. Click **Finish**. The outbound Oracle MSMQ Adapter is now configured and the `composite.xml` appears, as shown in Figure 13-23.

Figure 13-23 MSMQ Outbound Adapter Configured with Private Out External Reference

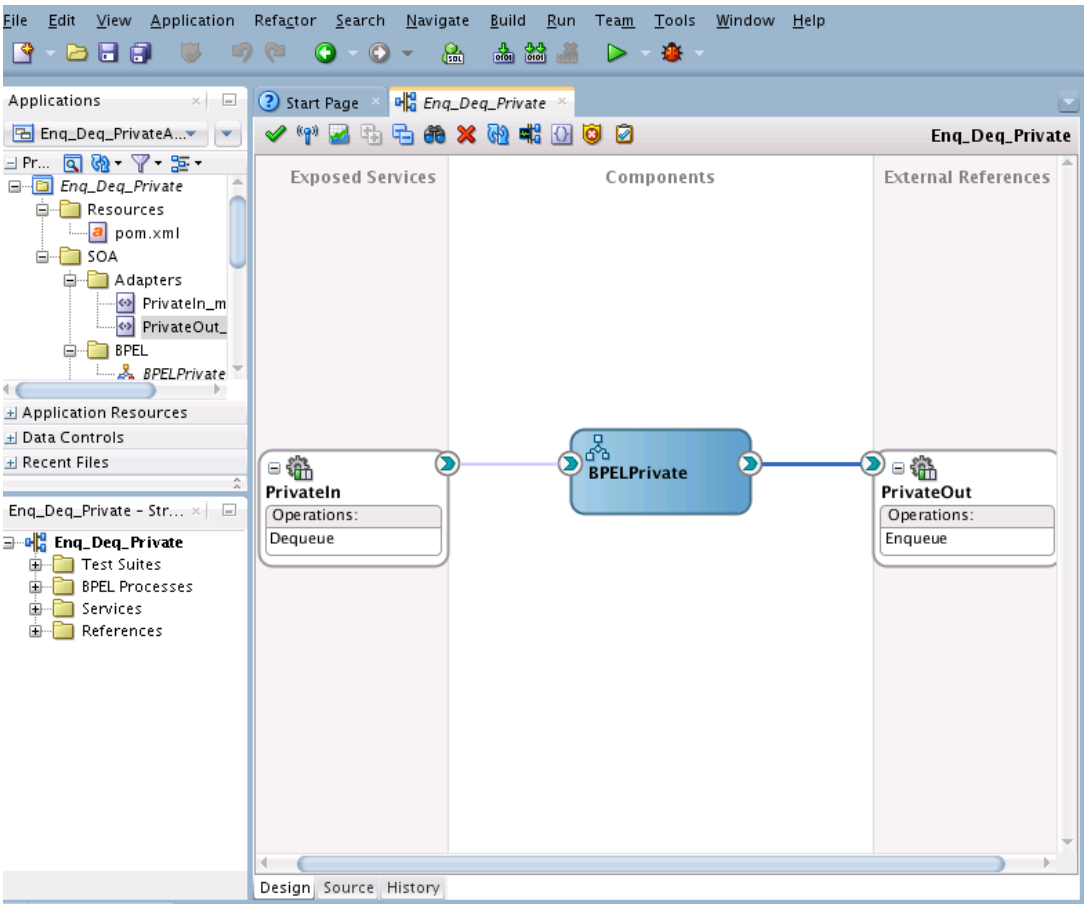


Wiring Services and Activities

You must assemble or wire the three components that you have created: Inbound Adapter service, BPEL process, Outbound Adapter reference. Perform the following steps to wire the components.

1. Drag the small triangle in the `PrivateIn` in the **Exposed Services** area to the drop zone that appears as a green triangle in the BPEL process in the **Components** area.
2. Choose `async.persist` for **Delivery Type**.
3. Drag the small triangle in the BPEL process in the **Components** area to the drop zone that appears as a green triangle in the `PrivateOut` in the External References area. The JDeveloper `composite.xml` appears, as shown in Figure 13-24.

Figure 13-24 The MSMQ Adapter JDeveloper composite.xml Wired



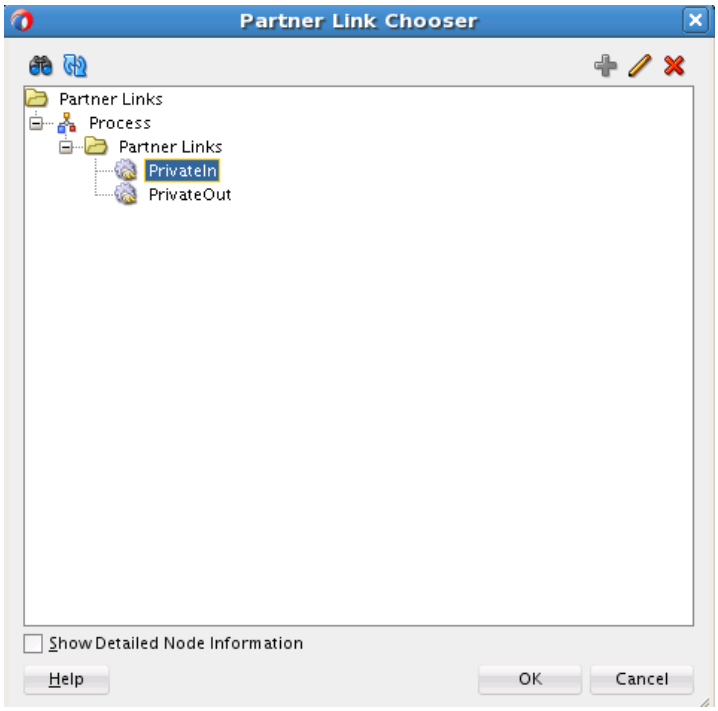
4. Click **File**, **Save All**.

Adding a Receive Activity

The next step is to add a Receive activity to the composite application.

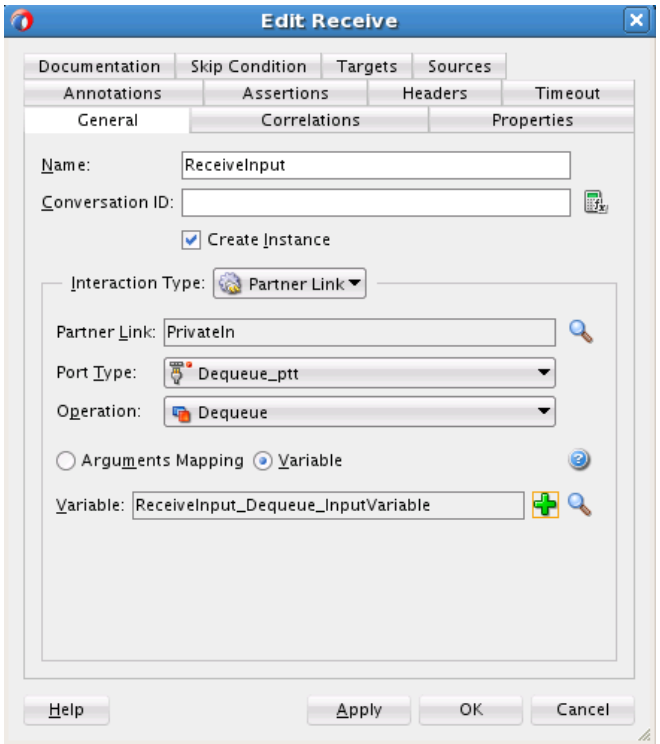
1. Double-click **BPELPrivate**. The **BPELPrivate.bpel** page is displayed.
2. Drag and drop a **Receive** activity from the **Components** area to the design area.
3. Double-click the **Receive** activity. The **Receive** dialog is displayed.
4. Enter `ReceiveInput` in the **Name** field.
5. Click **Browse Partner Links** at the end of the **Partner Link** field. The Configuration Wizard displays the **Partner Link Chooser** dialog.
6. Select `PrivateIn`, as shown in Figure 13-25 and click **OK**.

Figure 13-25 Private Link Chooser Dialog with PrivateIn Selected as Partner LInk



7. Click the **Auto-Create Variable** icon to the right of the **Variable** field in the **Receive** dialog, as shown in Figure 13-26. The Adapter Configuration Wizard displays the **Create Variable dialog**.

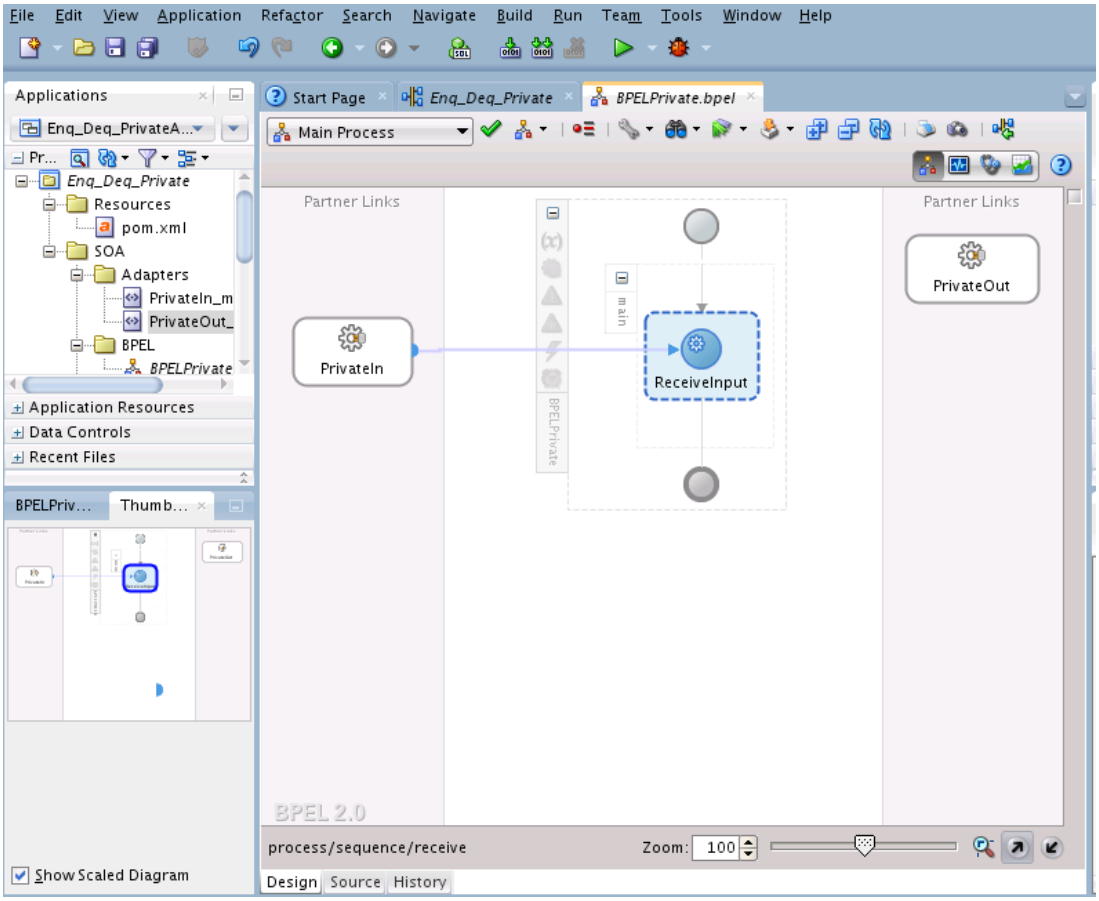
Figure 13-26 The Edit Receive Create Variable Dialog



8. Select the default variable name and click OK. The **Variable** field is populated with the default variable name.

9. Check **Create Instance**, and click **OK**. The JDeveloper **BPELPrivate.bpel** page appears, as shown in Figure 13-27.

Figure 13-27 The BPELPrivate.bpel Page, Showing the PrivateIn Partner Link Wired to the Receive Activity



Adding an Invoke Activity

The next step is to add an Invoke Activity.

1. Drag and drop an Invoke activity from the **Components** area to the design area.

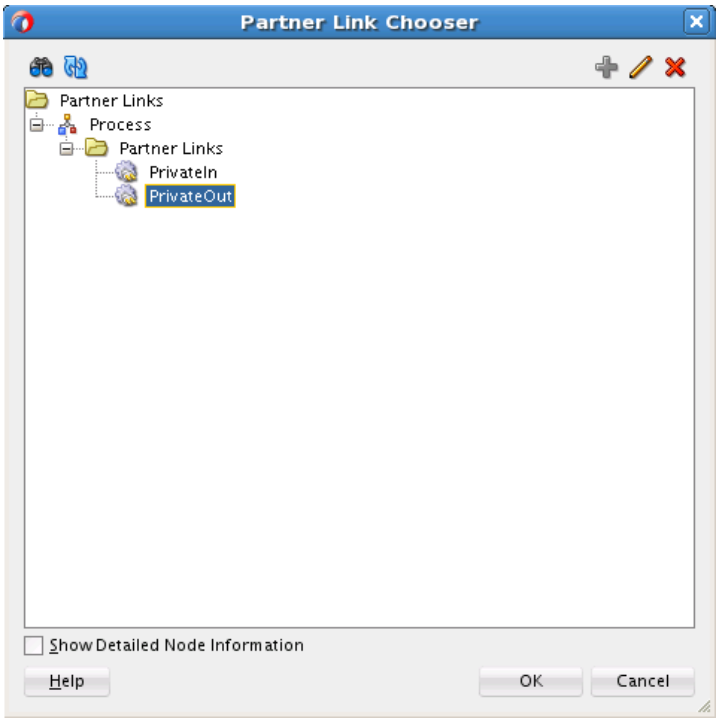
2. Double-click the Invoke activity. The **Invoke** dialog is displayed.

3. Enter `Enqueue_Message` in the **Name** field.

4. Click **Browse Partner Links** at the end of the **Partner Link** field. The **Partner Link Chooser** dialog is displayed

5. Select `PrivateOut`, as shown in Figure 13-28, and click **OK.**

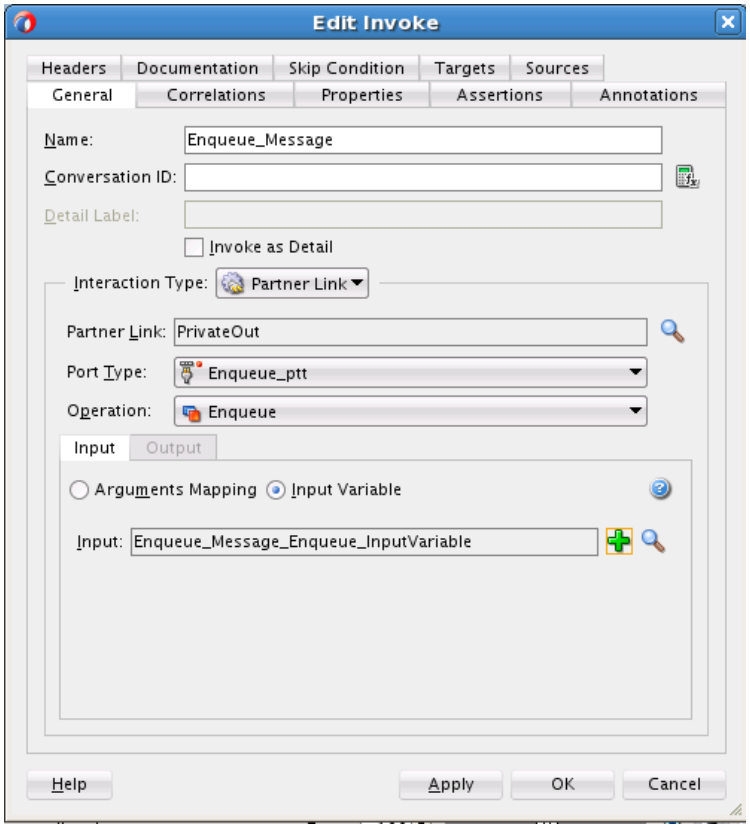
Figure 13-28 The Partner Link Chooser Dialog, with PrivateOut Selected



6. Click the **Automatically Create Input Variable** icon to the right of the **Input** variable field in the **Invoke** dialog. The **Create Variable** dialog is displayed.

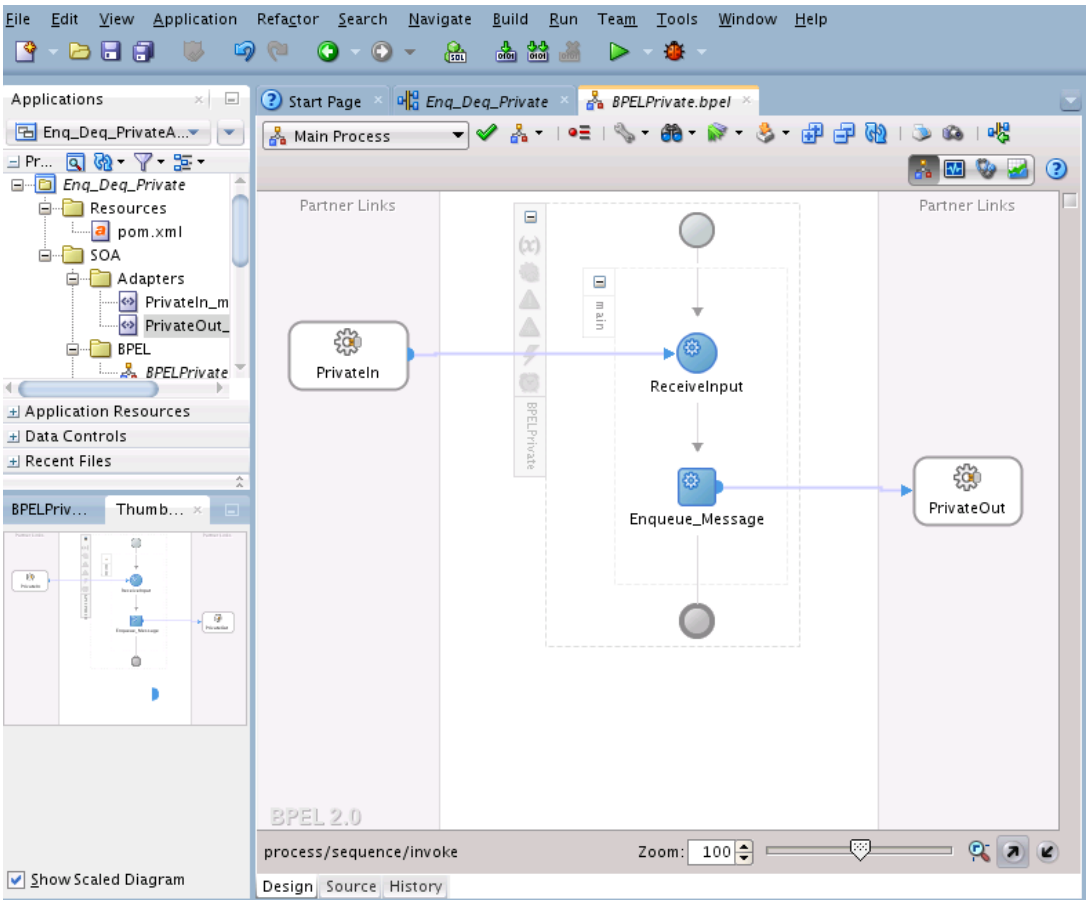
7. Select the default variable name and click **OK**. The **Variable** field is populated with the default variable name. The **Invoke** dialog is displayed, as shown in Figure 13-29.

Figure 13-29 The Edit Invoke Dialog



8. Click **OK**. The JDeveloper **BPELPrivate.bpel** page appears, as shown in Figure 13-30.

Figure 13-30 BPELPrivate.bpel Page After Invoke Activity Has Been Added

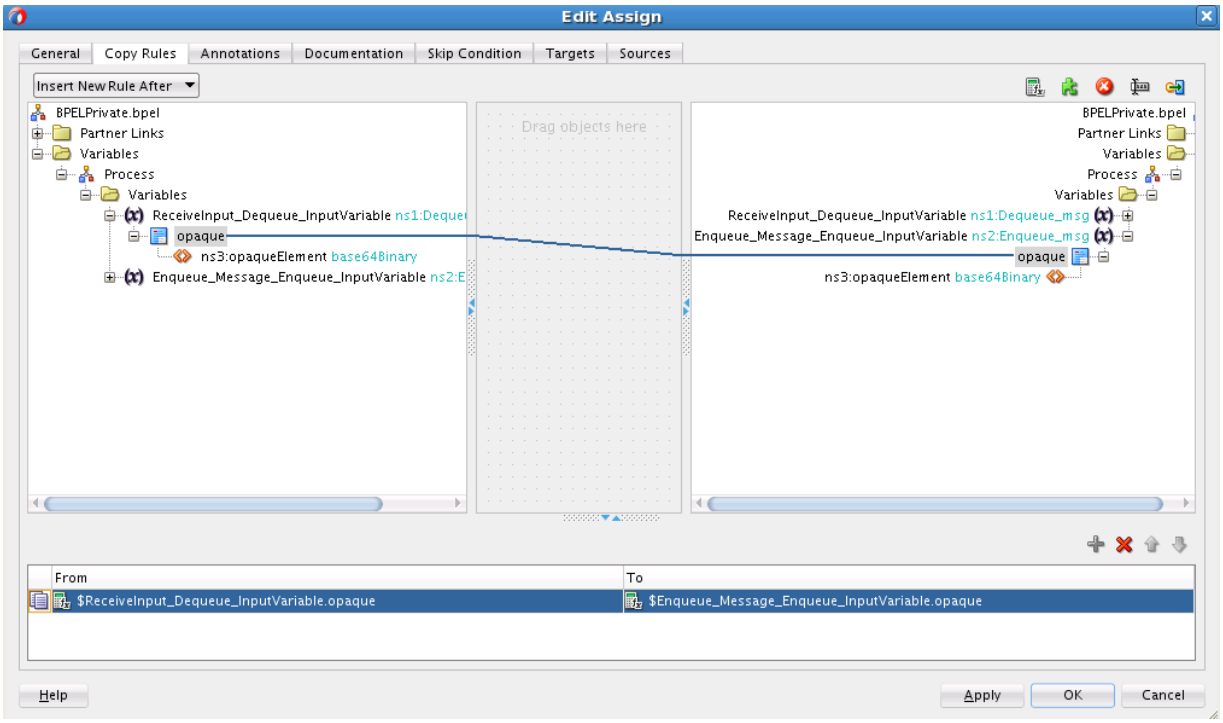


Adding an Assign Activity

The final step is to add an Assign activity.

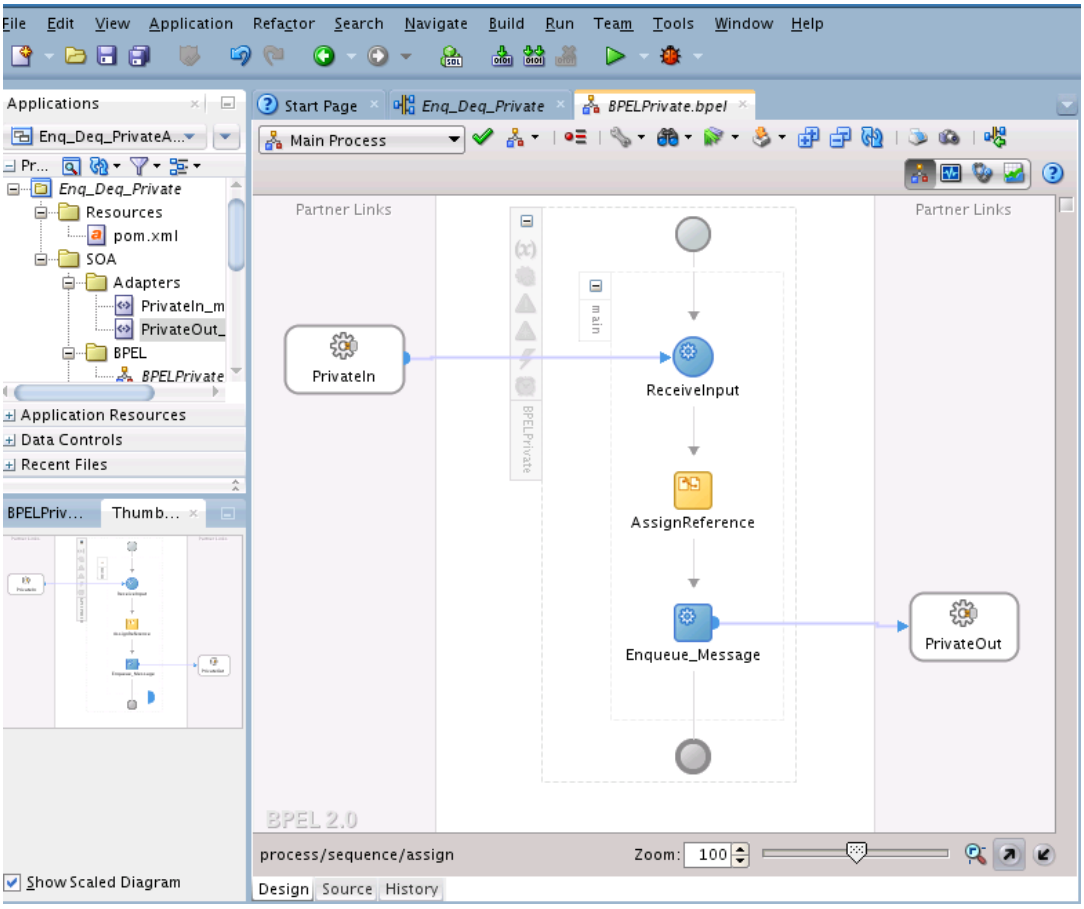
1. Drag and drop an **Assign** activity from the **Components** area in between the **Receive** and **Invoke** activities in the design area.
2. Double-click the **Assign** activity. The **Assign** dialog is displayed.
3. Click the **General** tab and enter AssignReference in the **Name** field.
4. Click the **Copy Rules** tab.
5. Wire between the opaque element of the variable `ReceiveInput_Dequeue_InputVariable` to the opaque element of the variable `Enqueue_Message_Enqueue_InputVariable` as shown in Figure 13-31.

Figure 13-31 Editing the Assign Activity to Wire Variables



6. Click **OK**, the JDeveloper **BPELPrivate.bpel** page is displayed.

Figure 13-32 The BPELPrivate.bpel Page After the Variables Are Assigned



7. Click **File**, **Save All**.

Enqueuing a Message to a Distribution List

Follow these steps to design a use case that includes enqueueing MSMQ messages to a distribution list.

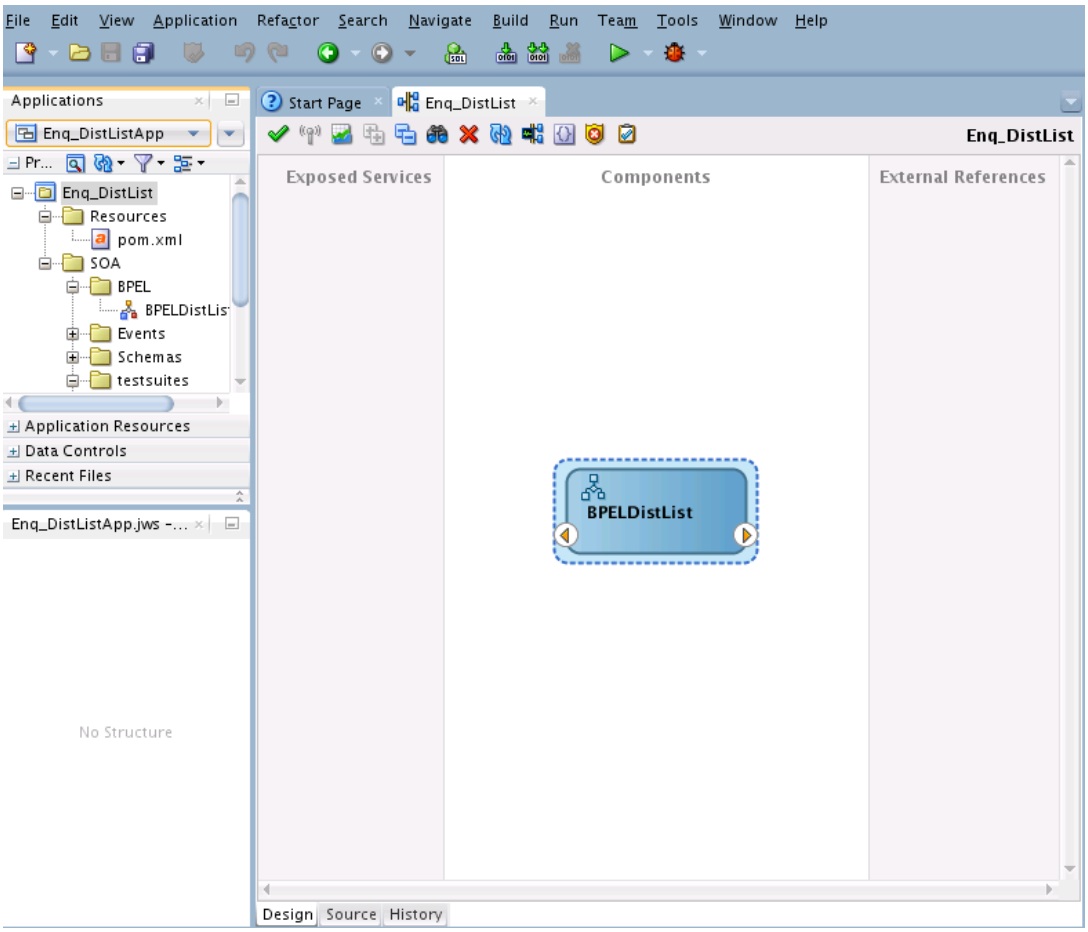
- Designing the SOA Composite
- Creating the Inbound Oracle File Adapter Service
- Creating the Outbound Oracle MSMQ Adapter Service
- Wiring Services and Activities
- Adding a Receive Activity
- Adding an Invoke Activity
- Adding an Assign Activity

Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the Application Navigator of JDeveloper, click **New Application**. Then select SOA Application. The Adapter Configuration Wizard displays the **Name your application page**.
2. Enter `Enq_DistListApp` in the **Application Name** field, and click **Next**. The Adapter Configuration Wizard displays **Name your project page**.
3. Enter `Enq_DistList` in the **Project Name** field, and click **Next**.
4. Select **Composite With BPEL** in the **Composite Template** box, and click **Finish**. The **Create BPEL Process - BPEL Process** page is displayed.
5. Enter `BPELDistList` in the **Name** field, select **Define Service Later** from the Template list.
6. Click **OK**. The `Enq_DistListApp` application and the `Enq_DistList` project appear in the design area, as shown in Figure 13-33.

Figure 13-33 The Enq_DistListApp Application and the Enq_DistList Project in the Design Area



Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read file from local directory.

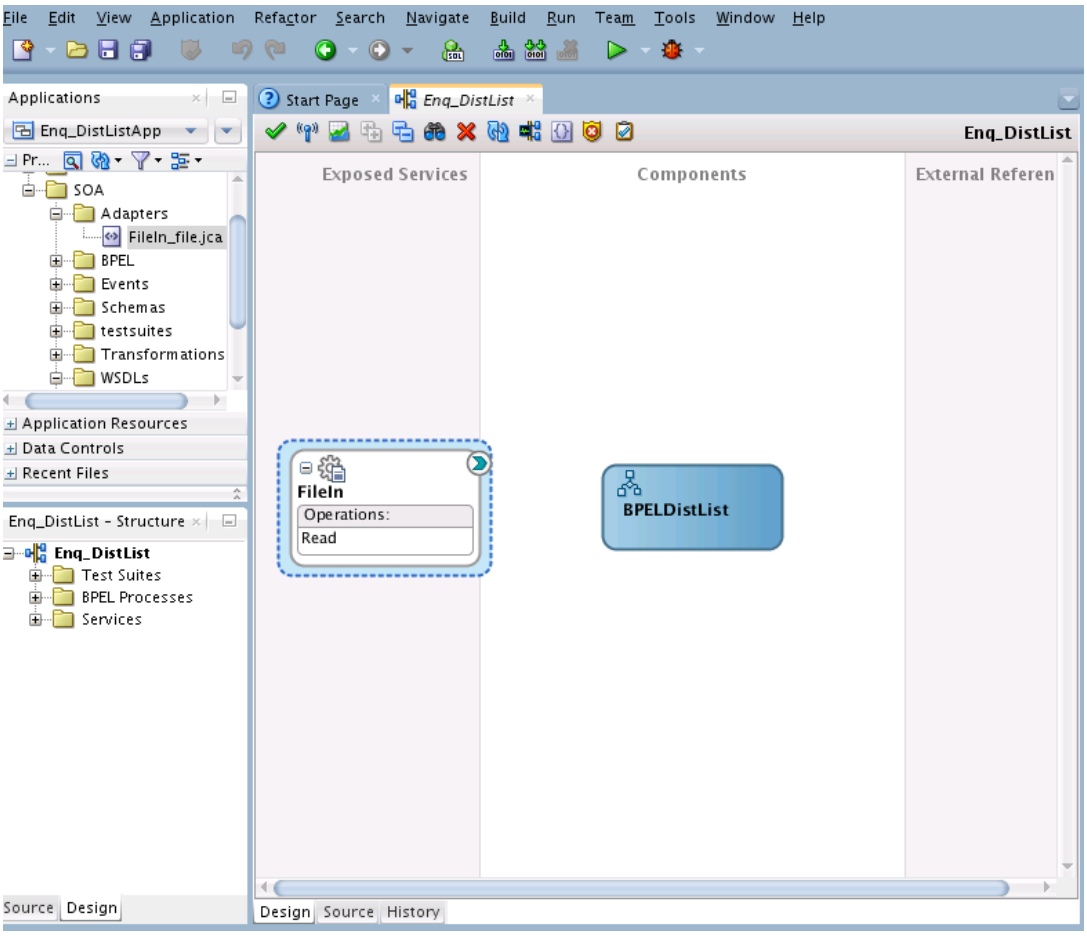
1. Drag and drop a File Adapter from the Components area to the Exposed Services swim lane. The Adapter Configuration Wizard **Welcome** page is displayed.

2. Enter `FileIn` in the **Service Name** field.
3. Click **Next**. The File Adapter Configuration Wizard displays the **File Server Connection** page is displayed.
4. Click **Next**. The File Adapter Configuration Wizard displays the **Operation** page.
5. Select **Operation Type** as `Read File`.
6. Click **Next**. The **File Directories** page is displayed.
7. Enter the physical path for the input directory as shown in Figure 13-34.

Figure 13-34 The File Adapter Configuration Wizard File Directories Page

8. Click **Next**. The File Adapter Configuration Wizard displays the **File Filtering** page.
9. Enter `*.txt` in the **Include Files With Name Pattern** field.
10. Click **Next**. The File Adapter Configuration Wizard displays the **File Polling** page.
11. Click **Next**. The File Adapter Configuration Wizard displays the **Messages** page.
12. Select the **Native format translation is not required (Schema is Opaque)** checkbox.
13. Click **Next**. The File Adapter Configuration Wizard displays the **Finish** page.
14. Click **Finish**. The inbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 13-35.

Figure 13-35 The Configured Inbound File Adapter with the composite.xml



Creating the Outbound Oracle MSMQ Adapter Service

Perform the following steps to create an outbound Oracle MSMQ Adapter service to enqueue the message from a local directory to distribution list:

1. Drag and drop the MSMQ Adapter from the Components area to the External References swim lane. The Adapter Configuration Wizard displays the **Welcome** page.
2. Enter `DistListOut` in the **Reference Name** field.
3. Click **Next**. The Adapter Configuration Wizard displays the **Connection** page.
4. Provide the JNDI connection name of the MSMQ Server. You can use the name to connect either Transactional queues or Non-Transactional queues. In this use case, the JNDI used for Non-Transactional Queue is `eis/forMSMQ/MSMQAdapter_NonTrans`.
5. Click **Next**. The Adapter Configuration Wizard displays the **Interface** page.
6. Click **Next**. The Adapter Configuration Wizard displays the **Operation Type** page.
7. Select **Operation Type** as **Put message into MSMQ** and **Operation Name** as **Enqueue**.

8. Click **Next**. The Adapter Configuration Wizard displays the **Put Message into MSMQ** page.
9. Select **Destination Type** as `Distribution List`. Enter `DistList_Enq` in the Destination Name.

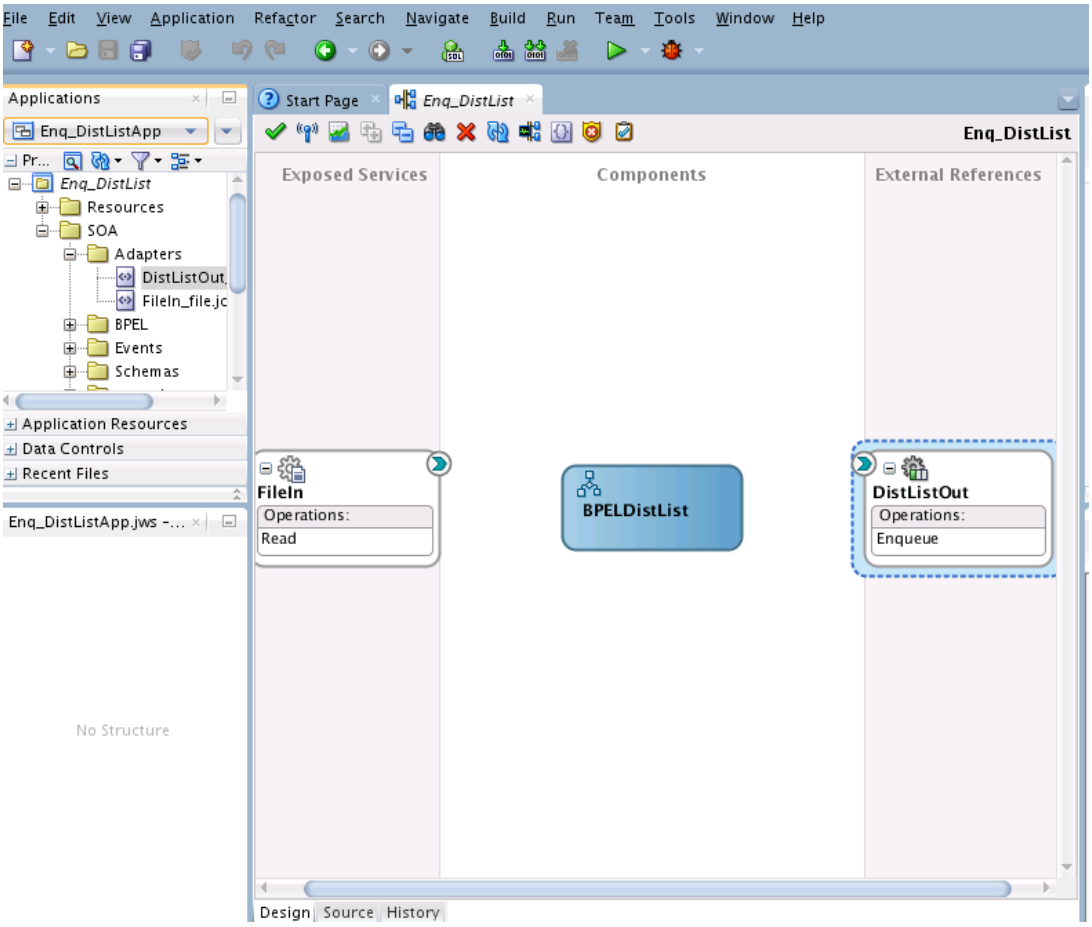
Note:

: If you have active directory path of the distribution list, you must select the **Use active directory path** checkbox and provide the value. Also, if you have the direct format name of the distribution list you can provide direct format name of the queue by selecting the checkbox **Use direct format name**.

---

10. Click **Next**. The Adapter Configuration Wizard displays the Messages page.
11. Select **Native format translation is not required (Schema is Opaque)** checkbox. Note: If you have a schema, you can provide the schema for translation.
12. Click **Next**. The Adapter Configuration Wizard displays the **Finish** page.
13. Click **Finish**. You have configured the outbound Oracle MSMQ Adapter and the `composite.xml` appears, as shown in Figure 13-36.

Figure 13-36 The Configured Outbound File Adapter with the composite.xml

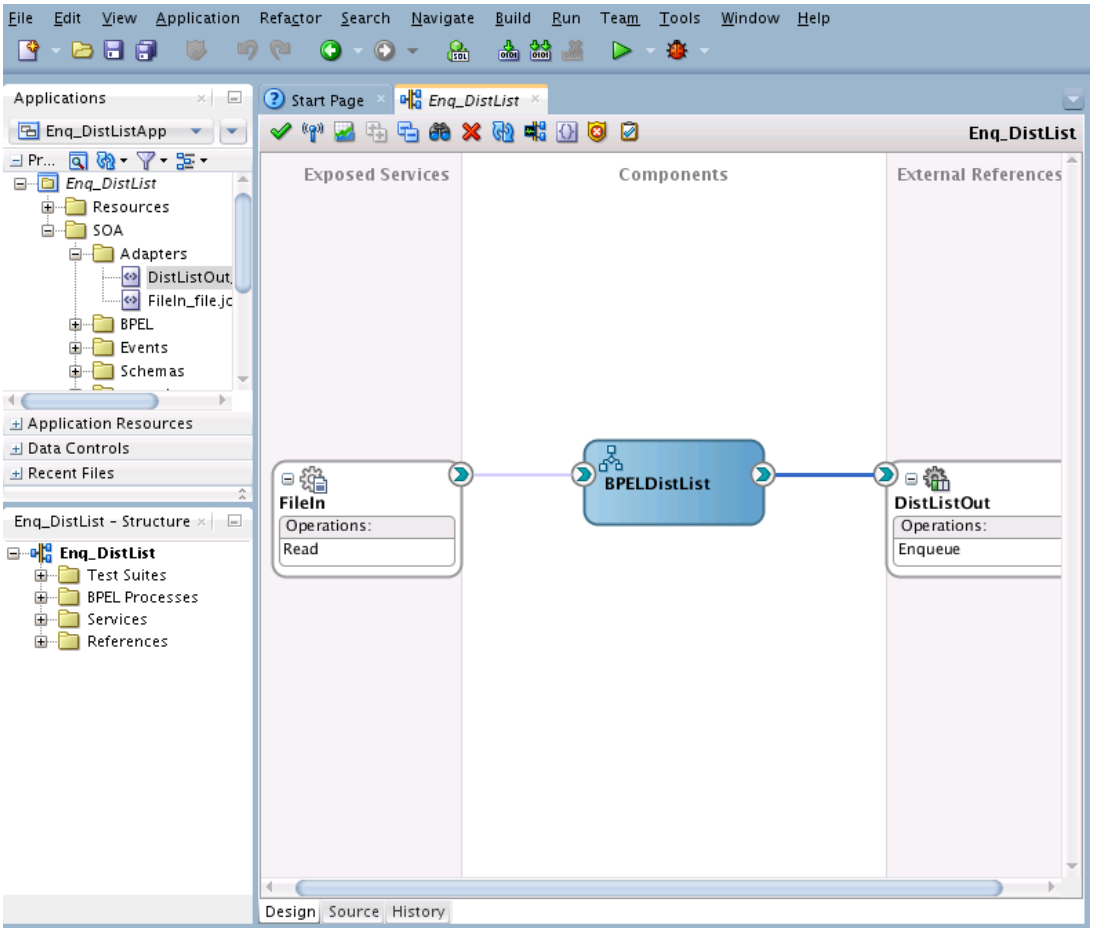


Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, the BPEL process, and the Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the `FileIn` in the **Exposed Services** area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Choose **Delivery Type** as `async.persist`.
3. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the `DistListOut` in the External References area. The JDeveloper `composite.xml` appears, as shown in Figure 13-37.

Figure 13-37 The composite.xml for the Distribution List Use Case After Wiring



4. Click **File**, **Save All**.

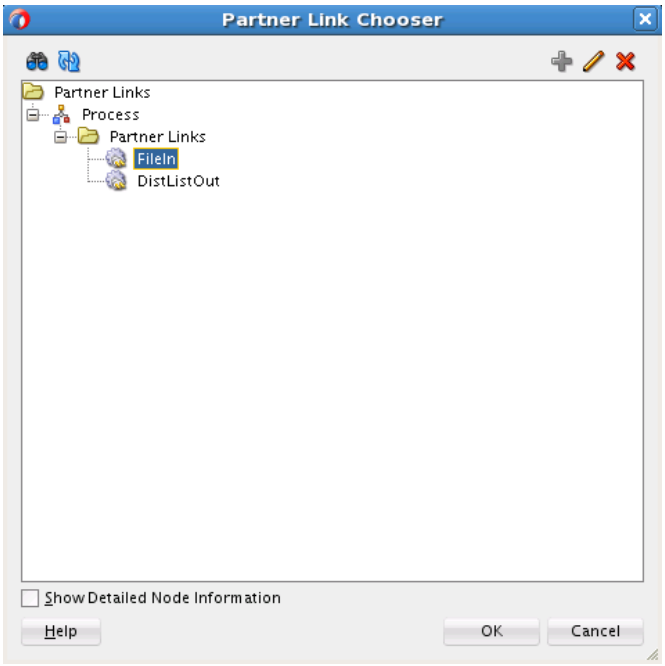
Adding a Receive Activity

Follow these steps to add a Receive Activity.

1. Double-click `BPELDistList`. The **BPELDistList.bpel** page is displayed.

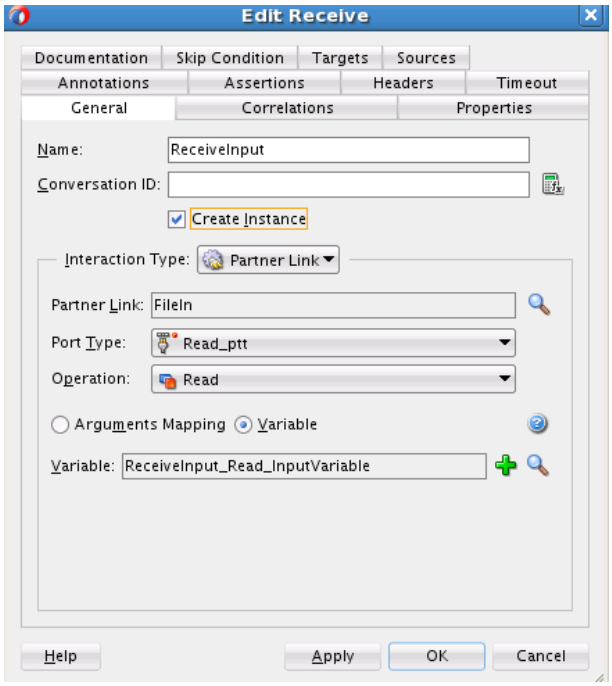
2. Drag and drop a Receive activity from the Component area to the design area.
3. Double-click the **Receive** activity. The MSMQ Adapter Configuration Wizard displays the **Receive** dialog.
4. Enter `ReceiveInput` in the **Name** field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link **Chooser** dialog is displayed.
6. Select `FileIn`, as shown in Figure 13-38 and click **OK**.

Figure 13-38 The Partner Link Chooser Dialog with the Inbound File Adapter Selected



7. Select the **Auto-Create Variable** icon to the right of the **Variable** field in the **Receive** dialog, as shown in Figure 13-39. The **Create Variable** dialog is displayed.

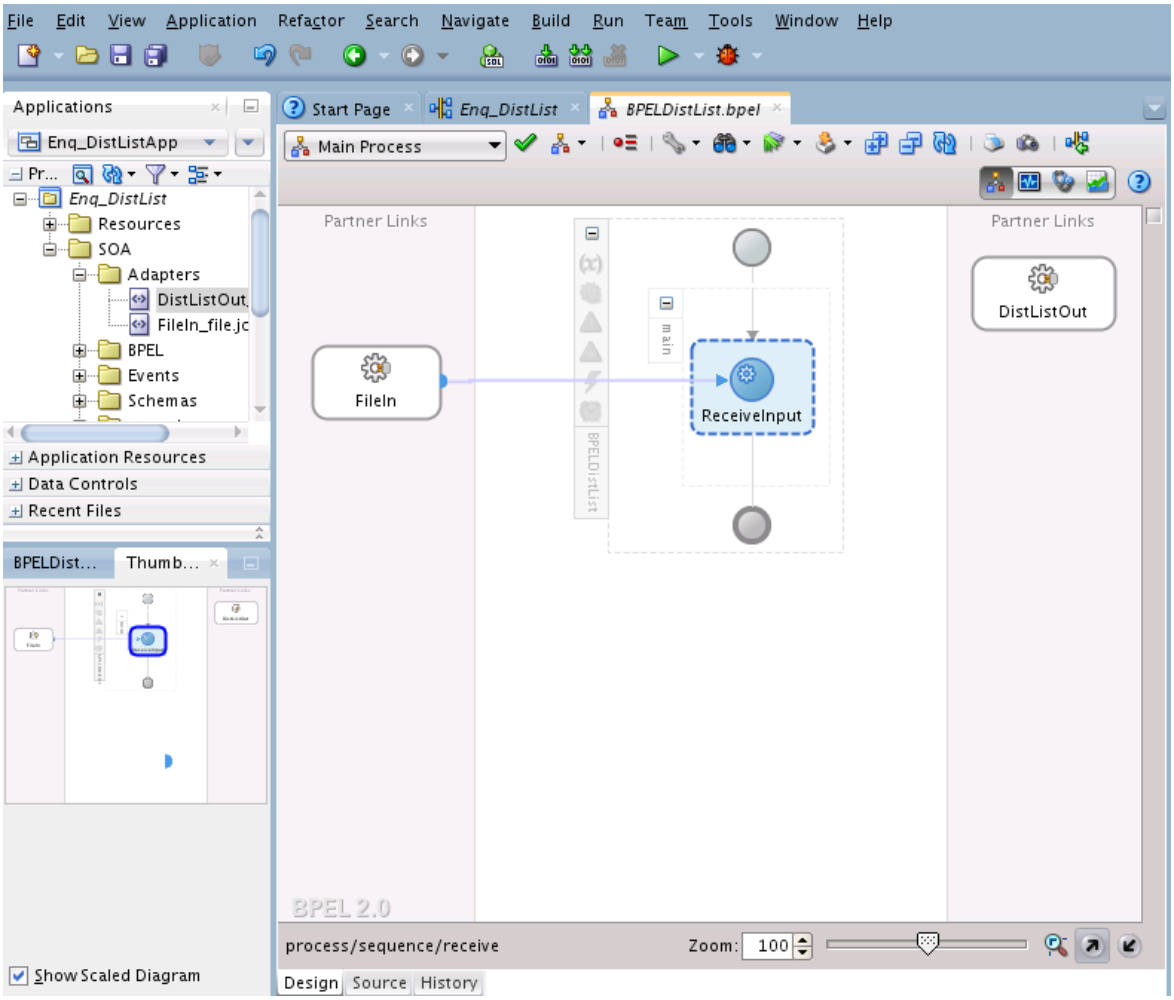
Figure 13-39 The Receive Dialog



8. Select the default variable name and click **OK**. The **Variable** field is populated with the default variable name.

9. Check **Create Instance**, and click **OK**. The JDeveloper **BPELDistList.bpel** page appears, as shown in Figure 13-40.

Figure 13-40 BPELDistList.bpel Page after ReceiveInput Created

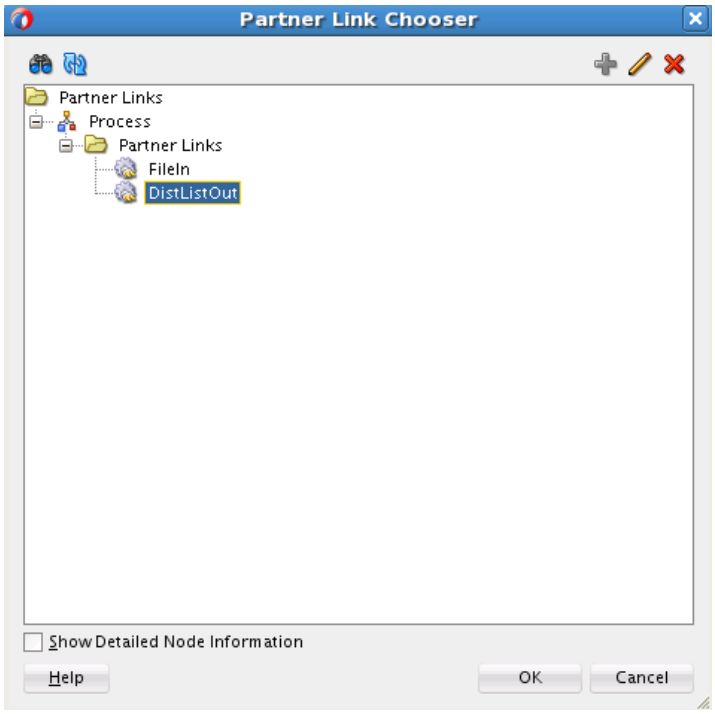


Adding an Invoke Activity

Follow these steps to add an Invoke activity.

1. Drag and drop an **Invoke** activity from the Component area to the design area.
2. Double-click the **Invoke** activity. The **Invoke** dialog is displayed.
3. Enter `Enqueue_Message` in the Name field.
4. Click **Browse Partner Links** at the end of the **Partner Link** field. The **Partner Link Chooser** dialog is displayed.
5. Select `DistListOut`, as shown in Figure 13-41 and click **OK**.

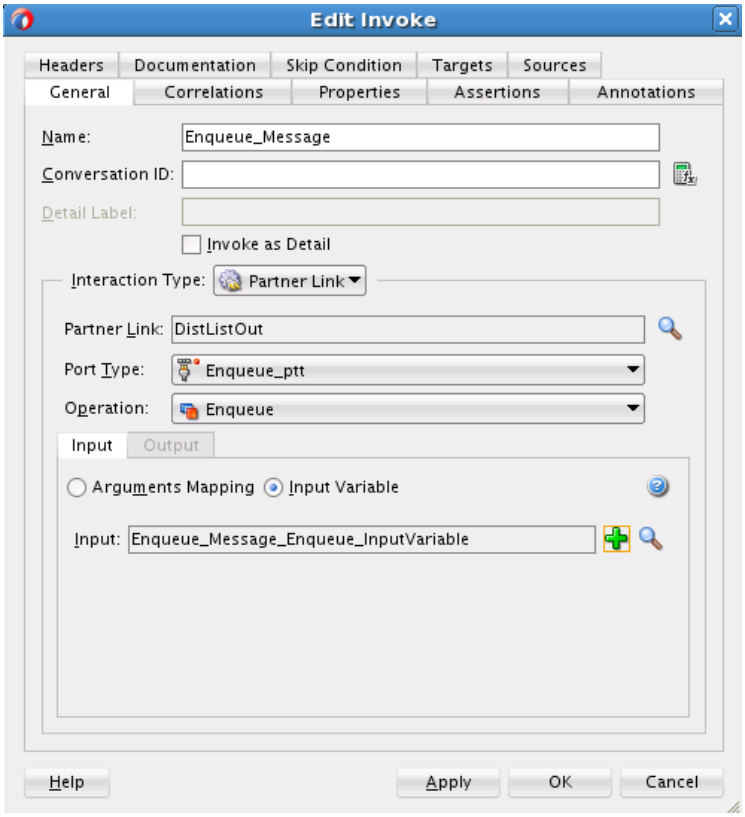
Figure 13-41 The Partner Link Chooser, with DistListOut Selected



6. Select the **Automatically Create Input Variable** icon to the right of the **Input** variable field in the Invoke dialog. The **Create Variable** dialog is displayed.

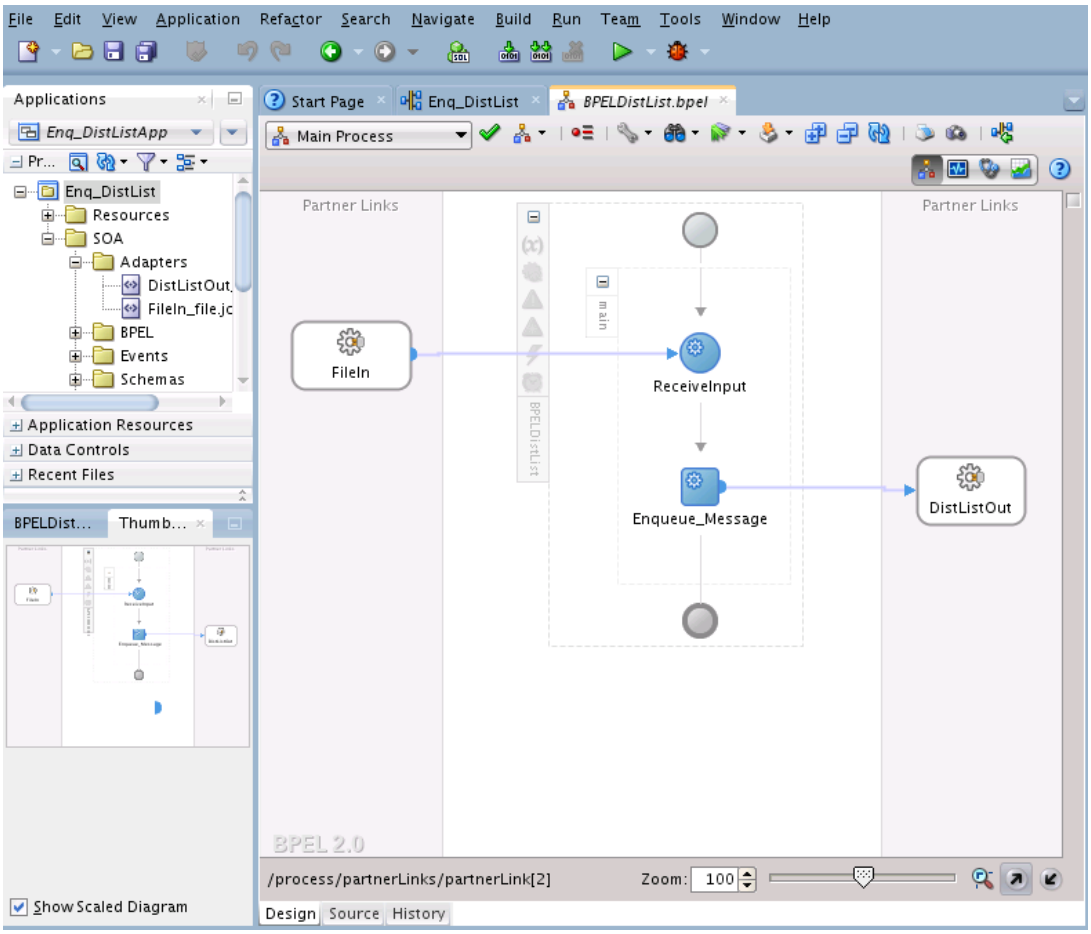
7. Select the default variable name and click **OK**. The **Variable** field is populated with the default variable name. The JDeveloper **Invoke** dialog is displayed, as in Figure 13-42

Figure 13-42 The JDeveloper Invoke Dialog with the Enqueue_Message_Enqueue_inputVariable



8. Click **OK**. The JDeveloper **BPELDistList.bpel** page appears, as shown in Figure 13-45.

Figure 13-43 The BPELDistList.bpel Page After the Invoke Activity is Created

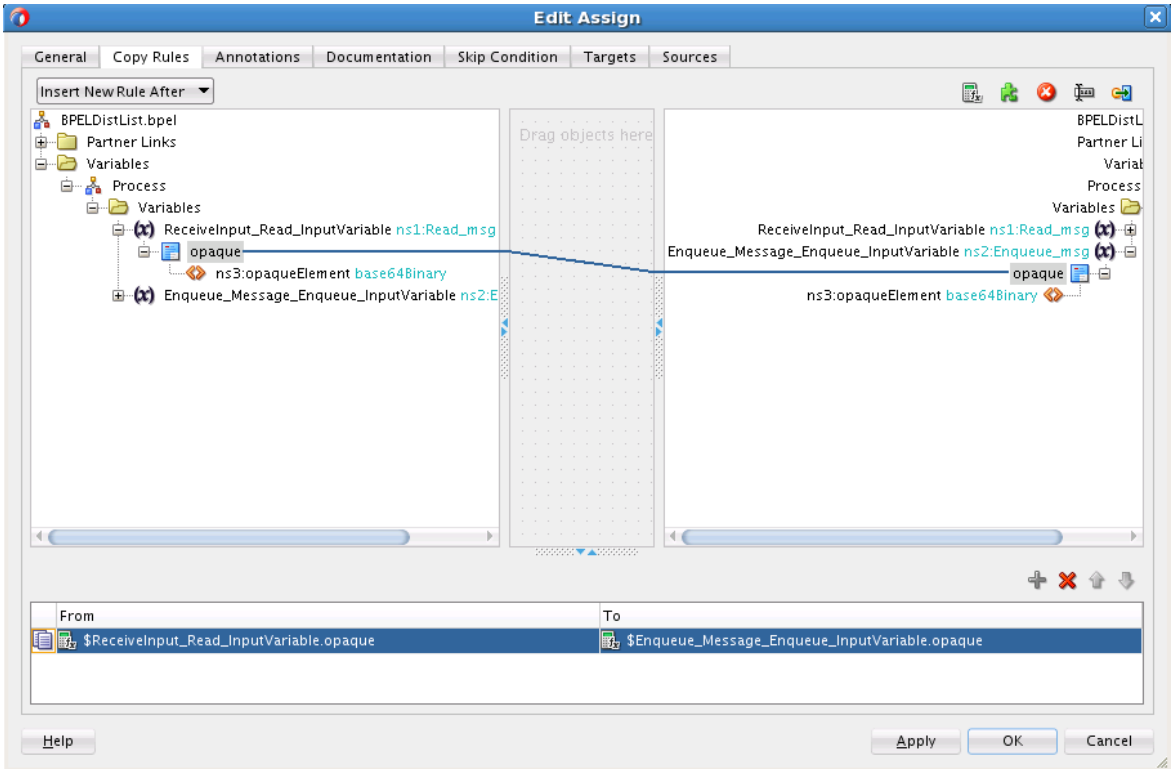


Adding an Assign Activity

Follow these steps to add an Assign activity.

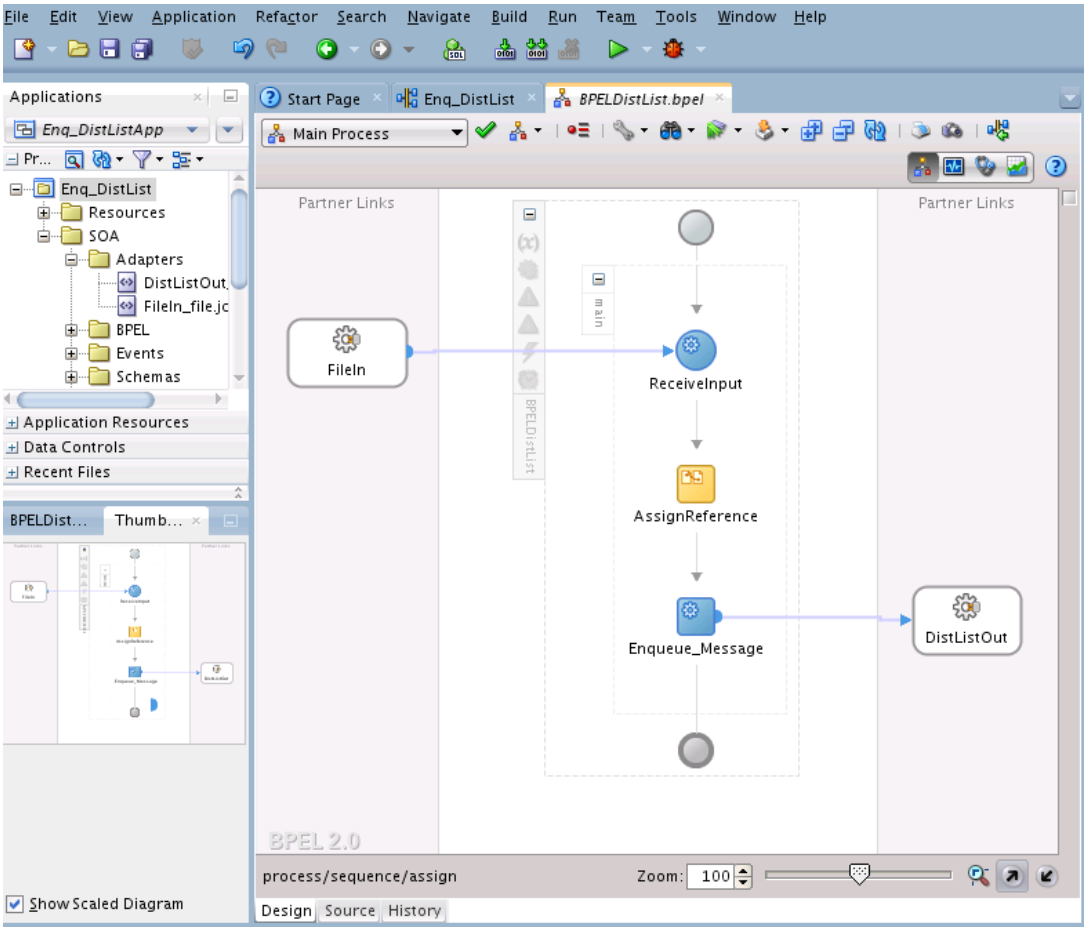
1. Drag and drop an Assign activity from the Component area in between the **Receive** and **Invoke** activities in the JDeveloper design area.
2. Double-click the **Assign** activity. The **Assign** dialog is displayed.
3. Click the **General** tab and enter `AssignReference` in the **Name** field.
4. Click the **Copy Rules** tab.
5. Wire between the opaque element of the variable `ReceiveInput_Dequeue_InputVariable` to the opaque element of the variable `Enqueue_Message_Enqueue_InputVariable` as shown in Figure 12.

Figure 13-44 The Jdeveloper Edit Assign Dialog, Wiring ReceiveInput_Dequeue_InputVariable to Enqueue_Message_Enqueue_InputVariable



6. Click **OK**, the JDeveloper **BPELDistList.bpel** page is displayed, as shown in Figure 13

Figure 13-45 The Completed BPELDistList.bpel Page



7. Click **File**, **Save All**.

14

Oracle JCA Adapter for Coherence

This chapter describes how to use the Oracle JCA Adapter for Coherence. It describes the basic rationale for using the Adapter, concepts and Coherence Adapter Configuration Wizard flow.

This chapter includes the following sections:

Oracle Coherence and Oracle JCA Coherence Adapter Concepts

Coherence provides replicated and distributed (partitioned) data management and caching services on top of a reliable, highly scalable peer-to-peer clustering protocol.

Coherence Cache

A Coherence cache is a collection of data objects that serves as an intermediary between the database and the client applications. Database data can be loaded into a cache and made available to different applications. Thus, Coherence cache reduces load on the database and provides faster access to database data. Objects in the cache can either be of XML or POJO (Plain Old Java Object) type.

Note:

A POJO Java class must be Serialized to use with the Coherence Adapter. That is, a Java class you use with the Coherence Adapter should implement either java.io.Serializable or com.tangosol.io.pof.PortableObject.

For more information on Coherence, see the Oracle Coherence Library. You can also refer to the Oracle Coherence Security Guide for information related to Coherence security.

The Coherence Adapter

The Coherence Adapter is a JCA 1.5-compliant resource adapter for Oracle Coherence. When deployed in a SOA environment, the Coherence Adapter is used as an integration vehicle by SOA composite applications when they integrate with Oracle Coherence.

The Coherence Adapter enables you to perform useful Coherence operations such as adding an item to a Coherence cache, obtaining an item from a Coherence cache, removing an item and querying from a Coherence cache.

Compatibilty

Note that when using the Coherence adapter in 12.1.3, you would not be able to interact with a remote Coherence cluster other than version 12.1.3. or a newer version.

This is due to a Coherence product limitation on the use of the Coherence extend client with older versions of the extend proxy.

Specifically, regarding compatibility between a Coherence server and an extend client, only forward compatibility is maintained from extend clients to cluster proxy servers. That is, an extend client can connect to cluster servers that have either the same or higher version numbers, but not the lower version numbers.

See the link below for the limitation imposed by the Coherence product.

http://docs.oracle.com/cd/E24290_01/coh.371/e22839/gs_install.htm#CBHJGGBG

Oracle Coherence Adapter Features

You can use the Oracle Coherence Adapter to perform the following activities associated with Oracle Coherence:

- Add a Cache Entry: Create a new entry in the Coherence Cache.
- Remove Cache entries: Identify an item to be removed from the Cache, and the system removes the entry from the Cache. You can also remove multiple entries from the cache by providing a filter or search criteria which match the multiple records in cache.
- Get Cache entry value: After specifying an entry to obtain the associated value, the system returns the value of that entry to you.
- Query Cache: After you identify the Cache, and specify search criteria, the system returns the entries that match the search criteria.

Basic Use Cases

There are two basic use cases for the Coherence Adapter.

- A Coherence Adapter connecting to a local cluster. This configuration supports transactional caches.
- A Coherence Adapter connecting to a standalone Coherence cluster or a WebLogic Server. This configuration does not support transactional caches.

  Note that the local Coherence Adapter operations described in this chapter are similar for a remote cache, with limitations as noted. The main difference is that the

name of the cache you specify when you do operations against a remote cache is the name of the remote cache and not that related to a local cluster.

The Coherence Adapter uses the Coherence extend client to connect to the remote cluster. See the next section for a discussion of configuring a remote cache that you can access using the Coherence Adapter.

Configuring the Coherence Adapter Connection to a Remote Cluster

You can use the SOA Coherence Adapter to access remote caches.

The name of the remote cache is captured in a configuration file; in the following example, the file is called `extend-config.xml` file. The location of this configuration file needs to be specified as the value for property `CacheConfigLocation`.In addition to containing the cache name, the configuration file also requires information to connect to a remote Coherence cluster.

Note that the `ManagedConnectionFactory` example here also points to a location for the `PojoJarFile`.

For more information, see the whitepaper on Configuring and Using Coherence Extend.

Also, see Chapter 3 of the documentation, Setting Up Coherence*Extend.

Example - ManagedConnectionFactory for a Remote Cluster

```
<connection-instance>
<jndi-name>eis/Coherence/Remote</jndi-name>
  <connection-properties>
   <properties>
     <property>
       <name>CacheConfigLocation</name>
        <value>/scratch/amahajan/Temp/coherence/dhqa/extend-config.xml</value>
     </property>
   <property>
     <name>ClassLoaderMode</name>
     <value>CUSTOM</value>
   </property>
   <property>
     <name>PojoJarFile</name>
     <value>/scratch/amahajan/Temp/coherence/dhqa/book.jar</value>
 </property>
 <property>
   <name>WLSExtendProxy</name>
   <value>false</value>
</property>
</properties>
</connection-properties>
</connection-instance>
```

Example - Sample extend-config to Define a Remote Cache

In this example, the extend configuration enables the adapter to connect to an extend proxy running at address 10.240.82.123 and listening on port 14777.

```
<?xml version="1.0"?>

<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
   <caching-scheme-mapping>
```

```
      <cache-mapping>
         <cache-name>samples-cache</cache-name>
         <scheme-name>extend-dist</scheme-name>
      </cache-mapping>
      <cache-mapping>
         <cache-name>samples-cache-binxml</cache-name>
         <scheme-name>extend-dist</scheme-name>
      </cache-mapping>
    </caching-scheme-mapping>
    <caching-schemes>
   <remote-cache-scheme>
      <scheme-name>extend-dist</scheme-name>
      <service-name>ExtendTcpCacheService</service-name>
      <initiator-config>
        <tcp-initiator>
          <remote-addresses>
            <socket-address>
              <address>10.240.82.123</address>
              <port>14777</port>
            </socket-address>
          </remote-addresses>
          <connect-timeout>10s</connect-timeout>
        </tcp-initiator>
        <outgoing-message-handler>
          <request-timeout>5s</request-timeout>
        </outgoing-message-handler>
      </initiator-config>
    </remote-cache-scheme>
   </caching-schemes>
</cache-config>
```

Coherence Adapter Connection to Local Cluster

-

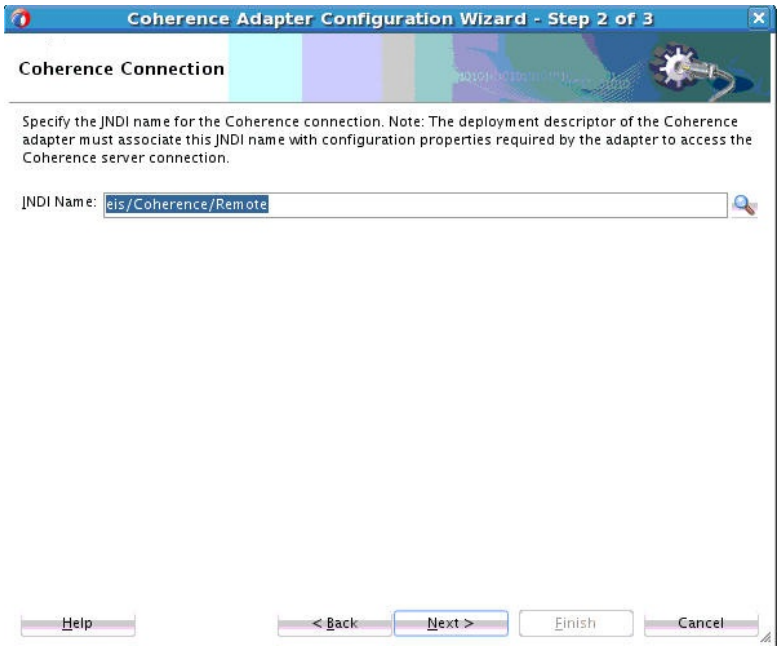
The Coherence Adapter provides a default connection factory to connect to an out-of-box Coherence cache and also a cache called `adapter-local`.

Configuring the Coherence Adapter

To use the Coherence Adapter Configuration Wizard:

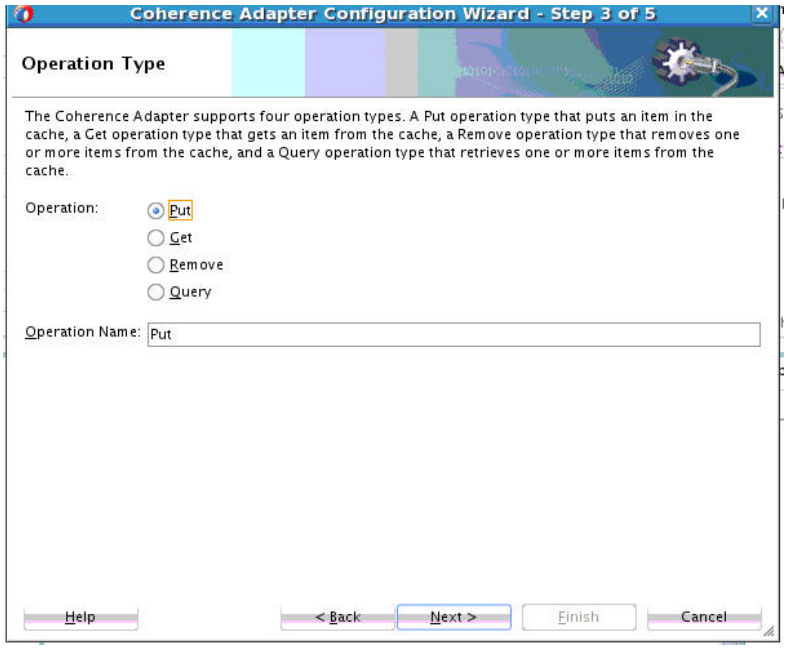
1. In the **Application Navigator** of JDeveloper, select **New Application**. The **Create Generic Application - Name your application** page is displayed.
2. Drag and drop the Coherence Adapter from the Components window of JDeveloper BPEL Designer. The **Coherence Adapter Welcome Screen** appears. Click Next.
3. On the Coherence Connection screen, specify a JNDI name for the Coherence Connection. Select **Next**.

Figure 14-1 The Coherence Adapter Configuration Wizard Service Name Screen



4. On the **Operation Type** Page, specify a valid operation against the Coherence cache that you want to perform. When you import an existing WSDL, the operation name is pre-populated. Operations are **Put (Put item in cache)**, **Get (Get item from cache)**, **Remove (Remove item from cache**) or **Query (Query item from cache)**. In the screen example below, **Put** is selected.

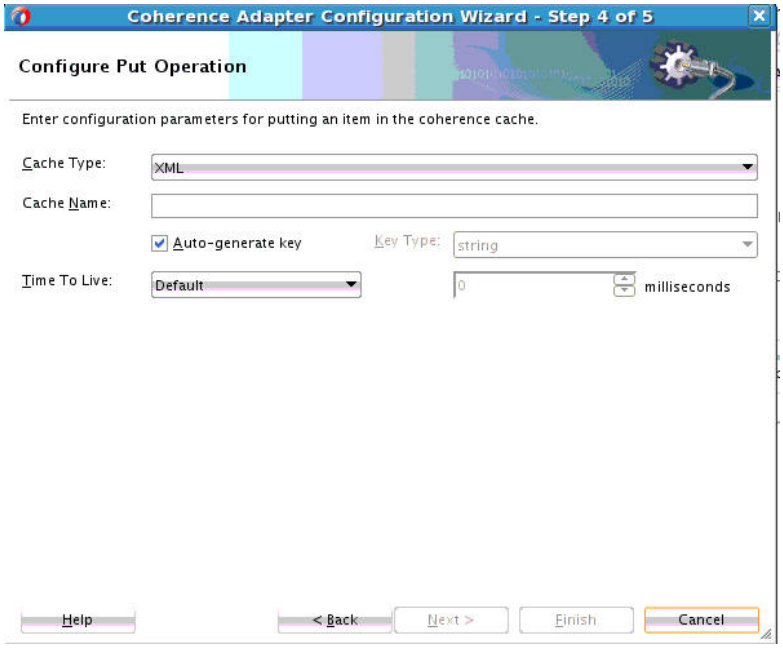
Figure 14-2 Coherence Adapter Configuration Wizard Operation Type Page with Put Selected



5. If you selected Put, the **Put Page** appears. This page captures the configuration parameters for the Put operation. On this screen, you indicate the following fields:

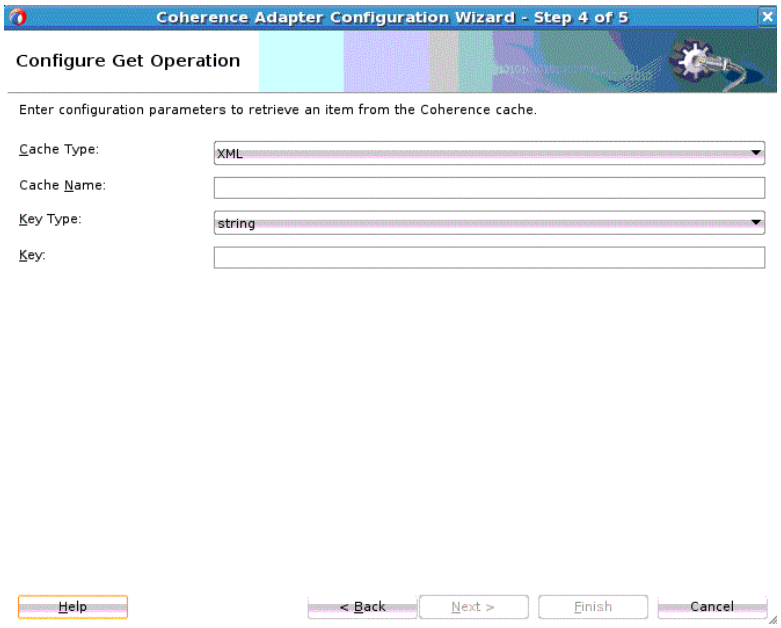
- **Cache Type** – This is a drop-down combo-box with values – XML, POJO. Use the value that corresponds to the type of item you want to put into the cache.
- **Cache Name** – Enter a cache name in this text field. This is the name that uniquely identifies the Coherence cache.
- **Key** – Either enter a key in the text field, or check the auto-generate checkbox to have the key generated. If checked, the key is automatically generated by the Coherence runtime. The key auto-generate process sets the Key Type to `String`.
- **Key Type** – Key Type and Key are disabled, if you choose to enter a filter It is enabled if you choose to enter a Key Type and Key. Here Key Type combo-box with list of Java simple types: string, integer, long, float, double.
- **Auto-generate Key** – Check this box if you want the Coherence Adapter Configuration Wizard to generate a key for you.
- **Time to Live** – Choose Default, Always, or Custom. If you choose Custom, you can specify a value in milliseconds. This value indicates how long an entry should remain in the Coherence cache. The default is that the message never expires. The Time To Live property is applicable more for Remote Caches than for Local caches. For a Local cache, the entry always remains in the cache until you remove it or the SOA server terminates.

Figure 14-3 The Coherence Adapter Configuration Wizard Configure Put Operation Page



6. If you had specified the **Get item from cache** operation on the Coherence Adapter Operation Type page, the `Configure Get Operation` screen appears:

Figure 14-4 The Coherence Adapter Configure Get Operation Page

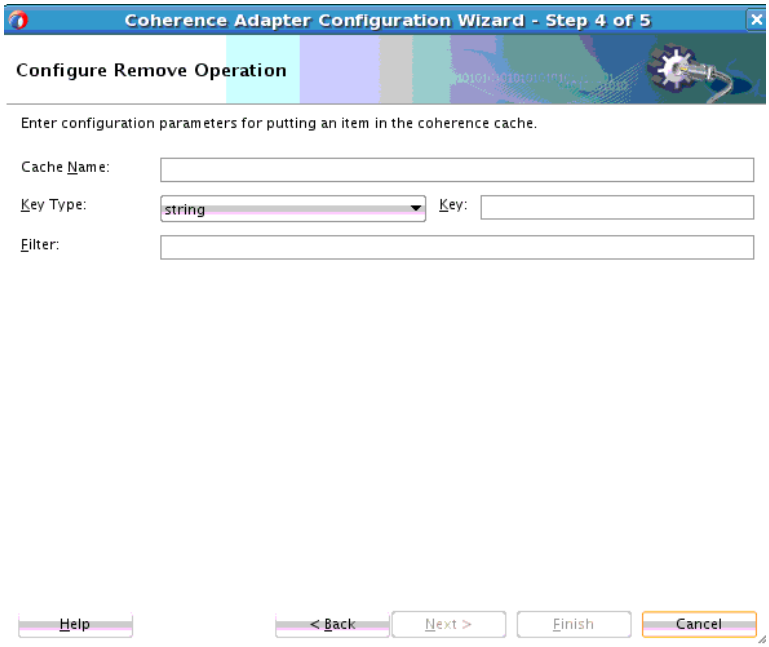


Fill in the following fields to retrieve items from cache:

- **Cache Type** – This is a drop-down combo-box with values – `XML`, `POJO`. Choose the value that corresponds to the type of entry you want to retrieve.
- **Cache Name** – Enter a cache name from where the item is to be retrieved.
- **Key Type** – This field is always enabled for a Get operation.
- **Key** – Enter the key of the cache entry in this text field.

7. If you had chosen the **Remove from cache** option on the Operation Type screen, the Remove Item from Cache screen would appear:

Figure 14-5 The Coherence Adapter Configuration Configure Remove Operation Screen



Enter the configuration parameters to remove an item from Cache, and click Next:

- **Cache Name** –The name of the cache item.
- **Key Type** – This is enabled when performing a Remove operation.
- **Key** – The key for the cache item.
- **Filter** – A string filter for the cache name.You can specify a key or a filter but you cannot specify both.

Note:

When using a filter for a Remove operation, the Coherence Adapter does not report the count of entries affected by the remove operation, regardless of whether the remove operation is successful.

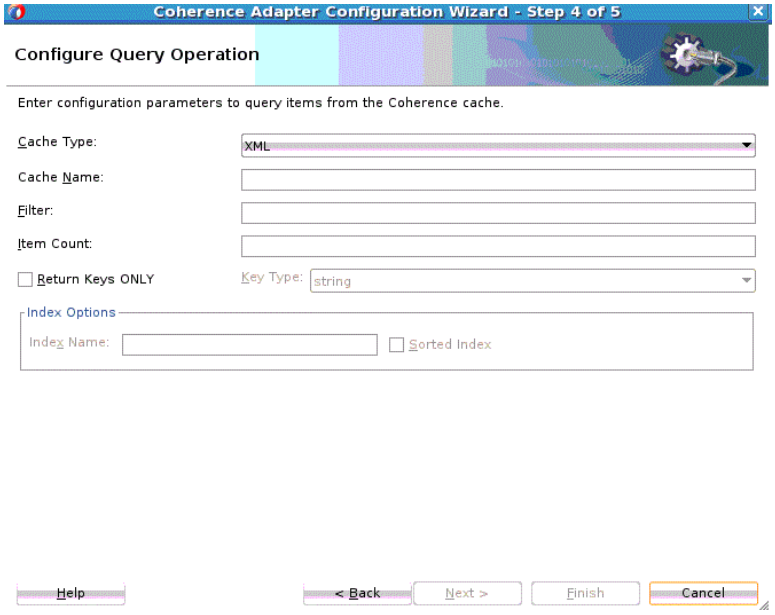
When using a key to remove a specific entry, the Coherence Adapter does report the count, which is always 1 if a Coherence Remove operation is successful.

Querying Items in the Coherence Cache

You can also query items in the Coherence cache.

1. To do, select **Query** from the Operations Screen. The Coherence Adapter Configure Query Operation screen appears.

Figure 14-6 The Coherence Adapter Configure Query Operation Screen



2. On the Configure Query Operation screen fill in the following:

   - **Cache Type** - Select XML or POJO from the dropdown list.
   - **Cache Name** -The name of the cache.
   - **Filter** - You can enter a Coherence Query Language filter expression manually. If you do not specify a Filter, the Coherence Adapter Configuration Wizard warns that all items are to be returned.
   - **Item Count** - An integer to specify the limit to the item count returned from the query.
   - **Index Name** - (Optional) A index name to be created in the cache. Select the Sorted checkbox to create a sorted index.
   - **Return Keys ONLY** - Check the box and supply a return key type. If this box is checked, only the keys will be returned whose values match the entries that are returned from the query.

     Index and Ordered field key types are only valid for the POJO Cache Type. For the XML Cache Type, you can only use key() token in the filter expression.

     This means that when you specify a filter expression you can only use the special token key(), which means the cache object key. For example, you could use either of these keys:

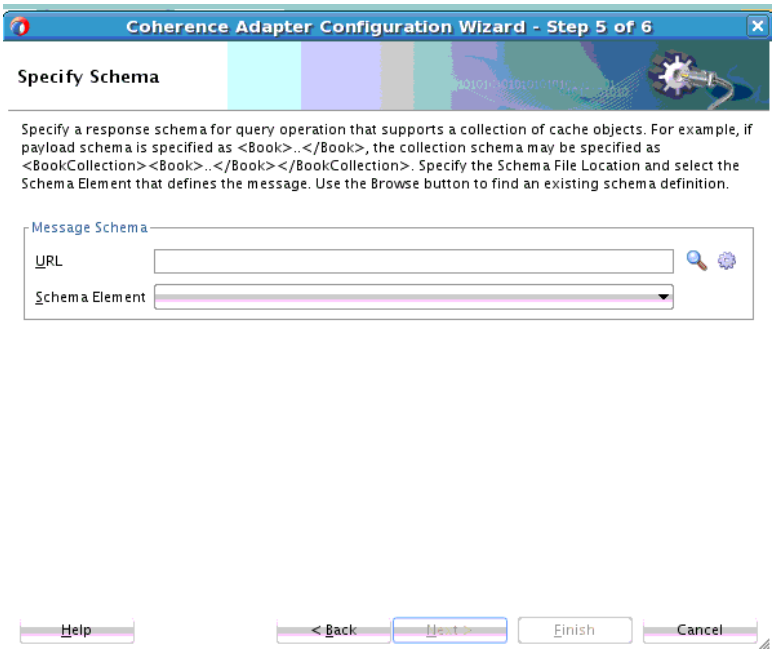
     ```
     key() = 1234
     key() = 5678
     ```

     If you are using the POJO cache type, you can create a filter expression that uses both tokens from the POJO object as well as the special built-in key() token.

Defining Messages for Put, Get and Query Operations if XML is Chosen

If you have chosen XML as the cache type on any of the Put, Get or Query operations, the Specify Schema Page appears. On this page, you select a schema for the Coherence cache object. See Figure 14-7.

Figure 14-7 Coherence Adapter Configuration Wizard Messages (Specify Schema) Screen



1. Specify the Schema File Location in the **URL** field and select the Schema element that defines the elements in the incoming files. Use the Browse button to find an existing schema definition.

2. Some considerations that are applicable to your choices at this point:

   - For a Put operation, a request message is generated that corresponds to schema generated for the value type specified, or to a schema you supplied on the Schema Page. A response message is with the `ReturnIdentifier` of the cache entry created in the Coherence server.

   - For Get operation, the response message points to the schema for the value type, or the schema you supplied on the Schema Page. An empty request message is created.

   - For Query operation, if the filter expression has bind variables, a request message containing elements for these bind variables will be generated. The response message will be the schema generated for the value type, or the schema you supplied on the Schema Page.

Defining Messages for Put, Get and Query Operations if Pojo is Chosen

If you choose POJO as the cache type for a Get, Put, or Query operation, the **Specify Value Type Class Screen** appears.

Figure 14-8 Coherence Adapter Configuration Wizard Value Type Class Screen

To define messages when you have specified POJO as the cache type:

1. Enter the value type for the POJO cache type for the PUT operation in the **Value Type** box. You can use the browser to find a value type.
2. Optionally, enter the metadata mapping file for the schema in the **Metadata Mapping** field.

   The Metadata mapping file helps in POJO to XML and XML to POJO conversion. The mapping file you selected is copied to the JDeveloper project. You can use the browser to find the metadata mapping file. The mapping file is stored in the JDeveloper project. (When you provide XML, the Adapter converts the XML to an object before adding it to the cache. The reverse happens when a you query an object in the cache. The Adapter converts the object stored in cache to XML and passes it along as output to you.)

   If a mapping file is specified, the Coherence Adapter automatically generates the schema.

   If you do not specify a metadata file, the Coherence Adapter Configuration wizard automatically generates the mapping file by introspecting the value type class.

Note:

: For a Query Operation type, if you select Return Keys, the Messages page is not displayed for either XML or POJO Cache Types.

Coherence Adapter Files and Artifacts

This section provides examples of Coherence Adapter design-time artifacts.

JCA File

The JCA file stores JCA property values for each of the supported operations.

Example - JCA File Created

```
<adapter-config name="cohPut1" adapter="Coherence Adapter"
wsdlLocation="../WSDLs/cohPut1.wsdl"            xmlns="http://
platform.integration.oracle/blocks/adapter/fw/metadata">
      <connection-factory location="eis/coherence"/>
       <endpoint-interaction portType="Put_ptt" operation="Put">
    <interaction-spec className=
"oracle.tip.adapter.coherence.CoherenceInteractionSpec">
   <property name="CacheName" value="TestCache"/>
      <property name="Key" value="ABC12345678"/>
      <property name="KeyType" value="String"/>
      <property name="ValueType" value="com.coherence.vt.Book"/>
      <property name="TimeToLive" value="60"/>
      <property name="MappingsMetadataFile" value="book-oxm-mappings.xml"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

WSDL for Put Operation

The following example shows the WSDL for the Coherence Adapter Put Operation

Example - WSDL for Coherence Adapter Put Operation

```
<wsdl:definitions name="cohPut1" targetNamespace=   "http://xmlns.oracle.com/pcbpel/
adapter/coherence/Application1/Project1/cohPut1"
     xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns=http://xmlns.oracle.com/pcbpel/adapter/coherence/
Application1/              Project1/cohPut1
     xmlns:imp1=" http://xmlns.oracle.com/pcbpel/adapter/coherence"
     xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
    <plt:partnerLinkType name="Put_plt">
        <plt:role name="Put_role">
            <plt:portType name="tns:Put_ptt"/>
        </plt:role>
    </plt:partnerLinkType>
 <wsdl:types>
        <schema targetNamespace= “http://xmlns.oracle.com/pcbpel/adapter/coherence/
Application1/Project1/cohPut1"  xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace=" http://xmlns.oracle.com/pcbpel/adapter/coherence "
schemaLocation="xsd/book_cache.xsd"/>
        </schema>
        <schema <schema targetNamespace= “http://xmlns.oracle.com/pcbpel         /
adapter/coherence/Application1/Project1/cohPut1"  xmlns="http://www.w3.org/2001/
XMLSchema">
           <element name="returnId" type="xsd:string"/>
         </schema>
    </wsdl:types>
    <wsdl:message name="Request_msg">
         <wsdl:part name="body" element="imp1:book"/>
    </wsdl:message>
    <wsdl:message name="Response_msg">
        <wsdl:part name="body" element="tns: returnId "/>
    </wsdl:message>
      <wsdl:portType name="Put_ptt">
```

```
        <wsdl:operation name="Put">
            <wsdl:input message="tns:Request_msg"/>
            <wsdl:output message="tns:Response_msg"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

WSDL for Remove with Filter Expression Having Bind Variables

shows the WSDL for the Coherence Remove Operation with a Filter Expression

Example - WSDL for Coherence Remove Operation with a Filter Expression

```
<wsdl:definitions name="cohRem1"
    targetNamespace="http://xmlns.oracle.com/pcbpel/adapter
        /coherence/Application1/Project1/cohRem1"
     xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:tns=http://xmlns.oracle.com/pcbpel/adapter/coherence
/Application1/Project1/cohRem1
     xmlns:plt="http://schemas.xmlsoap.org/ws/
                2003/05/partner-link/">
    <plt:partnerLinkType name="Remove_plt">
        <plt:role name="Remove_role">
            <plt:portType name="tns:Remove_ptt"/>
        </plt:role>
    </plt:partnerLinkType>
    <wsdl:types>
          <schema targetNamespace=
             “http://xmlns.oracle.com/pcbpel/adapter
            /coherence/Application1/Project1/cohRem1"      xmlns="http://www.w3.org/
2001/XMLSchema">
<element name="RemoveRequest">
  <complexType>
     <element name="bind1" type="string"/>
     <element name="bind2" type="string"/>
               </complexType>
              </element>
         </schema>
<schema targetNamespace= “http://xmlns.oracle.com/
          pcbpel/adapter/coherence
       /Application1/Project1/cohRem1"
          xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="ReturnCount" type="integer"/>
        </schema>
    </wsdl:types>
    <wsdl:message name="Request_msg">
        <wsdl:part name="body" element="tns:RemoveRequest "/>
    </wsdl:message>
    <wsdl:message name="Response_msg">
        <wsdl:part name="body" element="tns: ReturnCount "/>
    </wsdl:message>
    <wsdl:portType name="Remove_ptt">
        <wsdl:operation name="Remove">
            <wsdl:input message="tns:Request_msg"/>
            <wsdl:output message="tns:Response_msg"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

WSDL for Get Operation

The following example shows a generated WSDL for the Coherence Get Operation.

Example - WSDL for Get Operation

```
<wsdl:definitions name="cohRem1" targetNamespace=
         "http://xmlns.oracle.com/
     pcbpel/adapter/coherence/Application1/Project1/cohGet1"
     xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns=http://xmlns.oracle.com/pcbpel/
       adapter/coherence/Application1/Project1/cohGet1
     xmlns:imp1=" http://xmlns.oracle.com/pcbpel/              adapter/coherence"
     xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
    <plt:partnerLinkType name="Get_plt">
        <plt:role name="Get_role">
            <plt:portType name="tns:Get_ptt"/>
        </plt:role>
    </plt:partnerLinkType>
    <wsdl:types>
        <schema targetNamespace=            “http://xmlns.oracle.com/pcbpel/adapter
           /coherence/Application1/Project1/cohGet1"
           xmlns="http://www.w3.org/2001/XMLSchema">
          <element name="empty"><complexType/></element>
              </schema>
        <schema targetNamespace=
             “http://xmlns.oracle.com/pcbpel/adapter/
              coherence/Application1/Project1/cohGet1"  xmlns="http://www.w3.org/
2001/XMLSchema">
<import namespace=
      "http://xmlns.oracle.com/pcbpel/adapter/coherence
           "schemaLocation="xsd/book_cache.xsd"/>
            </schema>
    </wsdl:types>
    <wsdl:message name="Request_msg">
        <wsdl:part name="body" element="tns:empty "/>
    </wsdl:message>
    <wsdl:message name="Response_msg">
        <wsdl:part name="body" element="imp1:book "/>
   </wsdl:message>
    <wsdl:portType name="Get_ptt">
        <wsdl:operation name="Get">
            <wsdl:input message="tns:Request_msg"/>
            <wsdl:output message="tns:Response_msg"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

WSDL for Query with Filter Expression having Bind Variables

The following example shows the WSDL for the Query Operation, with a Filter Expression including Bind variables.

Example - WSDL for Query with Filter Expression Having Bind Variables

```
<wsdl:definitions name="cohQuery1"         targetNamespace="http://xmlns.oracle.com/
pcbpel/adapter/coherence    /Application1/Project1/cohQuery1"
     xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns=http://xmlns.oracle.com/pcbpel/adapter            /coherence/
```

```
Application1/Project1/cohQuery1
     xmlns:imp1=" http://xmlns.oracle.com/pcbpel/
                  adapter/coherence"
     xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
    <plt:partnerLinkType name="Query_plt">
       <plt:role name="Query_role">
        <plt:portType name="tns:Query_ptt"/>
        </plt:role>
    </plt:partnerLinkType>
    <wsdl:types>
<schema targetNamespace= “http://xmlns.oracle.com/
             pcbpel/adapter/coherence
            /Application1/Project1/cohQuery1"                    xmlns="http://
www.w3.org/2001/XMLSchema">
<element name="QueryRequest">
  <complexType>
     <element name="bind1" type="string"/>
     <element name="bind2" type="string"/>
              </complexType>
             </element>
         </schema>
<schema targetNamespace= “http://xmlns.oracle.com/
        pcbpel/adapter/coherence
       /Application1/Project1/cohQuery1"  xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace=" http://xmlns.oracle.com/
            pcbpel/adapter/coherence"
              schemaLocation="xsd/book_cache.xsd"/>
            </schema>
    </wsdl:types>
        <wsdl:message name="Request_msg">
        <wsdl:part name="body" element="tns:QueryRequest "/>
        </wsdl:message>
        <wsdl:message name="Response_msg">
        <wsdl:part name="body" element="imp1:book "/>
        </wsdl:message>
    <wsdl:portType name="Query_ptt">
        <wsdl:operation name="Query">
            <wsdl:input message="tns:Request_msg"/>
            <wsdl:output message="tns:Response_msg"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

Tips for Using the Coherence Adapter

If you see the following error:

```
Execute of operation 'Put' failed due to: Service "TransactionalCache" has been
started by a different configurable cache factory.; nested exception is:
java.lang.IllegalStateException: Service "TransactionalCache" has been started by a
different configurable cache factory.
```

This message means that the Service `TransactionalCache` is defined in multiple places. Obtaining this error is not limited only to the Coherence Adapter Put operation. You can get this exception for all operations supported by Coherence Adapter.

This issue occurs when you have the configuration files as below.

The first configuration file, `a-config.xml` contains the following:

```
<cache-config>
 <caching-scheme-mapping>
  <cache-mapping>
   <cache-name>adapter-local</cache-name>
   <scheme-name>transactional</scheme-name>
  </cache-mapping>
 </caching-scheme-mapping>

 <caching-schemes>
  <transactional-scheme>
   <scheme-name>transactional</scheme-name>
   <service-name>TransactionalCache</service-name>
   <autostart>true</autostart>
  </transactional-scheme>
 </caching-schemes>
</cache-config>
```

The second configuration file, `b-config.xml`, contains the following:

```
<cache-config>
 <caching-scheme-mapping>
  <cache-mapping>
   <cache-name>movie-local</cache-name>
   <scheme-name>transactional</scheme-name>
  </cache-mapping>
 </caching-scheme-mapping>

 <caching-schemes>
  <transactional-scheme>
   <scheme-name>transactional</scheme-name>
   <service-name>TransactionalCache</service-name>
   <autostart>true</autostart>
  </transactional-scheme>
 </caching-schemes>
</cache-config>
```

You use `a-config.xml` for one jndi and `b-config.xml` in another jndi and two composites use these two different jndis. When you deploy the first configuration, it completes acceptably, but the second deployment fails because the service `TransactionalCache` has already been started by the first one.

The solution is to have only one `config.xml` file and use the same `config.xml` across different jndis. For example, you can use the following `config.xml file` across different jndis.

```
<cache-config>
 <caching-scheme-mapping>
  <cache-mapping>
   <cache-name>adapter-local</cache-name>
   <scheme-name>transactional</scheme-name>
  </cache-mapping>
  <cache-mapping>
   <cache-name>movie-local</cache-name>
   <scheme-name>transactional</scheme-name>
  </cache-mapping>
 </caching-scheme-mapping>
 <caching-schemes>
  <transactional-scheme>
   <scheme-name>transactional</scheme-name>
   <service-name>TransactionalCache</service-name>
   <autostart>true</autostart>
```

```
  </transactional-scheme>
 </caching-schemes>
</cache-config>
```

15

Oracle JCA Adapter for JDE Edwards World

This chapter shows how to use the Oracle JCA Adapter for JDE Edwards World, and provides both design time and runtime configuration information, along with a brief conceptual overview.The JDE World System is an ERP (Enterprise Resource Planning) product running on IBM minicomputers. The JD Edwards World Adapter uses a new JDBC-driver based connection to the IBM system and enables SOA integration users to interact with JDE World system as a typical database.

For complete information on the JD Edwards World System, see the Oracle JD Edwards World documentation page.

This chapter includes the following sections:

JD Edwards World System and JDE Edwards World Adapter Concepts

J.D. Edwards World System enables integration with Enterprise systems through its interoperability framework. The Oracle JD Edwards World Adapter uses the Adapter framework and leverages various integration access methods to provide the greatest amount of flexibility and functionality.

JDE Edwards World System is a database-based integration: the J.D.Edwards application is intrinsically a database. Its reports and any information extraction from it are based on SQL queries. Similarly, write interactions with JDE Edwards World System are based on inserts into special tables. The special tables are called z-tables and are used to temporarily store data that you have imported from an external source (a flat file, spreadsheet, or another type of file). The JD Edwards World System processes this data and imports it into real database tables. The Z-tables help to ensure that the format and process of importing data are as consistent as possible.

Z-files contain all the information needed to process a transaction in the JD Edwards System and there is an associated batch processing program for each Z-file.

Z-files correspond to different types of transactions, such as sales orders and addresses. The JD Edwards World System documentation documents these different types of files and their use within the JD Edwards World System.

The JD Edwards World Adapter exposes the underlying JD Edwards World System database tables through the filter of an integration by doing the following:

- Exposing only integration points, that is, alternative descriptive names of tables/columns. The JD Edwards World Adapter does not expose the "real" names of the JD Edwards World tables.
- Providing a powerful UI that enables you to find the correct integration touch points
- Providing a JDBC driver which ensures that only certain trusted callers are allowed to insert data into a limited set of tables.

The information in the JD Edwards World system is in tables in a format unusable to a Database Adapter. Numbers are stored without the decimal place, dates are in a certain format, and connections must be made with "translate binary=true" (AS/400 DB2 specific).

For the purposes of the JD Edwards World Adapter, these formats are handled by the JD Edwards World JDBC driver, which also handles most of the typical runtime Adapter functions such as environment setup and data translation. Thus, the JDBC driver provides a means of communication between the JD Edwards World Adapter and the JD Edwards World system Z-tables.

JD Edwards World Adapter Features

The JD Edwards World Adapter Features includes these features

- **JDE World Create Connection Template**. Through the Configuration Wizard, the Adapter provides the use of connection templates to assist in connecting to the JD Edwards World System.
- **JDE World Custom Connection Parameters**. Through the Configuration Wizard, the JD Edwards World Adapter provides the use of connection parameters specific to JDE World System.
- **Operation determined by table selected**. The JD Edwards World Adapter enables the user to choose an operation that is determined by the table that is selected.
- **Select Table Browser**. This functionality enables you to browse JD Edwards World Z-tables, as an aide and prerequisite to your using the JD Edwards Adapter to perform a Select against the Z-tables.
- **JD Edwards World System Z-tables grouped by business module**. To assist you in determining which tables you want to perform inserts and selects against, the JD Edwards World Adapter provides JD Edwards World System business module groupings of Z-tables. These business modules have descriptive names that are clearly indicative of their use in the Enterprise.
- **The JD Edwards World Adapter exposes only those Z-tables that are to be used as integrations endpoints**. It does not expose Z-tables that are not integration endpoints.
- **The JD Edwards World Adapter provides connection pooling support**.
- **Z-table Insert Support**. Inserts to JD Edwards World System Z-tables require that the SQL connections obtained by the JDE World JCA adapter have permission to execute insert statements.

  Again, the Z-tables against which the JD World System Adapter provides you the ability to perform operations are staging tables: that is, they are data interfaces to import/export data to and from J D Edwards. External Applications always

interface with the Z-tables from where you can move data into real JDEdwards World System tables using batch programming and other interfaces on the back-end JD Edwards World System.Different transactions in different functional areas have their own defined Z-tables and configurations to move them into real JD Edwards World tables. Refer to each of the vertical (Financial, Manufacturing, Human Resources) sets of JD Edwards World documentation, where you can find information about processing options that enable you to move data from their respective Z-tables.

- **No Polling Support**. There is no polling support for the JD Edwards World Adapter.
- **No Transaction Support**. Though many adapters generally support XA transactions, the JDEdwards World Adapter does not. This is partly because JDE World systems are non-transactional. Journalling is typically not enabled on the AS/400 DB2 database underlying the JD Edwards World System, meaning that not only are XA transactions not possible, but starting and committing a local transaction is also not supported.

  Also, there are only two database options available within the JD Edwards World Adapter: select and insert. Select does not need to be transactional. Insert is also to a Z-table only, and is always associated with a non-transactional RPG call.

Configuring the JD Edwards World Adapter

There are several preparatory configuration steps involved with the JD Edwards World Adapter. Prior to deploying the Adapter, you must provide both runtime and designtime configuration.

Configuring Connection Pooling for the JDEdwards World Adapter

A JCA connector must provide pooling support. With the Database Adapter, SQL connections are obtained from a Data source. In this way, pooling requirements are delegated to the underlying JDBC driver and its DataSource implementation.

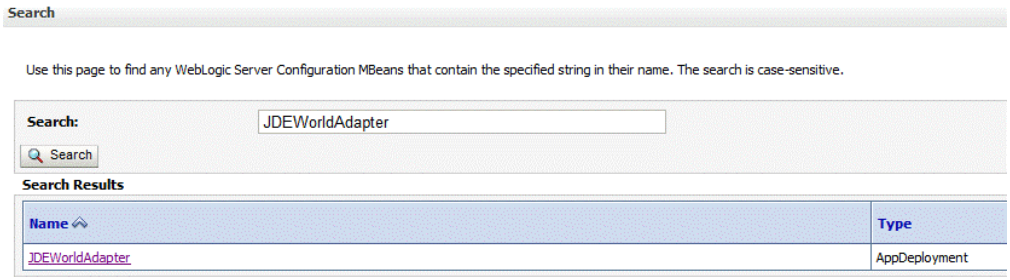
Because the JD Edwards World JCA Adapter is based on the Oracle Database Adapter, the process for configuring pooling support for the JD Edwards World Adapter is similar to that of providing pooling support for the Oracle Database Adapter.

To provide connection pooling for the JD Edwards World Adapter:

1. Ensure that the Oracle WebLogic Server is installed on a build which contains the JD Edwards World Adapter. Next, search for and find the JD Edwards World Adapter on the **Search Panel** on the WebLogic Console.

   Enter the name of the Adapter in the **Search** field. The **Search Panel** displays the search results, JDEdwards World Adapter, with its type AppDeployment, See Figure 15-1.

Figure 15-1 Searching for the JDEdwards World Adapter on the WebLogic Console

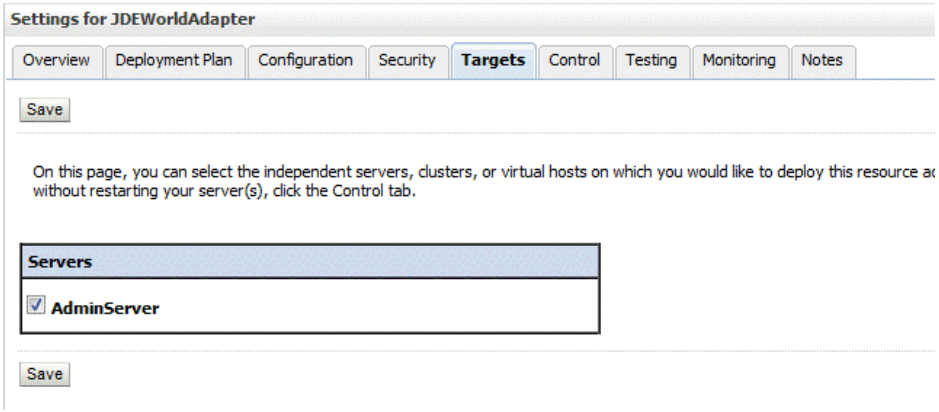


2. The JDEdwards World Adapter is an optional Adapter, thus its initial JDeveloper state is Installed, not Active. You must make the JDEdwards World Adapter state active. To make the state Active:

   a. Select the **Targets** tab on the WebLogic Console

   b. Select the **Servers** to which to deploy the Adapter. Click **Save**.

   Figure 15-2 Selecting a Server on the WebLogic Console

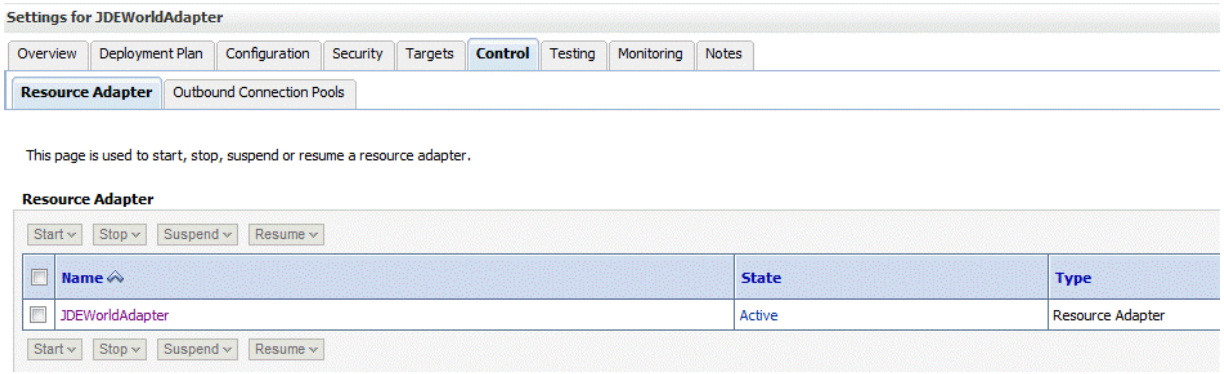


3. You must obtain the `jt400.jar` file from Sourceforge (the `jt400.jar` file is called JTOpen 7.9 at that location.) JTOpen is the open source version of the IBM Toolbox for Java Licensed Program Product (LPP).

   Place the `jt400.jar` file under `<middleware_home>/user_projects/domains/WLS_SOA/lib/`

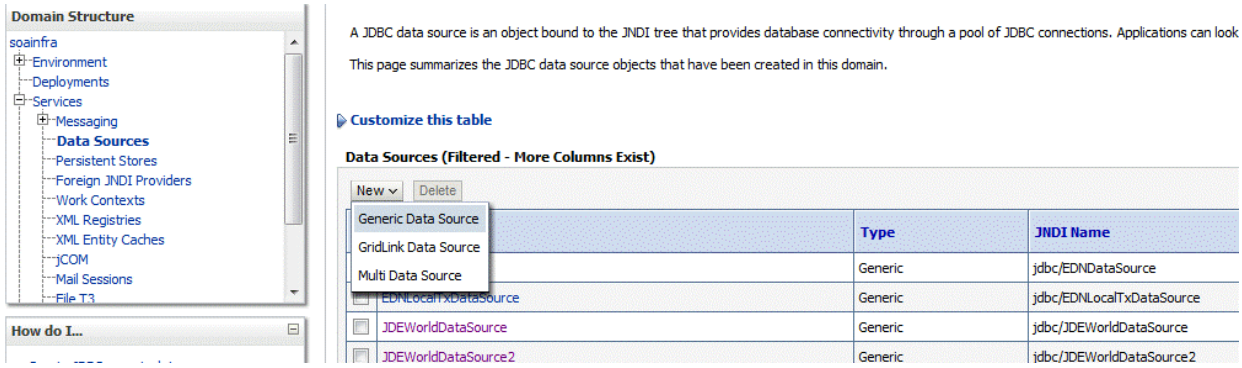
4. To verify that the Adapter is active, select the **Control** tab. You might need to manually activate it. To do so, select the checkbox next to `JDBCWorldAdapter` and choose **Start**.

Figure 15-3 Verifying the JD Edwards World Adapter is in Active Status



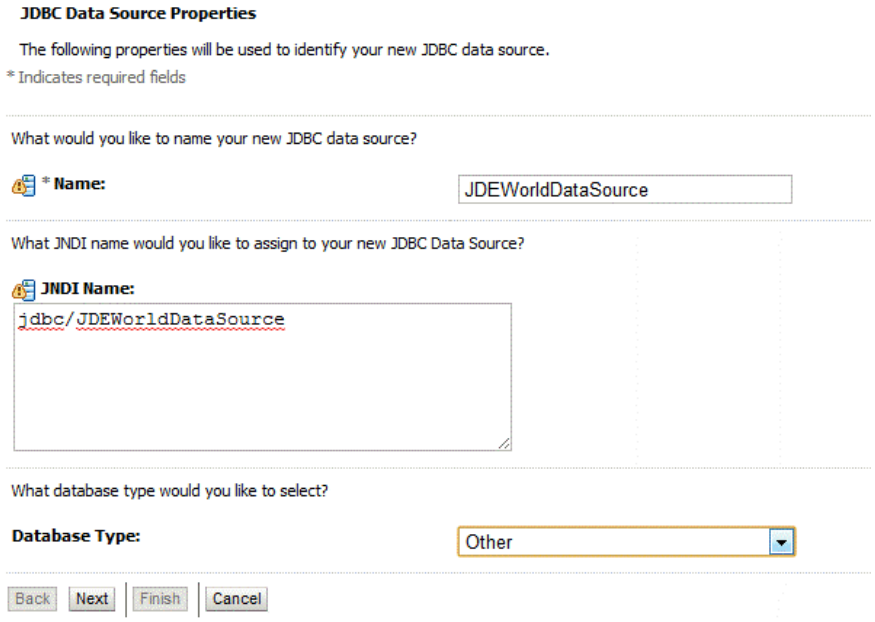
5. The next step is to create a data source connection by proceeding to **Data Sources -->Services** on the WebLogic Console and selecting **Generic Data Source**:

Figure 15-4 Choosing a Generic Data Source



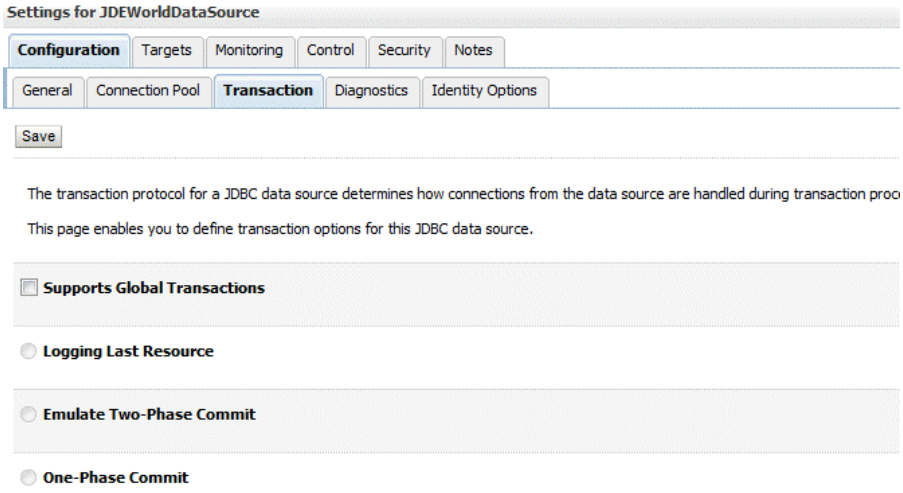
6. Supply the name `jdbc/JDEWorldDataSource`. This represents the term to which the default JCA `eis/JDEWorld/JDEWorldDemo` instance is referring. Use database type `Other`.

Figure 15-5 Defining the JDBC Data Source JDEWorldDataSource



7. Within the **Transactions** tab, disable XA by ensuring that **Supports Global Transactions** is not selected. See Figure 15-6.

Figure 15-6 Ensuring Supports Global Transactions is Not Selected on the WebLogic Transactions Tab



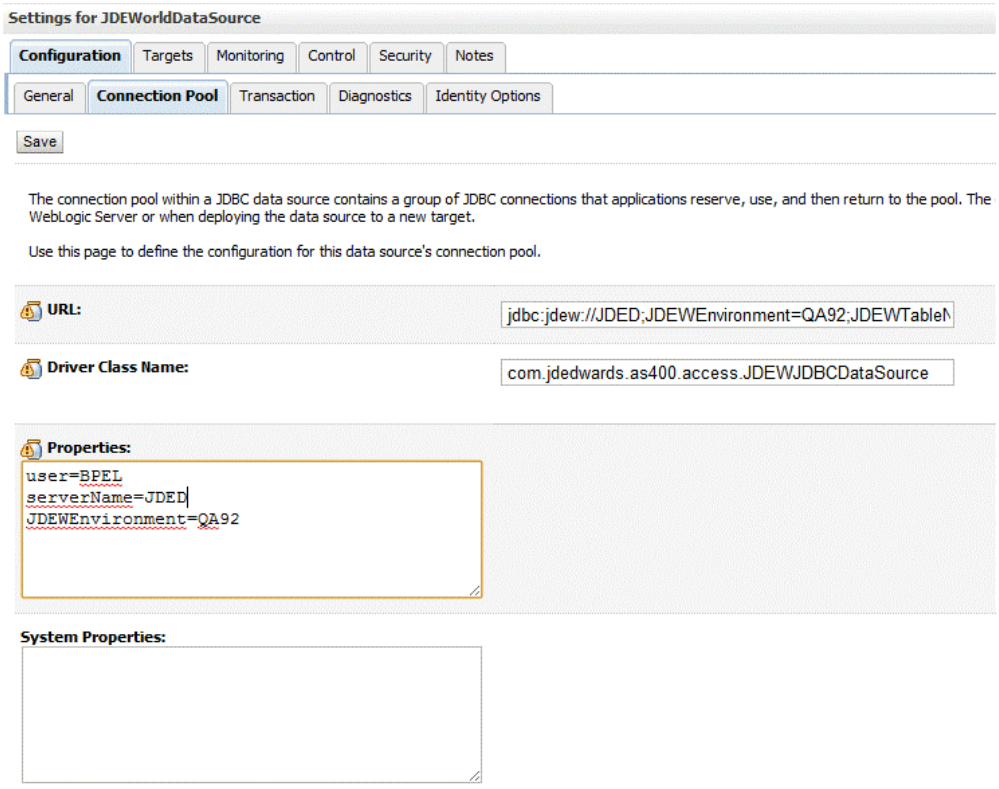
8. Proceed to the **Connection Pool** page to configure the Connection Pool information. See Figure 15-7.

   The Connection Pool URL should resemble the following. Be sure to also specify three settings (**User**, **ServerName**, and **JDEWEnvironment**) as properties on the Connection Pool page, even though all three appear either in the connection URL already or were entered when you configured the data source:

```
jdbc:jdew://
JDED;JDEWEnvironment=QA92;JDEWTableNomenclature=OBJN_OBJT;JDEWColumnNomenclature=FDFT_FDFN;transla
te binary=true;date format=mdy;prompt=false;auto commit=true;
```

```
user=BPELTEST
serverName=JDED2.us.mydomain.com
JDEWEnvironment=A93BPELTEST
JDEWTableNomenclature=OBJN_OBJT
JDEWColumnNomenclature=FDFT_FDFN
```

Figure 15-7 Configuring the Connection Pool URL on the WebLogic Console

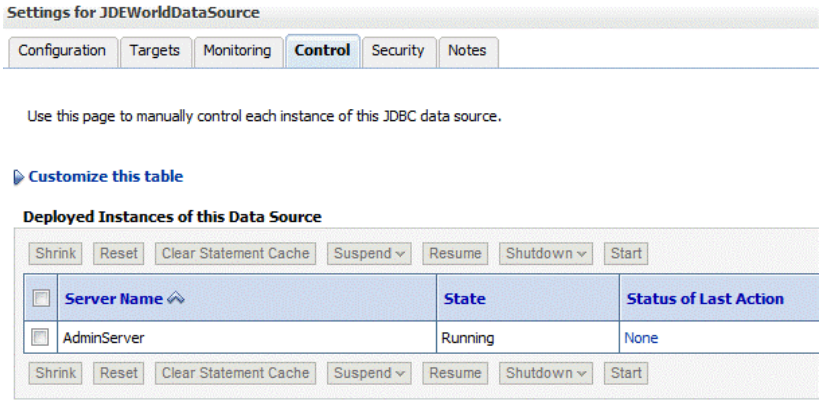


9. Select **Advanced** and set **Connection Reserved Timeout** to 180. This value is required because creating a connection can take three minutes the first time the Adapter connects using the Connection Pool.

Figure 15-8 Setting the Connection Reserved Timeout Value

10. Next, select the **Control** tab to manually control each instance of the JDBC Data Source.

Figure 15-9 Using the Control Tab to Manually Manage Each Instance of the Deployed Data Source

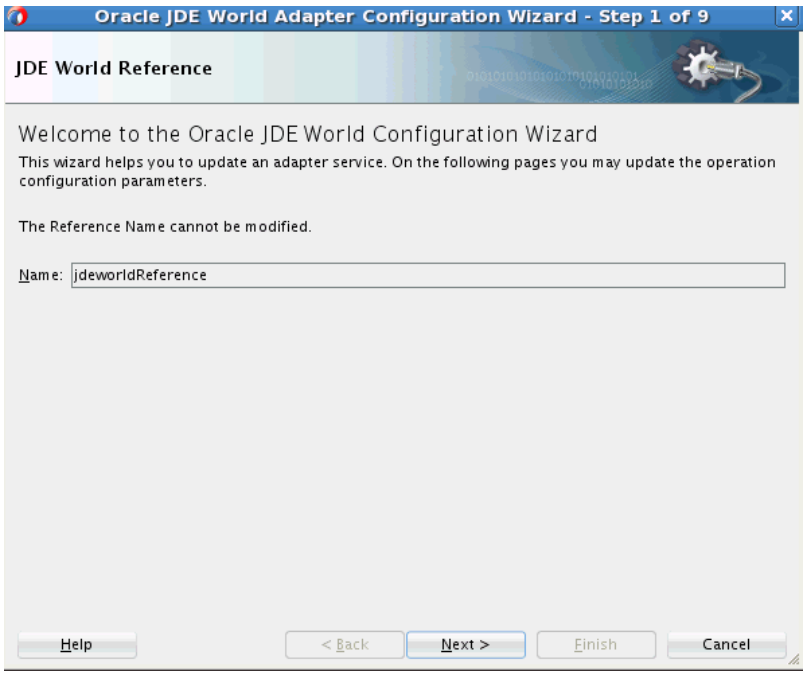


JD Edwards Word Adapter Configuration Wizard Flow: Insert Operation

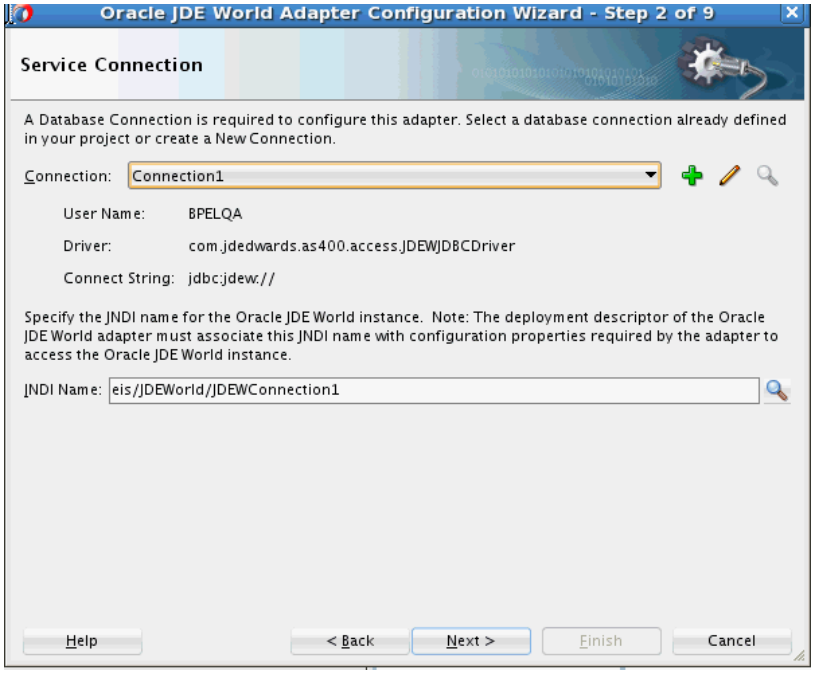
You can use the JD Edwards Adapter Configuration Wizard to configure the JD Edwards Adapter to perform an insert operation to the JDEdwards World Z-tables.

To do so:

1. Launch the JDE World Adapter Configuration Wizard by dragging and dropping it onto a composite or BPEL process in the Reference section of the JDeveloper design view. The JD Edwards World Adapter Configuration Wizard Welcome Screen appears.

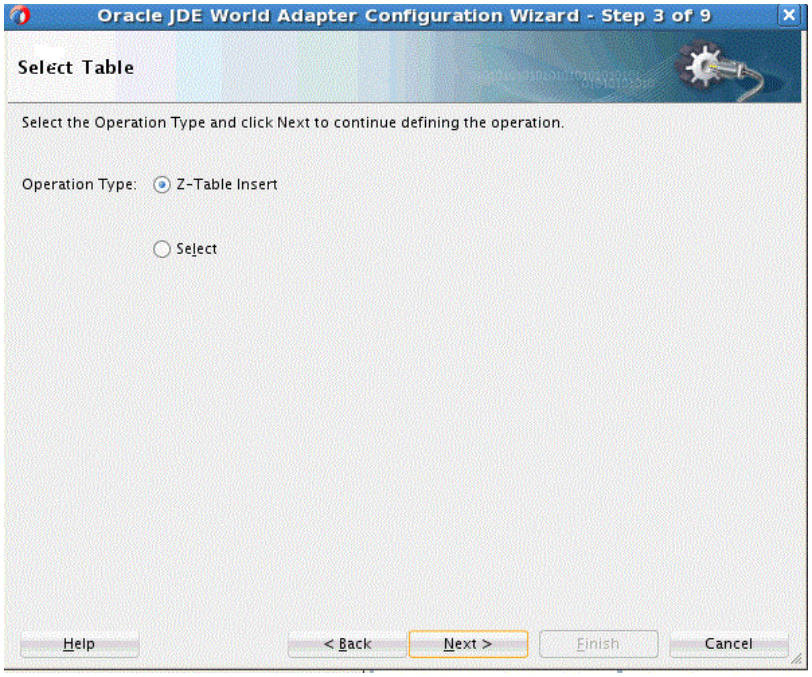


2. Use the **Service Connection** screen to establish a service connection. Edit an existing connection by browsing to that connection or supply new connection information on this page. Click **Next**.



3. The **JDEWorld Adapter Configuration Wizard Select Table** screen is displayed. This screen enables you to select either one of two operations you can perform against JD Edwards World z-tables: **Z-table Insert** or **Select**. In this example, **Z-table Insert** has been selected. Click **Next**.

Figure 15-10 JDE World Adapter Configuration Wizard Select Table Screen



4. The **Service Connection** page opens, enabling you to select the root database for this service's query.

   To do this, you must have a basic understanding of the Z-Tables against which you want to perform the operation and the root database table. It also helps to

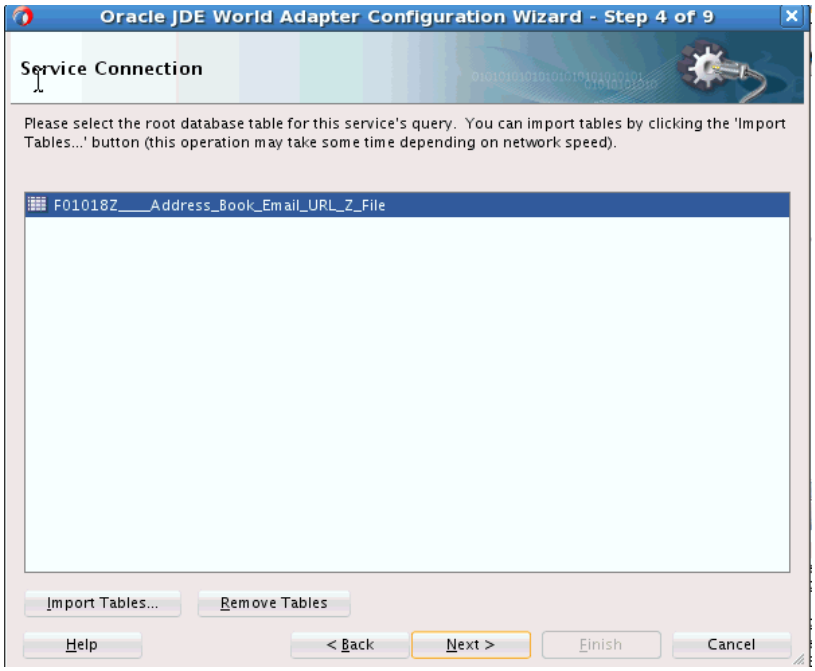
understand the JD Edwards World System business module against which you are performing insertions or making queries.

This page enables you to import Z-tables by clicking the **Import Tables** button at the bottom of the page. When you select a Z-Table from this list, you are selecting a table that is the root database for the operation.You can also remove Z-tables you have added.

To do so:

- Select the **Import Tables** button to import Z-tables, and select the table you want as the root database table, or
- After importing, you can remove tables. To remove a table from the list, select the appropriate tables, then select the **Remove Tables** button to remove Z-Tables from the list of ones you want imported.
- Select **Next**.

Figure 15-11 Selecting the Root Database Table



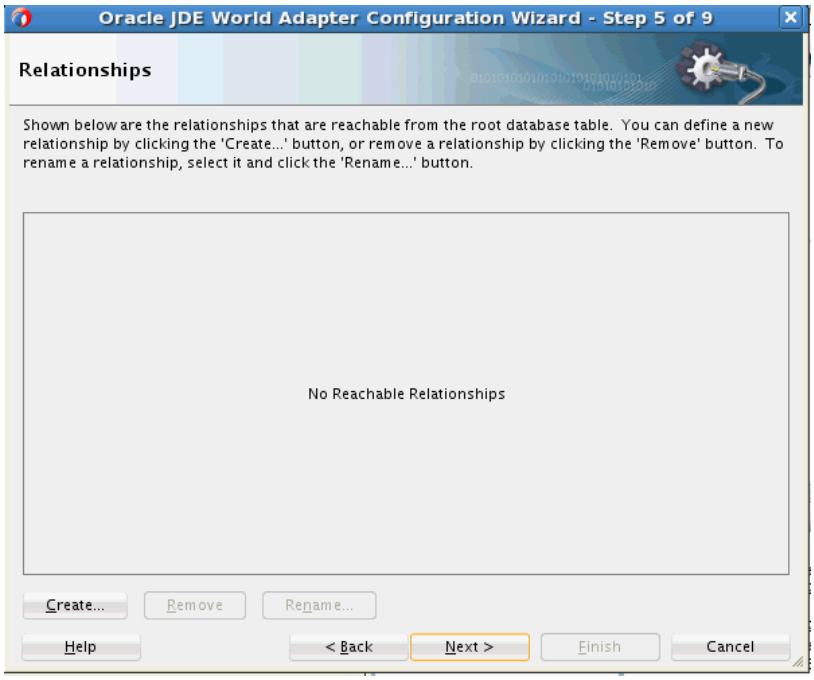
5. The JD Edwards World Adapter Configuration Wizard displays the **Relationships** screen.This screen provides a list of relationships that are available from the root database.

   You can define new relationships by clicking the **Create** button, which is shown at the bottom of the screen and enables you to provide a table that can be reachable from the root database. Next to the **Create** button is the **Remove** button, which enables you to remove a table from being reachable from the root database table.

   You can also rename a relationship table by clicking it in rename it. You can also just move to the next screen.

   After you have optionally completed creation or removal, or not performed either of these, as in the example, select **Next**.

Figure 15-12 The JDEWorld Adapter Configuration Wizard Relationships Screen



6. The JDEWorld Adapter Configuration Wizard **Attribute Filtering Screen** appears. This shows the attribute filter that is created from the imported table definitions, including any relationships that you may have defined.

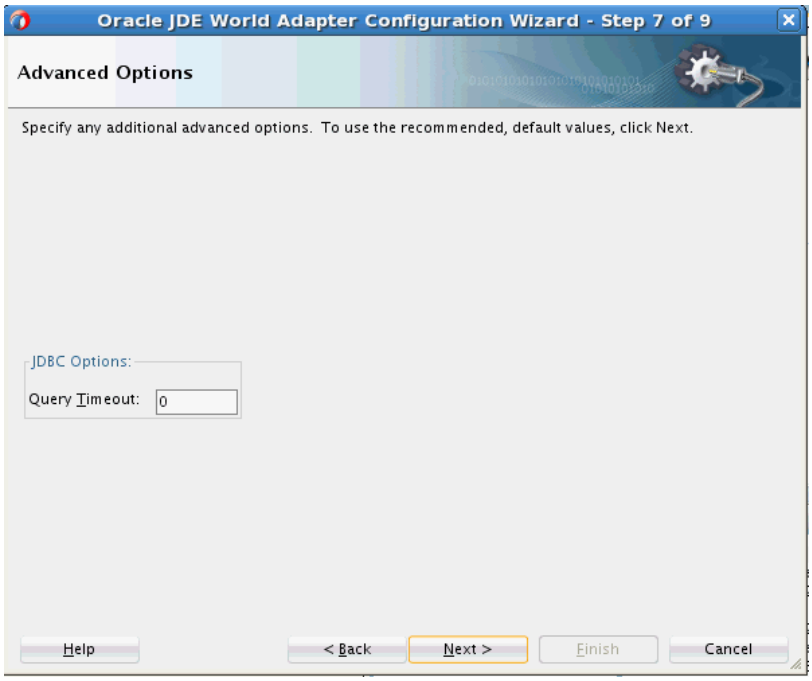
   You can also uncheck any attributes you want to exclude from the database queries you perform. Note that you cannot exclude primary key attributes. Select **Select All** to select all attributes or **Deselect All** to deselect all attributes.

   Select **Next**.

Figure 15-13 The JDEWorld Adapter Configuration Wizard Attribute Filtering Screen

7. The JDE World Adapter Configuration Wizard **Advanced Options Screen** appears. On this page, the **JDBC Options Query Timeout** option is provided. You can either change this value from its default, 0, or select **Next** to accept the default value.

Figure 15-14 The JDE World Adapter Configuration Wizard Advanced Options Screen

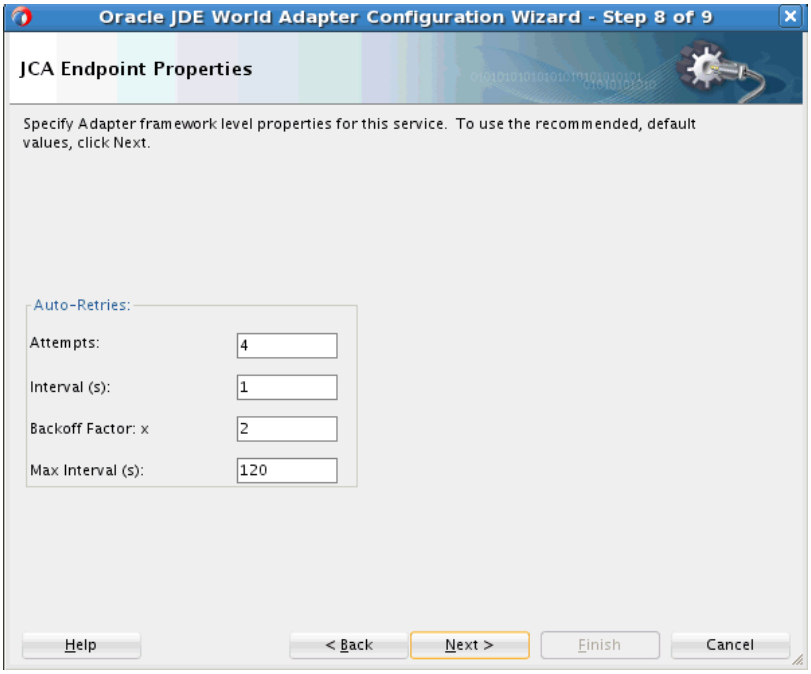


8. The JDE World Adapter Configuration Wizard JCA Endpoint Properties Screen appears.

In the **Auto-Retries** section, specify the value for auto-retry in case of an operation time out. This value is used if there is a connection-related fault, and the Invoke activity can be automatically retried a limited number of times. You can specify the following values in the fields in this section:

- To retry indefinitely, type `unlimited` in the **Attempts** field.
- **Interval** is the delay between retries.
- **Backoff Factor: x** enables you to wait for increasing periods of time between retries. 9 attempts with a starting interval of 1 and a back off of 2 leads to retries after 1, 2, 4, 8, 16, 32, 64, 128, and 256 (2^8) seconds.
- **Max Interval** is the number of intervals of delays between retries.

Figure 15-15 JDE World Adapter Configuration Wizard JCA Endpoint Properties



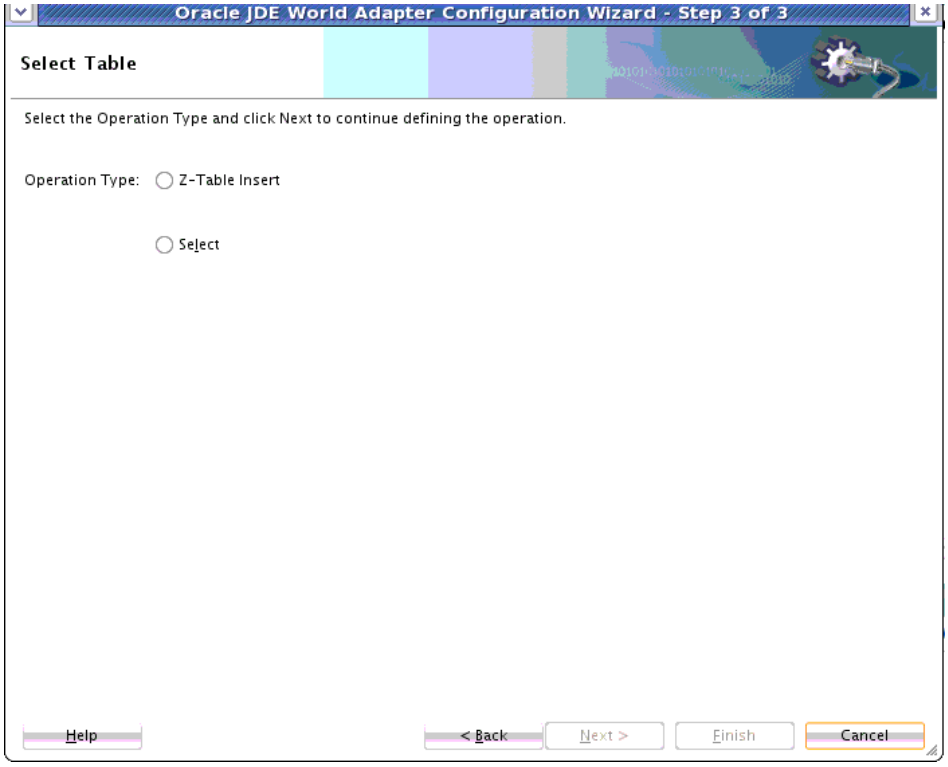
JD Edwards World Adapter: Select Operation

The JDEWorld Adapter provides the ability to perform a select, or query from a JD Edwards World Z-table. You can configure this operation using the JD Edwards World Adapter Configuration Wizard.

To do so:

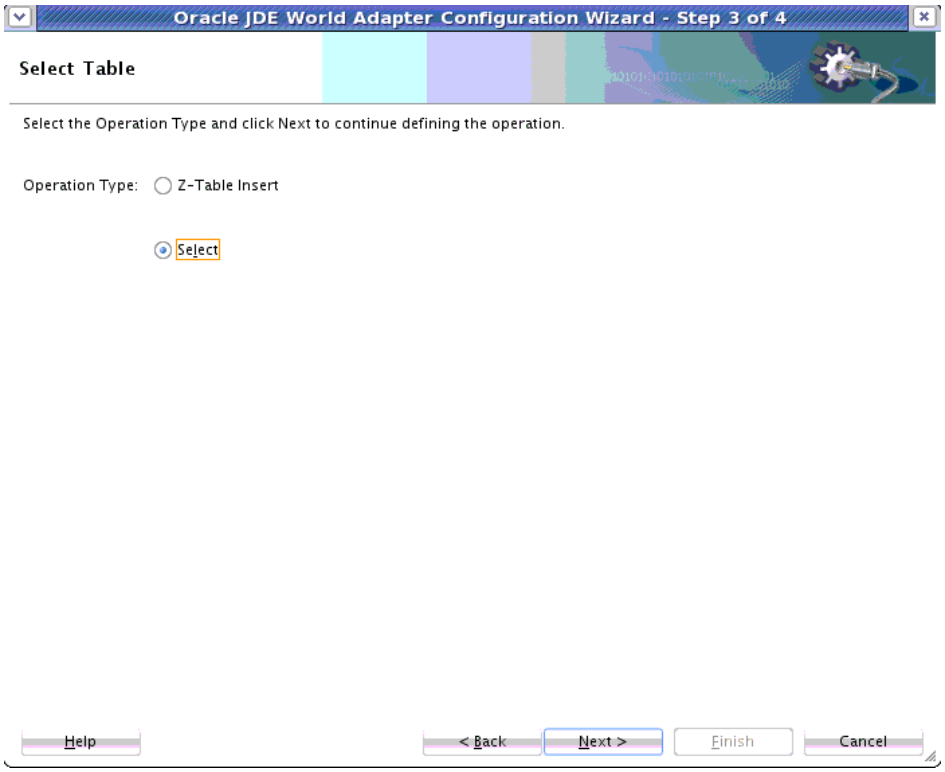
1. On the **Select Table** screen, choose the **Select** Operation Type, and select **Next**.

Figure 15-16 JD Edwards World Adapter Select Table Screen



Here, **Select** has been chosen.

Figure 15-17 JDE World Adapter Configuration Wizard with Select Operation Chosen



2. The **Service Connection** page opens, enabling you to select the root database for this service's query. To do this, you must have a basic understanding of the Z-Tables against which you want to perform the operation and the root database table. It also helps to understand the JD Edwards World System business module against which you are performing insertions or making queries.

   This page enables you to import Z-tables by selecting the **Import Tables** button at the bottom of the page. When you select a Z-Table from this list, you are selecting a table that is the root database for the operation.

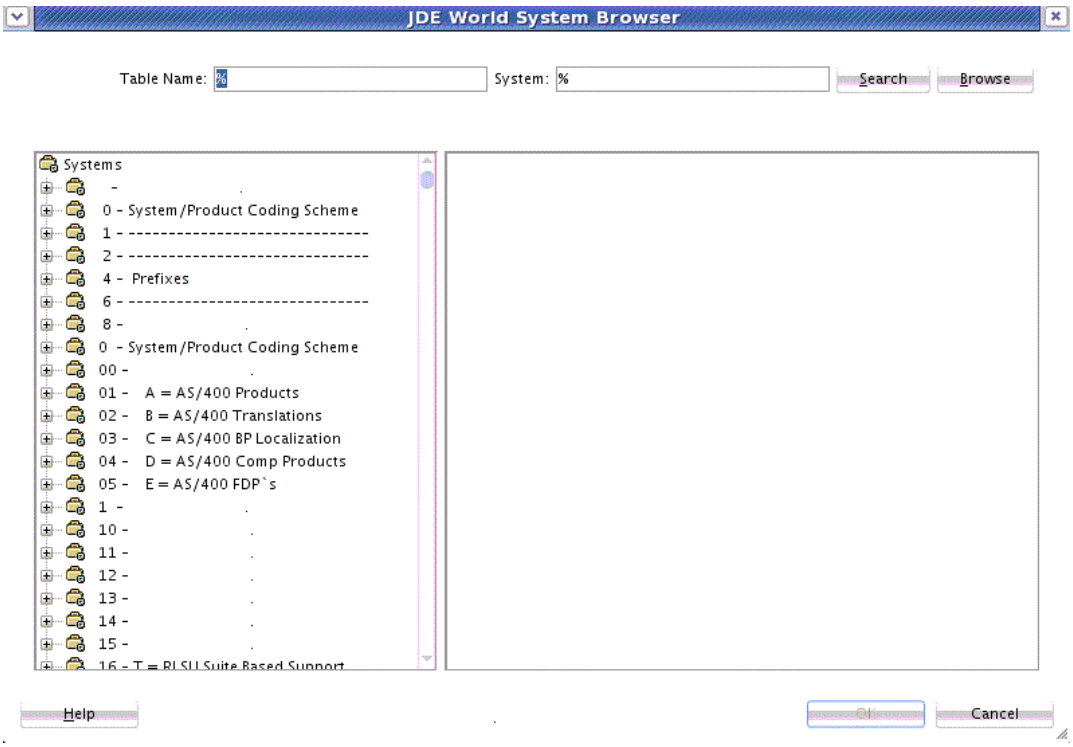
   You can also remove Z-tables you have added. To do so:

   - Click the **Import Tables** button to import Z-tables, and select the table you want as the root database table, or
   - After importing tables, you can remove them. To remove a table from the list, select the appropriate tables, then select the **Remove Tables** button to remove Z-Tables from the list of ones you want imported.
   - Click **Next**.

   Figure 15-18 JDE World Adapter Configuration Wizard Service Connection Screen

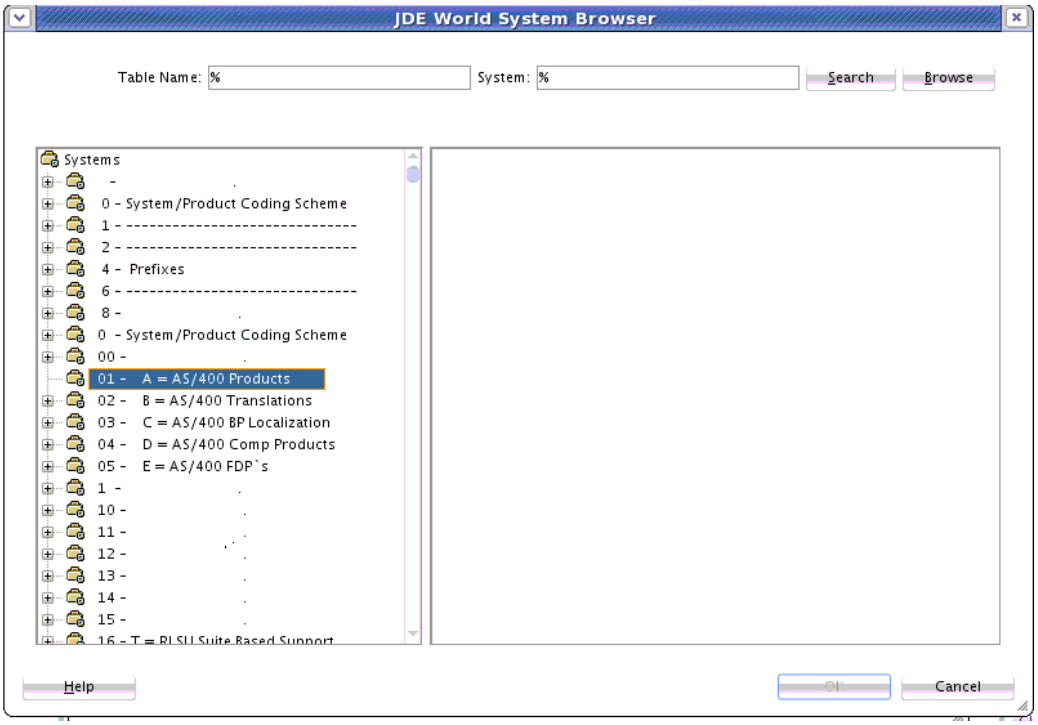
3. The JD Edwards World Adapter **System Browser** appears. Here you can browse for tables you want to import.

Figure 15-19 The JDE World System Browser



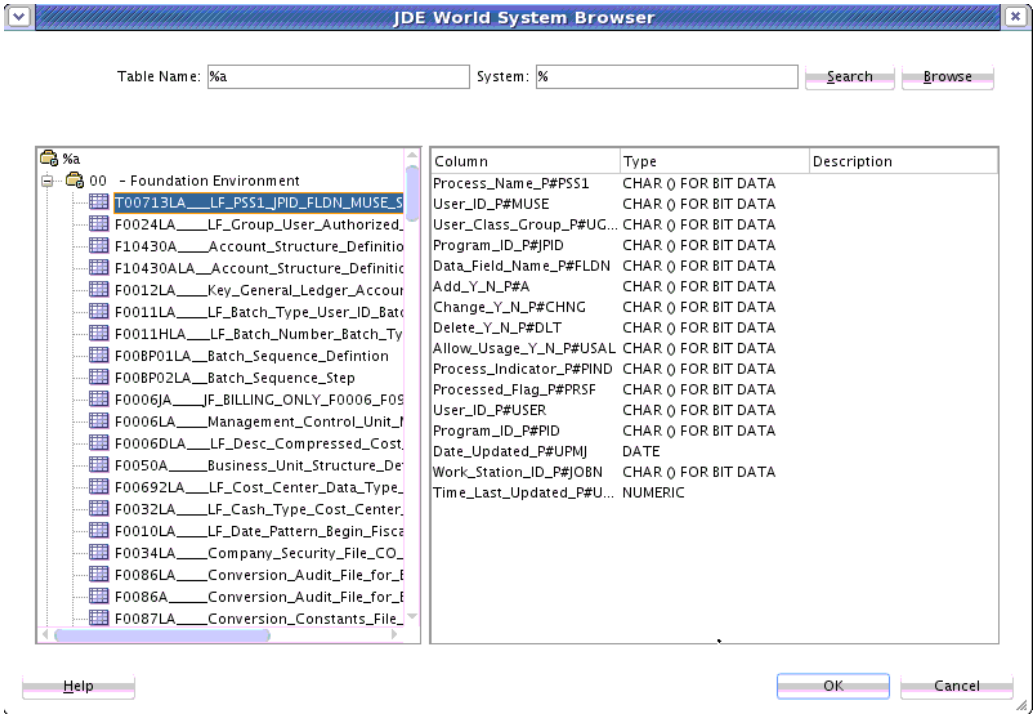
4. Click on a system table to select a system table in the JDE World System Browser.

Figure 15-20 Selecting a System Table in the JDE World System Browser



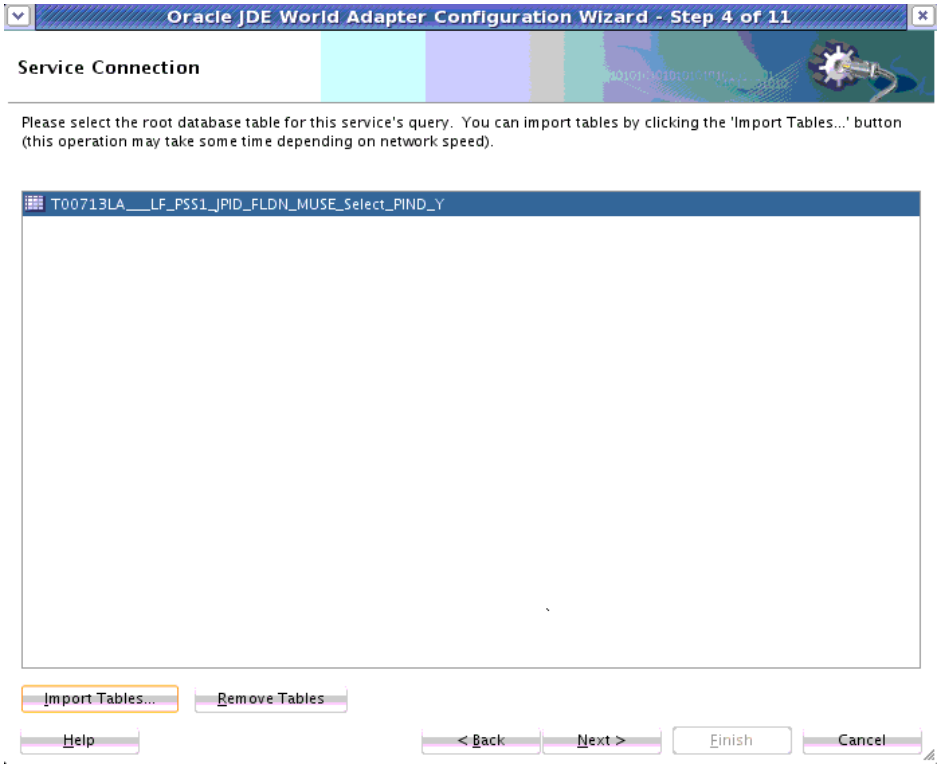
5. Once you have selected a system table, you can expand the table's node in the display and search for columns within tables. Click **OK** to select what you have chosen for importing.

Figure 15-21 Using the JDE World System Browser to Display Column and Type Information



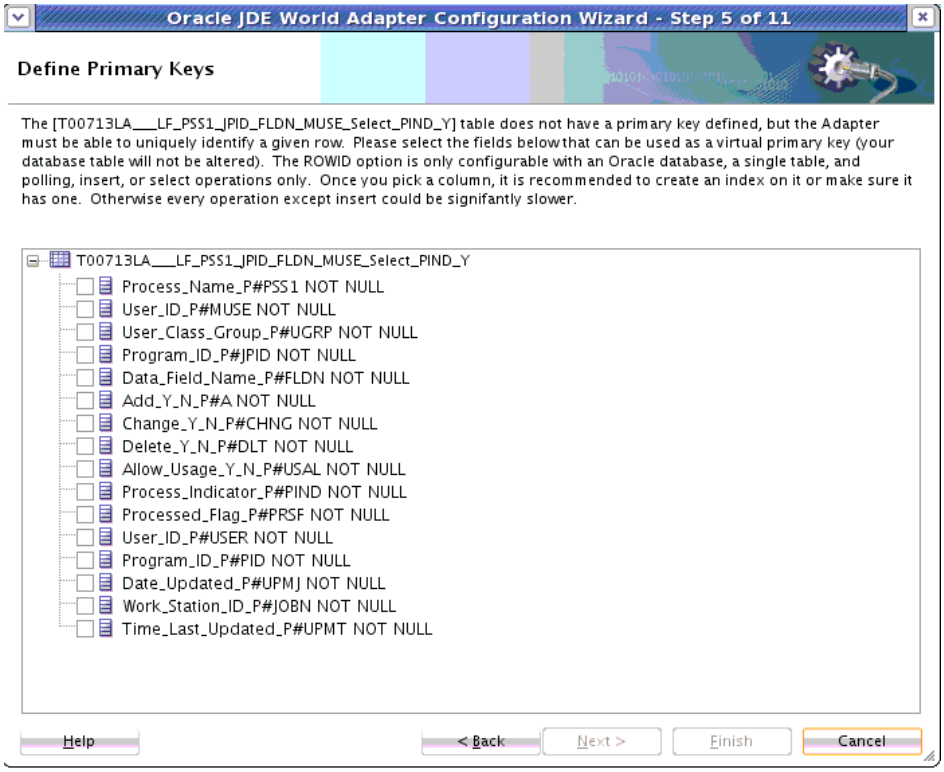
6. Once you have selected root database tables for importing, the **Service Connection** screen reappears, and asks you to choose a table for this service's querying. Select the root database table, and click **Next** to import it. You can also click the `Remove Tables` button to remove tables from your prospective Import list that you do not wish to import,

Figure 15-22 JDE World Adapter Configuration Wizard Service Connection Screen with Tables Ready for Import



7. The **Define Primary Keys** Screen appears if you must define a primary key. The screen gives you the opportunity to define a Primary Key. The Adapter must be able to uniquely identify a given row. As it states, the ROWID option is only configurable with Oracle databases, a single table, and polling, insert, and select operations only. The screen supplies additional information about defining primary keys. To use this screen, select a Primary Key candidate or candidates and click **Next**.

Figure 15-23 The JDE Adapter Configuration Wizard Define Primary Keys Screen



8. If you selected a table and chose Import, the JDE Edwards World Adapter uses the table as the root table for your query operation and the JDE World Adapter Configuration Wizard **Relationships** screen appears.

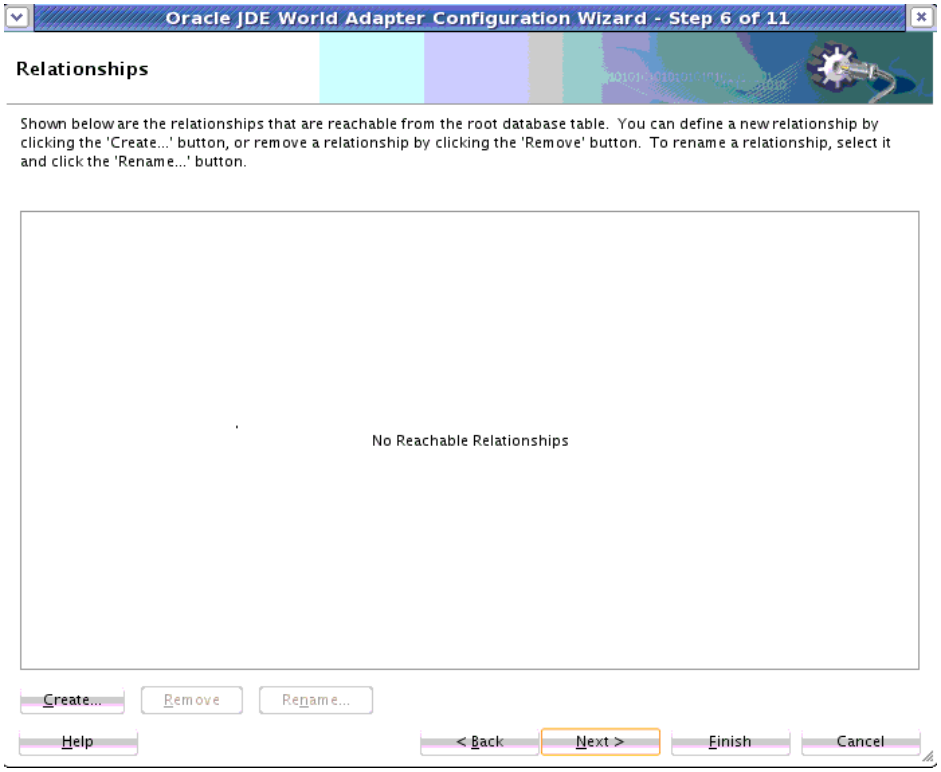
   This screen shows the relationships that are reachable from the root database table you have indicated on the previous screen. If there are no reachable relationships between tables, the screen says "No reachable relationships."

   To use this screen:

   - Define a new relationship by selecting the **Create** button, or
   - Select **Remove** to remove a relationship.
   - To rename a relationship, select the relationship and click **Rename**.

   When you have finished defining, renaming or removing a relationship, or if you have no relationships to create or modify, select **Next**.

Figure 15-24 The JDE World Adapter Configuration Wizard Relationships Screen

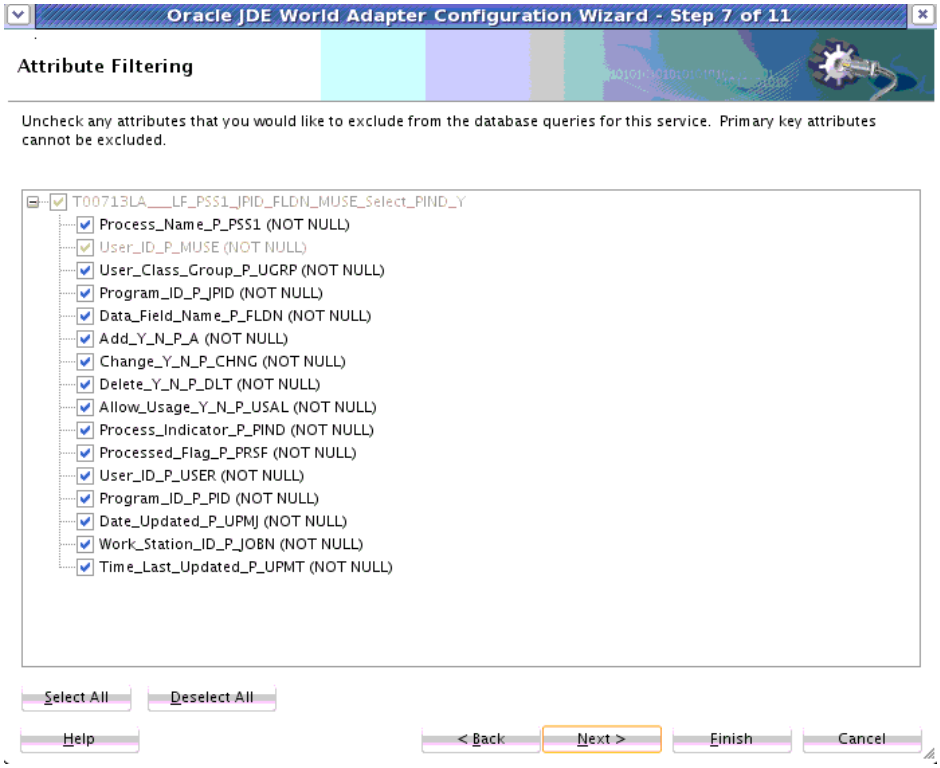


9. The JD Edwards World Adapter Configuration Wizard Attribute Filtering screen appears. See Figure 15-25.

   You can use this screen to exclude any attributes from your database queries. To do so:

   - Uncheck any attributes you do not want to use in your queries.
   - Use the **Select All** button at the bottom of the screen to select all attributes.
   - Use the **Deselect All** button to deselect attributes.
   - Click **Next**.

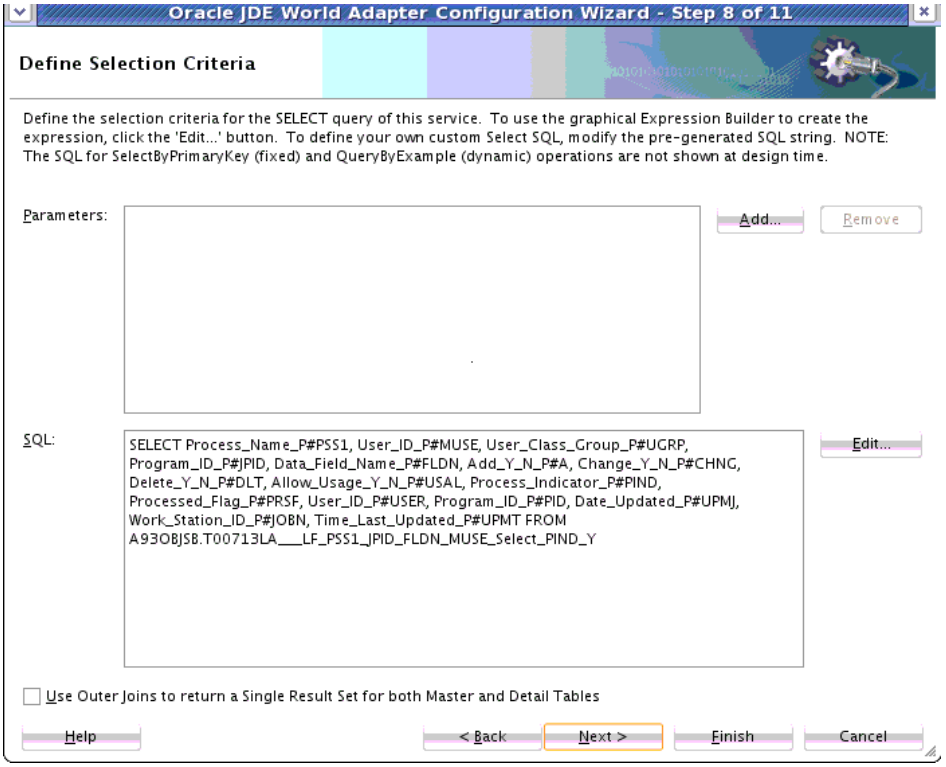
Figure 15-25 The JD Edwards World Adapter Configuration Wizard Attribute Filtering Screen



10. The JD Edwards World Adapter Configuration Wizard Define Selection Criteria Screen appears, where you can define the Selection Criteria for this JD Edwards World Adapter reference. See Figure 15-26

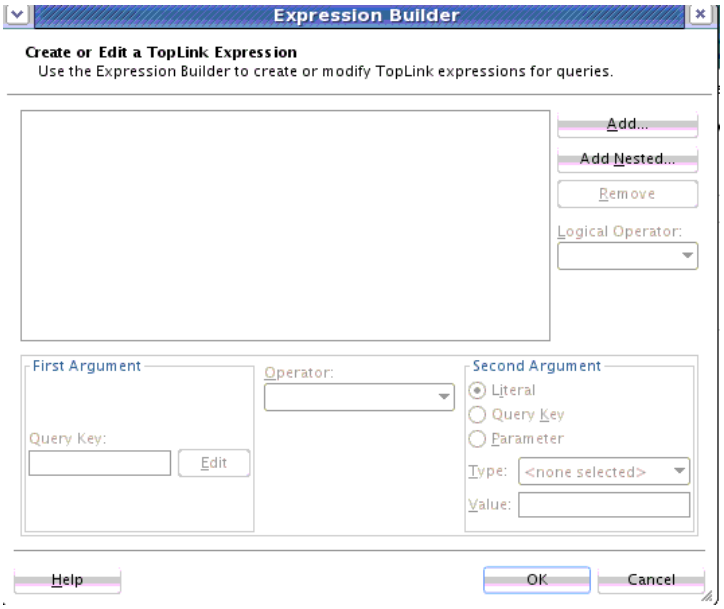
- Enter the parameters in the field at the top and/or select the **Add** button to add additional SQL criteria.

Figure 15-26 The JDE World Adapter Configuration Wizard Define Selection Criteria Screen



- Select **Edit** to use the SQL Expression Builder to create or modify the SQL Expression. See Figure 15-27 for an example of the screen.

Figure 15-27 The Expression Builder



- To define your own custom Select SQL, modify the pre-generated SQL string in the Expression Builder, and click **OK** to use the modified SQL.

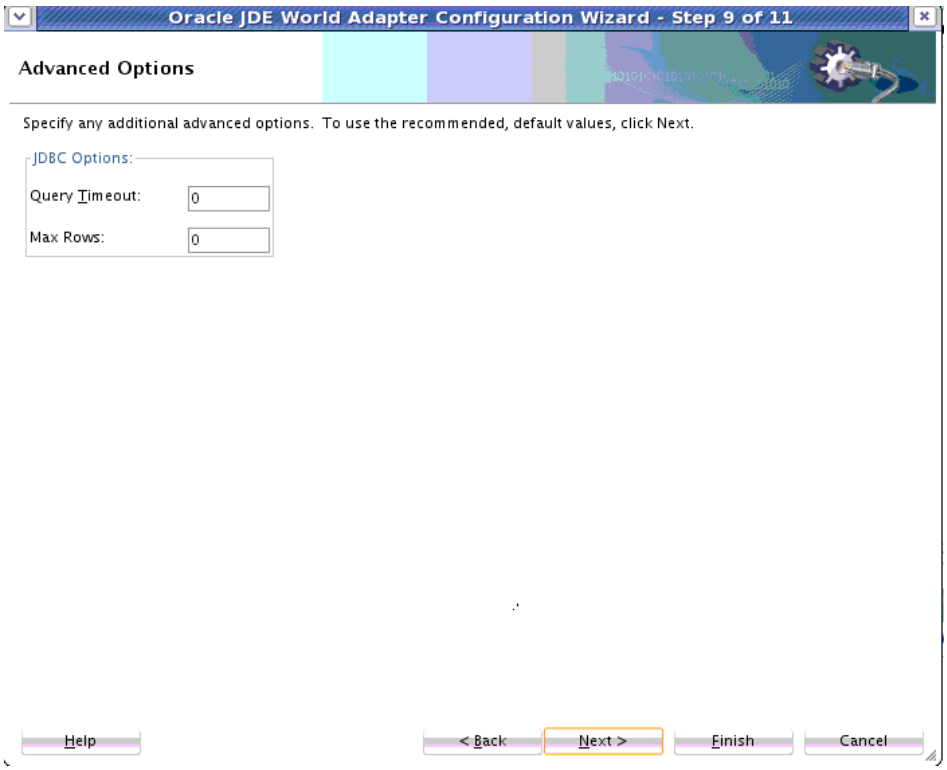
- Back on the Define Selection Criteria Screen, select **Use Outer Joins to return a Single Result Set for both Master and Detail Tables** to use an advanced feature that influences how many total statements are used when querying against multiple related tables. The safest method is to use the default (1 per table), and this feature attempts 1 in total, by performing an outer join on all related tables into a single result set.

11. The JD Edwards World Adapter Configuration Wizard Advanced Options screen appears. See Figure 15-28.

    On this screen, you can enter new values for either or both of the JDBC options provided, or select **Next** to accept the default values of 0.

    - **Query Timeout** - enables you to configure a timeout value on the Select operation.
    - **Max Rows** - the maximum number of rows to return from the Select operation.
    - Click **Next** when finished.

Figure 15-28 The JDEdwards World Adapter Configuration Wizard Adapter Advanced Options Screen



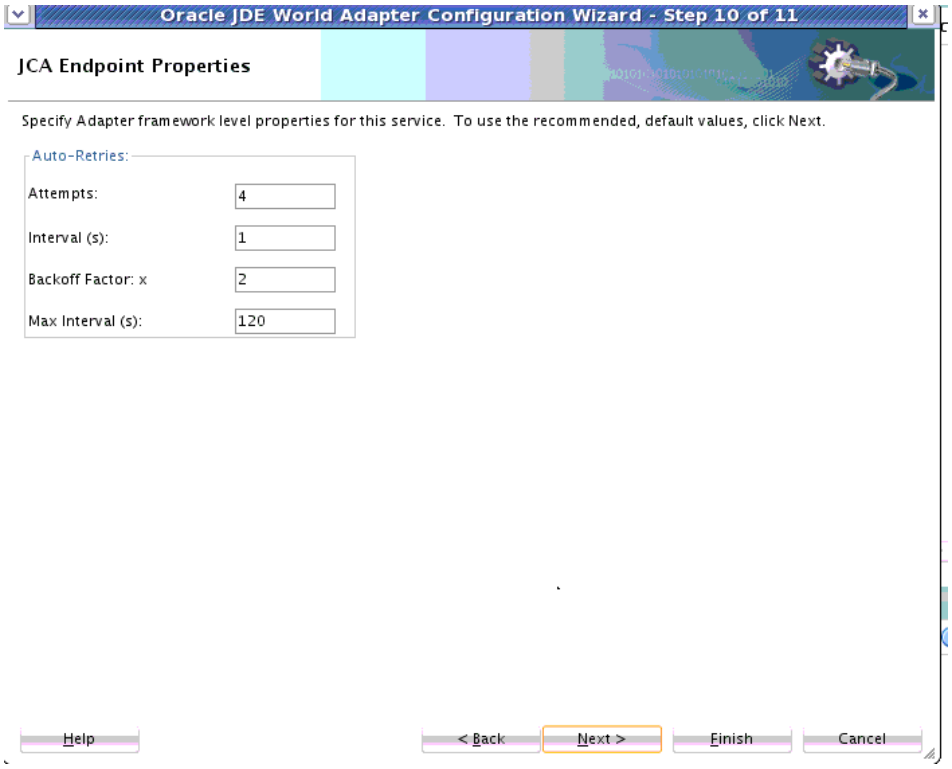
12. The JDE World Adapter Configuration JCA Endpoint Properties Screen appears. See Figure 15-29. On this screen, you can specify Adapter framework-level properties.

    To use the recommended values, select **Next**. These properties include:

    - **Attempts**-The number of retries. This defaults to 4.
    - **Interval(s)**-The delay, or interval, between retries. This value defaults to 1.

- **Backoff factor:x**- Enables you to wait for increasing periods of time between retries. For example, 9 attempts with a starting interval of 1 and a back off of 2 leads to retries after 1, 2, 4, 8, 16, 32, 64, 128, and 256 (2^8) seconds. This value defaults to 2.
- **Max Interval**- The maximum interval for retries. This value defaults to 120 seconds.

Figure 15-29 Oracle JDEdwards World Adapter Configuration Wizard JCA Endpoint Properties Screen



13. The JDEdwards World Adapter **Finish Screen** appears. You have finished defining the JD Edwards World Adapter.

Configuration Files

The WSDL and .jca configuration files for each of the example configured JD Edwards Adapters, using the `Addressbook` tables in this chapter, follow for your reference.

Insert Operation Example WSDL and .jca File

The following example shows the WSDL file for the Insert operation example.

Example - WSDL File for Insert Operation

```
<wsdl:definitions
     name="jdeworldReference"
     targetNamespace="http://xmlns.oracle.com/pcbpel/
         adapter/jdeworld/JDEW_selectAll/Insert/jdeworldReference"
     xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/jdeworld/JDEW_selectAll/
Insert/jdeworldReference"
     xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
```

```
     xmlns:top="http://xmlns.oracle.com/pcbpel/adapter/
               db/top/jdeworldReference"
    >
  <plt:partnerLinkType name="jdeworldReference_plt" >
    <plt:role name="jdeworldReference_role" >
      <plt:portType name="tns:jdeworldReference_ptt" />
    </plt:role>
  </plt:partnerLinkType>
    <wsdl:types>
      <schema xmlns="http://www.w3.org/2001/XMLSchema" >
        <import namespace="http://xmlns.oracle.com/pcbpel/
            adapter/db/top/jdeworldReference" schemaLocation="../Schemas/
jdeworldReference_table.xsd" />
      </schema>
    </wsdl:types>
    <wsdl:message name="F01018Z____Address_Book_Email_URL_Z_FileCollection_msg">
        <wsdl:part name="F01018Z____Address_Book_Email_URL_Z_FileCollection"
element="top:F01018Z____Address_Book_Email_URL_Z_FileCollection"/>
    </wsdl:message>
    <wsdl:portType name="jdeworldReference_ptt">
        <wsdl:operation name="zFileInsert">
            <wsdl:input
message="tns:F01018Z____Address_Book_Email_URL_Z_FileCollection_msg"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

The following example shows the .jca file for the Insert operation example.

Example - .jca File for Sample Insert Operation

```
<adapter-config name="jdeworldReference" adapter="jdeworld"
    wsdlLocation="../WSDLs/jdeworldReference.wsdl"
       xmlns="http://platform.integration.oracle/blocks/
                 adapter/fw/metadata">
  <connection-factory UIConnectionName="JDEWConnection1"
location="eis/JDEWorld/JDEWConnection1"/>
  <endpoint-interaction portType="jdeworldReference_ptt" operation="zFileInsert">
    <interaction-spec className="oracle.tip.adapter.jdeworld.
JDEWorldZTableInteractionSpec">
      <property name="DescriptorName"
value="jdeworldReference.F01018Z____Address_Book_Email_URL_Z_File"/>
      <property name="MappingsMetaDataURL" value="jdeworldReference-or-
mappings.xml"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

Select Operation Example WSDL and .jca File

The following example shows the WSDL file for the sample Select operation example.

Example - WSDL File for Sample Select Operation

```
<wsdl:definitions
     name="jd"
     targetNamespace="http://xmlns.oracle.com/pcbpel/
          adapter/jdeworld/JDEW_selectAll/JDED_select1/jd"
     xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns="http://xmlns.oracle.com/
```

```
          pcbpel/adapter/jdeworld/JDEW_selectAll/JDED_select1/jd"
     xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
     xmlns:top="http://xmlns.oracle.com/pcbpel/adapter/db/top/jd"
    >
  <plt:partnerLinkType name="jd_plt" >
    <plt:role name="jd_role" >
      <plt:portType name="tns:jd_ptt" />
    </plt:role>
  </plt:partnerLinkType>
    <wsdl:types>
      <schema xmlns="http://www.w3.org/2001/XMLSchema" >
        <import namespace=
         "http://xmlns.oracle.com/pcbpel/adapter/db/top/jd"
schemaLocation="../Schemas/jd_table.xsd" />
      </schema>
    </wsdl:types>
    <wsdl:message name="jdSelect_inputParameters">
       <wsdl:part name="jdSelect_inputParameters"
element="top:jdSelectInputParameters"/>
    </wsdl:message>
    <wsdl:message name=
           "F0101_____Address_Book_MasterCollection_msg">
       <wsdl:part name=
       "F0101_____Address_Book_MasterCollection"
element="top:F0101_____Address_Book_MasterCollection"/>
    </wsdl:message>
    <wsdl:portType name="jd_ptt">
        <wsdl:operation name="jdSelect">
            <wsdl:input message="tns:jdSelect_inputParameters"/>
            <wsdl:output message=
            "tns:F0101_____Address_Book_MasterCollection_msg"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

The following example shows the .jca file for the sample Select operation

Example - .jca File for Sample Select Operation

```
<adapter-config name="jd" adapter="jdeworld"
          wsdlLocation="../WSDLs/jd.wsdl" xmlns="http://platform.integration.oracle/
blocks/
               adapter/fw/metadata">
  <connection-factory UIConnectionName="JDEWConnection1" location="eis/JDEWorld/
JDEWConnection1"/>
  <endpoint-interaction portType="jd_ptt" operation="jdSelect">
    <interaction-spec
className="oracle.tip.adapter.jdeworld.JDEWorldReadInteractionSpec">
      <property name="DescriptorName"
           value="jd.F0101_____Address_Book_Master"/>
      <property name="QueryName" value="jdSelect"/>
      <property name="MappingsMetaDataURL"
            value="jd-or-mappings.xml"/>
      <property name="ReturnSingleResultSet" value="false"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

16

Oracle JCA Adapter Tuning Guide

This chapter provides a summary of the most important tuning and performance considerations for the Oracle JCA Adapter framework and for a set of JCA Adapters, and recommendations are provided to assist you in discovering the symptoms of tuning issues and in resolving the issues. The recommendations consist of a discussion of the most important tuning parameters, or properties, related to an Adapter, and sometimes a specific tuning practice associated with the parameter.

The list of adapters includes:

- Oracle JCA Adapter Framework Performance and Tuning
- JMS Adapter
- AQ Adapter
- File/FTP adapter
- Database Adapter

Each section includes the following for each of the tuning guide parameters related to the general framework or a specific adapter:

- A general description of the practice or parameter.
- Symptoms if not properly tuned: what are the symptoms of improper tuning related to this parameter or practice?
- Downside of tuning: of what possible issues do you need be aware, related to these parameters and practices?
- What to do if symptoms occur: how can you adjust the parameter or practice under consideration to ensure better performance?

Oracle JCA Adapter Framework Performance and Tuning

The most important tuning practices and parameters for the Oracle JCA Adapter framework include:

- payloadSizeThreshold
- minimumDelayBetweenMessages

payloadSizeThreshold

This parameter controls the maximum size (measured in bytes) acceptable to any JCA adapter when data is consumed, that is, either asynchronously received (JCA Service) during endpoint polling, or synchronously read using a JCA Reference.

If the size of a (native) message being consumed exceeds the configured `payloadSizeThreshold` (configured on a per SCA Service or Reference basis) the message will be rejected.

When parsing a native message into Oracle XDK XML, the message will require 10-15 times the amount of memory occupied by the native message. For example, a flat file in the file system of size 1Mb will consume 10-15Mb on the JVM heap (as DOM).

The default value for this property is -1, which means "unlimited".

There is a global `payloadSizeThreshold` (in the AdapterConfigMBean) which can be set using the Fusion Middleware Control and which becomes the default limit for all JCA endpoints, which have not explicitly configured a specific value for `payloadSizeThreshold`.

In other words, the endpoint setting always takes precedence over the global setting.

This setting must typically be configured if there is any doubt about the message size that can be expected.

If the application itself is generating the messages being consumed, assumptions related to the application can be made that would alleviate the need for configuring this property.

The goal of the property is to limit the amount of JVM heap space occupied by incoming adapter messages. The value would be (at least) impacted by the following factors within synchronous or asynchronous business flow:

- **Synchronous flow**. Each inbound consumed message must stay on the JVM heap for a longer time, for the duration of the synchronous instance execution. The number of adapter polling threads determines how many message instances exists concurrently in the JVM. The more polling threads, the lower the `payloadSizeThreshold`.
- **Asynchronous flow**. The inbound message exists on the heap for a short time, between the time the adapter reads and translates the incoming message and until the first downstream service engine receives and persists the message. However, the threading model (for example, BPEL worker threads) for these downstream service engines decides how many instances can exist concurrently on the heap, which you need to factor back into a reasonable `payloadSizeThreshold` value.

DOM or Scalable DOM

If Streaming has been enabled, that is, the adapter is producing SDOM (Scalable DOM) instances, you can use higher values for `payloadSizeThreshold` because the Oracle XDK keeps most of the XML in paging storage. This depends on the DOM access pattern employed in xpath, xslt and xqueries executed on the instance.

How much less memory will be used by SDOM instances largely depends on your business logic, so empirical test data should guide your setting of this parameter.

Synchronous Consume

This use case is related to situations where the business logic invokes the adapter to perform a synchronous read operation, that is, read a file or select data from a table. If, for example, there is a risk that a SQL select operation would return too many (nested) rows, setting a reasonable value for `payloadSizeThreshold` is highly recommended. If the size is exceeded, a business fault is thrown and it is up to the business logic to decide compensatory action, that is, there is no automatic rejection involved.

Symptoms if payloadSizeThreshold is Not Properly Tuned

You must choose a well considered value for `payloadSizeThreshold`, which is balanced against the competing concerns of:

- Protecting the JVM heap from becoming swamped with over-sized messages, versus
- The problem of having too many messages being rejected.

Consequently, leaving the value at the default -1 (unlimited) or setting it too high can lead to fatal runtime failures when the JVM runs out of available heap space, if there are too many polling or processing threads allowed to work in parallel on separate inbound messages.

Setting the value too low causes too many messages to be rejected, that is, doing so unnecessarily disrupts normal business flow, and requires either scheduled or manual recovery of messages which likely are of an acceptable size.

Downside of Tuning

The cost involved in calculating the size of the native message is the potential downside of configuring this property. Setting a threshold value can lead to a measurable performance impact especially in the case of the Database Adapter.

Recommendations if Symptoms Occur

If out-of-memory (OOM) errors start occurring in SOA are seen in the logs), or overall system slowdown is detected or too frequent garbage collection starts occurring - if any of these occur and you can correlate them back to the inflow of over-sized native adapter messages, a message threshold should likely be imposed immediately.

You can impose a new threshold dynamically, either globally by using the configuration MBean (which would apply to all endpoints) or by changing the Binding Property `payloadSizeThreshold` of one or more inbound JCA endpoints (the endpoints suspected of receiving very large messages).

minimumDelayBetweenMessages

`minimumDelayBetweenMessages` is a simple mechanism to delay the execution of an adapter polling thread.

It adds a trivial thread sleep as part of the instance execution, that is, on a per polling thread basis. The setting is measured in milliseconds. The delay is added immediately before invoking the next downstream component.

If the adapter initiates a transaction as part of the poller thread, the transaction timeout must accommodate the maximum of the worst case acceptable instance execution, plus the value of `minimumDelayBetweenMessages`.

If the instance execution itself (the time between the two most recent message publications) exceeds the value of `minimumDelayBetweenMessages`, no delay will be added (hence the name "minimumDelay...").

Symptoms if Not Properly Tuned

If not configured, there is no delay between messages coming from the inbound JCA adapter endpoint.

If the business process has an asynchronous structure/signature, this situation can lead to overloading of the downstream business logic if it is not able to keep up with the speed of inbound messages arriving from the adapter endpoint (where messages pile up in the dispatcher queue).

If, alternatively, the business process flow is synchronous, the posting adapter polling thread will not reacquire a new incoming message before the instance execution is complete of the previous message (this only pertains to the scope leading up to the first downstream dehydration point).

However, configuring a high value for minimumDelayBetweenMessages can make the downstream business process unnecessarily idle, or lead to JTA/XA transaction timeouts.

Downside of Tuning

Setting a value for minimumDelayBetweenMessages imposes a fixed delay, which does not adjust to changing performance characteristics of the downstream business log. That is, if the downstream process flow performance improves, the minimumDelayBetweenMessages is still applied unconditionally.

Recommendations if Symptoms Occur

If the downstream business logic becomes increasingly backlogged, for example, in terms of the number of unprocessed messages in the BPEL dispatcher queue, this backlog indicates the logic cannot keep up with the rate of incoming adapter messages.

You can then apply this property to the JCA endpoint, choosing a value so the aggregate delay across all polling threads for the endpoint matches the execution speed of the asynchronous business process.

The rule is that if there are 10 polling threads, the effective delay is = minimumDelayBetweenMessages / # of polling threads.

JMS Adapter

The most important tuning practices and parameters for the JMS Adapter include:

- adapter.jms.receive.threads
- EnableStreaming
- adapter.jms.receive.timeout

adapter.jms.receive.threads

This parameter indicates the number of poller threads that are created when an adapter endpoint is activated. The default is 1. Each poller thread receives its own message that is processed independently and thus allows for increased throughput.

Symptoms if Not Properly Tuned

If the message processing is slow, resulting in an increase in the queue backlog, one way to speed up processing is to increase the number of poller threads.

Downside of Tuning

An increased number of polling threads increases the system resources required for concurrent processing.

Recommendations if Symptoms Occur

If symptoms occur, increase the thread while performance continues to scale linearly.

EnableStreaming

The default for `EnableStreaming` is false. Its importance is usecase-specific, but when the property is set to `true`, the payload from the JMS destination is streamed (that is, the adapter generates an instance of SDOM) instead of processing the payload as an in-memory DOM. You must use the SDOM feature while handling large payloads.

Symptoms if Not Properly Tuned

Large payloads from the JMS destination can result in an out-of-memory condition.

Downside of Tuning

Setting this parameter to `True` incurs an extra cost to process SDOM.

Recommendations if Symptoms Occur

Turn streaming on as it enables a larger value of the `PayloadSizeThreshold` property to be used with the JMS adapter.

adapter.jms.receive.timeout

The default value for this parameter is 1 second. This timeout value is used for the synchronous receive call, and is the time after which the `receive()` API times out if a message is not received on the inbound JMS queue.

Symptoms if Not Properly Tuned

Symptoms can include high CPU usage during the idle cycle (when there are no messages in the queue for an extended time.)

Downside of Tuning

A large value for this parameter can lead to transaction timeouts.

Recommendations if Symptoms Occur

Increase the timeout value, taking into consideration the total transaction timeout value that is set for the SOA environment.

AQ Adapter

The most important tuning practices and parameters for the AQ Adapter include:

- adapter.aq.dequeue.threads
- EnableStreaming
- DequeueTimeOut

adapter.aq.dequeue.threads

This is the number of poller threads that are created when an AQ adapter endpoint is activated. The default is 1. Each poller thread receives its own message that is processed independently and which thus allows for increased throughput.

Symptoms if Not Properly Tuned

Not necessarily a symptom of improper tuning, but if message processing is slow, resulting in an increase in the AQ queue backlog, one way to increase message processing speed is to increase the number of threads.

Downside of Tuning

Increased threads will probably increase system resources that are required for concurrent processing.

Recommendations if Symptoms Occur

Increase poller threads for as long as performance continues to scale linearly.

EnableStreaming

This AQ parameter's default is `false` and the parameter is use-case specific. You must use this feature while handling large payloads.

When the property is set to `true`, the payload from the JMS destination is streamed (the adapter generates an instance of SDOM) instead of processing as an in-memory DOM.)

Symptoms if Not Properly Tuned

Large payloads will most likely result in out of memory.

Downside of Tuning

Incurs extra cost to process SDOM.

Recommendations if Symptoms Arise

If out of memory conditions occur, turn streaming on as it enables you to use a larger value of `PayloadSizeThreshold` property with the AQ adapter.

DequeueTimeOut

`DequeueTimeOut` is the interval after which the `dequeue()` API times out if a message is not received on the inbound queue. The default value is 1 second.

Symptoms if Not Properly Tuned

If this parameter is not properly tuned, you might see high CPU usage during the idle cycle (when there are no messages in the queue for an extended time.)

Downside of Tuning

A large value can lead to transaction timeouts.

Recommendations if Symptoms Arise

Increase this timeout parameter value, taking into consideration the total transaction timeout set for the SOA environment.

File/FTP adapter

The most important tuning practices and parameters for the File/FTP Adapter include:

- Thread Count and Single Thread Model
- maxRaiseSize
- PublishSize
- ChunkSize

Thread Count and Single Thread Model

The inbound design for File and FTP Adapters is based on a producer/consumer model where each deployment of a composite application (with inbound File/Ftp Adapter service) results in the creation of a single poller thread and an optional number of processor threads.

The following steps highlight the default threading behavior of File and Ftp Adapters:

1. The poller thread periodically scans for files in the input directory based on a configured polling frequency.
2. For each new file that the poller detects in the inbound directory, the poller enqueues information such as file name, file directory, modified time, and file size into an internal system-wide in-memory queue.
3. A global pool of processor threads (four in number) wait to process from this in-memory queue.
4. The items in the in-memory queue are dequeued and processed by these threads; processing includes translation of file content into XML and publishing to SOA infrastructure, followed by post-processing, for example, deletion/archival of the file.

This is the default threading behavior for both the File and FTP Adapters. However, you can change this behavior by configuring either a `ThreadCount` or `SingleThreadModel` property in the inbound jca configuration file.

Specifying a finite value for `ThreadCount` results in that particular File/FTP adapter inbound service to change into a partitioned threaded model, where each service receives its own in-memory queue and its own set of dedicated processor threads.

Because the processor threads themselves come from the SOA Work manager, they consume memory/CPU time and hence this property must be configured carefully. Recommendation is to start with a value of 2 and slowly increase the speed as required; the system, however, sets the upper bound for this configuration of `ThreadCount` at 40.

Setting `SingleThreadModel` as true in the JCA configuration results in the poller assuming the role of the processor. In other words, the poller scans the inbound directory and processes the files in the same thread (one after another).

This parameter is particularly useful if you want to throttle the system. You can also use this parameter if you want to process files in a specific order (for example, process files in descending order of last modified time)

Symptoms if Not Properly Tuned

Low performance in high volume situations when multiple inbound services are running due to a single shared in-memory queue and shared processor threads.

Downside of Tuning

Increased number of processor threads increase the amount of system resources required. A dedicated in-memory queue for each service also increases the memory requirement.

Recommendations if Symptoms Arise

Start increasing the value of the `ThreadCount` from 2 and slowly increase until you observe a linear performance increase.

maxRaiseSize

The `MaxRaiseSize` parameter is used by the inbound File/FTP Adapter to decide how many files the inbound adapter poller thread raises to the in-memory queue in a single polling cycle.

For example, if you have configured a Polling Frequency of 1 minute and a MaxRaiseSize of 10, the poller thread raises a maximum of 10 files every 1 minute into the in-memory queue.

This parameter is useful when the File/FTP Adapter is configured to execute in active/active cluster. In this case, there is more than one poller thread (one per managed server actually) polling the same directory in the cluster; this parameter ensures the files get more or less evenly distributed between the nodes and that no single managed server cannibalizes the processing of files.

Symptoms if Not Properly Tuned

If this parameter is not correctly tuned, balancing is not correct; for example, one managed server in an active-active cluster will be processing most of the messages.

Downside of Tuning

A low value of `MaxRaiseSize` can result in slower performance due to fewer files being processed in each polling cycle and additional poller thread sleeptime.

Recommendations if Symptoms Arise

Based on the expected load of your application, you can synchronize the `MaxRaiseSize` and Polling frequency to ensure less sleeptime within each polling cycle and also proper load balancing across poller threads.

PublishSize

The debatching usecase within the File/FTP Adapter occurs when a single inbound file can be broken down into multiple batches and each batch is processed individually. For example, if you have a single file that has millions of invoices, you cannot process the entire file in one attempt.

With debatching enabled, the File/FTP Adapter reads from the underlying stream and translates the first 1000 (given a `PublishSize` of 1000) invoices into XML and publishes to SOA; this process continues until the entire file has been processed. Additionally, the adapter saves enough information between iterations so it can restart the debatching from the point it left off during crash recovery.

You should specify a sufficiently large value for the `PublishSize` for better performance. For example, if you file has 1,000,000 logical records, then setting the `PublishSize` to say 10,000 would result in the entire file being processed in 100 iterations.

Each publish from the adapter results in many database activities, for example, saving to dehydration store and re-hydration at a later point of time. Reducing the number of publishes (by having a larger value for the `PublishSize`) improves performance.

Symptoms if Not Properly Tuned

Low performance due to a higher number of publish calls and many database activities.

Downside of Tuning

A larger value of `PublishSize` requires additional system resources.

Recommendations if Symptoms Arise

Increase the `PublishSize` for the duration of linear performance improvement.

ChunkSize

Chunked Read is analogous to debatching, but, it applies to the outbound File/FTP adapter and, more importantly, it is supported only in BPEL.

The usecase is a BPEL process that employs Chunked-Read uses an `<invoke>` activity within a `<while/>` BPEL construct to process a huge file, one logical chunk at a time.

At the end of each `<invoke>`, one logical chunk is returned back to BPEL as XML data (that is, materialized in memory).

The `ChunkSize` parameter defines how many logical records are returned back to BPEL during each `<invoke>`. While setting this parameter, you must be aware that a larger number of iterations in the `<while>` loop in BPEL will result in out-of-memory errors as it results in an increase in amount of audit information being held in memory.

While setting the value of `ChunkSize`, you need to ensure that you do not return too large a chunk of records while keeping the number of iterations to a minimum. For example, if you have a huge file with one million records, you should start with a large value, say 1000.

Symptoms if Not Properly Tuned

Large payloads might result in out-of-memory issues if the property is either not configured or the value is either too high or too low.

Downside of Tuning

You must find a balanced `ChunkSize` based on the expected input size to ensure that you do not return too large a chunk while keeping the number of iterations to a minimum.

Recommendations if Symptoms Arise

Based on the expected input file size and the available system resources, start with a balanced `ChunkSize` and then increase its value until the memory requirements are acceptable.

Database Adapter

This section provides a summary of the most important practices and parameters related to Database Adapter performance and tuning.

These include:

- Use Indexes
- MaxTransactionSize and MaxRaiseSize
- MaxTransactionSize and MaxRaiseSize
- Do not use RowsPerPollingInterval
- Enable Skip Locking true (Use Parameter usesSkipLocking)
- Increase NumberOfThreads

Use Indexes

The following information relates to using indexes with the Database Adapter.

Symptoms if Not Properly Tuned

Polling can be extremely slow. Delete polling strategy with a 1-M relationship can deadlock if the foreign key column is not indexed.

Downside of Tuning

All indexes have a cost. Using pure delete from a flat table and configuring no primary key (using rowid instead) is a viable alternative to needing indexes. Indexes may not be needed if the adapter can poll at a speed where there are never more than a few unprocessed rows (again for DeletePollingStrategy).

Recommendations if Symptoms Arise

Create an index or remove all indexes/constraints and use rowid (advanced, delete polling only).

Additionally, Use indexes is not a parameter which defaults to false. It is something you do separate from the Database Adapter itself by altering the relational schema you are polling against to make sure the SQL the adapter will be executing will be performed optimally. It is not a "tuning knob" per se, but it is still highly important.

Note that with the Rowid option, you get the fast selects, while avoiding the traditional costs of maintaining an index. For more information, see the section on Rowid in the Database Adapter Chapter.

MaxTransactionSize and MaxRaiseSize

Following are the considerations for `MaxTransactionSize` and `MaxRaiseSize`.

Symptoms if Not Properly Tuned

Good way to scale by reducing overhead using number of transactions/fetches (`MaxTransactionSize`) and number of downstream instances (MaxRaiseSize). If moving data in bulk passing multiple rows through the system as one payload can improve performance.

Downside of Tuning

`MaxRaiseSize=1` is often for business constraints, for instance the bpel process is designed to process a single record. Processing multiple records in batch can be slower overall if messages frequently need to be rejected (the bulk transaction must be rolled back and rows retried individually). Also if the end to end process is synchronous then with a high `MaxTransactionSize` and low `MaxRaiseSize` many downstream processes will be participating in a single global transaction, causing transaction timeouts.

Recommendations if Symptoms Arise

In case of transaction timeouts make the post asynchronous (to BPEL) or reduce the ratio `MaxTransactionSize/MaxRaiseSize`.

Note that BPEL has a high per-instance overhead, on the order of 10 times the overhead compared to OSB. Some of this use is dehydration and instance tracking. However, this cost is something of a fixed cost, whether each instance represents 10 rows in a single XML or 1 row in an XML. Consequently, one way to work around this overhead is for the Database Adapter to send and receive slightly larger payloads representing multiple rows. A symptom of such overhead may be slowness compared to a pure JDBC program.

Do not use RowsPerPollingInterval

You need to be very careful in your use of this tuning parameter.

Symptoms if not Properly Tuned

`RowsPerPollingInterval` is an explicit cap on throughput, designed to reduce burst load on downstream components, and must be used very carefully so as not to cap performance too much.

Downside of Tuning

May be set much lower than the burst throughput that the system can actually handle.

Recommendations if Symptoms Arise

Increase or eliminate `RowsPerPollingInterval`.

Enable Skip Locking true (Use Parameter usesSkipLocking)

Enable Skip Locking true is an important parameter.

Symptoms if not Properly Tuned

Polling will not scale with the number of threads as the exclusive locks used will block other threads from picking up any rows.

Downside of Tuning

None, though this parameter is only supported for Oracle Database and SQLServer platforms. Ensure the data source has enough capacity; if you don't have enough connections, adding more threads will simply create more waiting threads.

Recommendations if Symptoms Arise

Ensure `usesSkipLocking = true` (true by default).

Increase NumberOfThreads

This section discusses use of IncreaseNumberof Threads.

Symptoms if not Properly Tuned

Polling will not scale.The definition of scaling is approximately that performance increases linearly with number of threads/agents. Consequently, increasing threads in is a precondition to scaling. The other tuning parameters are to make sure the performance gains are linear (that is, skip locking by letting each thread work without being slowed down by other similar threads).

Downside of Tuning

Unless a distributed polling strategy like `usesSkipLocking=true` is used, increasing threads might not increase concurrency. Make sure the underlying data source connection has at least same the maximum capacity.

Recommendations if Symptoms Arise

Increase threads so long as performance continues to scale linearly.

A

Oracle JCA Adapter Properties

This appendix lists and describes the JCA and binding properties applicable Oracle JCA Adapters, and is meant to be used with the chapters in this book on the specific JCA Adapters, to assist in the configuration of the Adapters.

A normalized message is simplified to have only two parts, properties and payload. In Oracle BPEL Process Manager and Oracle Mediator, you can access, manipulate, and set headers with varying degrees of user interface support. For example, you can preserve a file name from the source directory to the target directory by propagating it through message headers.

Oracle JCA adapters transmit header information through Normalized Message properties (with the exception of the Oracle JCA Adapter for AQ's payload header).

Binding properties are message header properties for the service and reference binding components included in a deployed SOA composite application.

This appendix includes the following sections:

For more information, see "Configuring Service and Reference Binding Components" in the Administering Oracle SOA Suite and Oracle Business Process Management Suite guide.

Oracle File and FTP Adapters Properties

This section describes the properties applicable to the Oracle File and FTP Adapters, including:

- Table A-2
- Table A-3
- Table A-4
- Table A-5
- Table A-6

For properties applicable to all Oracle JCA Adapters, see Generic Oracle JCA Adapter Properties.

Table A-1 JCA Properties for Oracle File and FTP Adapters

Property	Description
`Append`	If this property is set to `true`, it causes Oracle and File and FTP Adapters to append to a file on outbound. If the file does not exist, then a new file is created. The file name can either be specified in the `JCA` file for the outbound operation or in the `jca.file.FileName` header.
`AsAttachment`	If set to `true`, it causes the inbound file to be published as an attachment.
`BatchSize`	Set it to the batch size for the batching transformation.
`CharacterSet`	Set it to the character set for the attachment. This parameter is not used internally by the Oracle File and FTP Adapters, and it is meant for third-party applications that process the attachments published by the Oracle File and FTP Adapters.
`ChunkSize`	Set it to the chunk size for the chunked interaction operation.
`ConcurrentThreshold`	The maximum number of translation activities that can be allowed to execute in parallel for a particular outbound scenario. The translation step during the outbound operation is CPU-intensive and must be guarded because it might cause other applications or threads to starve. The maximum value is `100`.
`ContentType`	Set it to the mime-type of the attachment. This parameter is not used internally by the Oracle File and FTP Adapters, and it is meant for third-party applications that process the attachments published by the Oracle File or FTP Adapter.
`DeleteFile`	If set to `true`, the Oracle File or FTP Adapter deletes the file after processing.
`DirectorySeparator`	When you choose multiple directories, the generated JCA files use semicolon (`;`) as the separator for these directories. However, you can change the separator to something else. If you do so, manually add `<property name="DirectorySeparator" value="`*`chosen separator`*`"/>` in the generated JCA file. For example, to use comma (`,`) as the separator, you must first change the separator to comma (`,`) in the Physical directory and then add `<property name="DirectorySeparator" value=","/>` in the JCA file.
`ElapsedTime`	This property is used for outbound batching. When the time specified elapses, the outgoing file is created. The parameter is of type `int` and is not mandatory. The default value is `1`.
`Encoding`	Set it to the encoding used for the attachment. This parameter is not used internally by the Oracle File and FTP Adapters, and it is meant for third-party applications that process the attachments published by the Oracle File and FTP Adapters.

Table A-1 (Cont.) JCA Properties for Oracle File and FTP Adapters

Property	Description
`ExcludeFiles`	This property specifies the pattern for types of files to be excluded during polling. The property is of type `String` and is not mandatory.
`FileName`	Use this parameter to specify a static single file name during the write operation.
`FileNamingConvention`	This property is used for the naming convention for the outbound write operation file.
`FileSize`	This property is used for outbound batching. The outgoing file is created when the file size condition is met. The parameter is of type `int` and is not mandatory. The default value is `1000` KB.
`IncludeFiles`	This property specifies the pattern for types of files to pick up during polling. The parameter is of type `String` and is mandatory.
`Lenient`	Lenient property works in conjunction with deployment validation. If the inbound directory cannot be read or written to, then the adapter throws a failure and rolls back the deployment. However, if Lenient is set to `true`, then the deployment goes through. The default value of this property is `false`.
`ListSorter`	This property specifies the sorter that the Oracle File and FTP Adapters use to sort files in inbound. You can set this parameter to: • `oracle.tip.adapter.file.inbound.listing.TimestampSorterAscending` to sort the file names by their modified time stamps in ascending manner or • `oracle.tip.adapter.file.inbound.listing.TimestampSorterDescending` to sort the file names by their modified time stamps in descending manner
`LogicalArchiveDirectory`	This property specifies the logical directory in which to archive successfully processed files. The property is of type `String` and is not mandatory.
`LogicalDirectory`	This parameter specifies the logical input directory to be polled. The parameter is of type `String`.
`MaxRaiseSize`	This property specifies the maximum number of files that the Oracle File or FTP Adapter submits for processing in each polling cycle. For example, if the inbound directory has 1000 files and `MaxRaiseSize` is set to `100` and the polling frequency is one minute, then the Oracle File or FTP Adapter submits 100 files every minute.
`MinimumAge`	This parameter specifies the minimum age of files to be retrieved. This specification enables a large file to be completely copied into the input directory before it is retrieved for processing. The age is determined by the last modified time stamp. For example, if you know that it takes three to four minutes for a file to be written, then set the minimum age of pollable files to five minutes. If a file is detected in the input directory and its modification time is within five minutes of the current time, then the file is not retrieved because it is still potentially being written to.
`NumberMessages`	This property is used for outbound batching. The outgoing file is created when the number of messages condition is met. The parameter is of type `int` and is not mandatory. The default value is `1`.
`PhysicalArchiveDirectory`	This property specifies where to archive successfully processed files. The property is of type `String` and is not mandatory.

Table A-1 (Cont.) JCA Properties for Oracle File and FTP Adapters

Property	Description
`PhysicalDirectory`	This property specifies the physical input directory or directories to be polled. The parameter is of type `String`. The inbound directory where the files appear is mandatory. You must specify the physical directory or logical directory.
`PollingFrequency`	This parameter specifies how often to poll a given input directory for new files. The parameter is of type `int` and is mandatory. The default value is 1 minute.
`PublishSize`	This property indicates whether the file contains multiple messages and how many messages to publish to the BPEL process at a time. The parameter is of type `int` and is not mandatory. The default value is `1`. For example, if a certain file has 11 records and this parameter is set to 2, then the file processes 2 records at a time and the final record is processed in the sixth iteration.
`Recursive`	If this property is set to `true`, then the adapter can process all the sub-directories under the main input directory recursively.
`SequenceName`	Specifies the Oracle database sequence name to be used if you have configured the outbound Oracle File or FTP Adapter for High Availability.
`SingleThreadModel`	If the value is `true`, the Oracle File or FTP Adapter poller processes files in the same thread. In other words, it does not use the global in-memory queue for processing.
`SourceFileName`	The source file for the File I/O operation.
`SourcePhysicalDirectory`	The source directory for the File I/O operation.
`SourceSchema`	Set to the schema for the source file.
`SourceSchemaRoot`	Set to the root element name for the source file.
`SourceType`	Set this to `native` if the source file is native and `xml` if the source file is xml.
`TargetFileName`	The target file for the File I/O operation.
`TargetPhysicalDirectory`	The target directory for the File I/O operation.
`TargetSchema`	Set it to the schema for the target file.
`TargetSchemaRoot`	Set it to the root element name for the target file.
`TargetType`	Set this to `native` if the target file is native and xml if the source file is xml.
`ThreadCount`	If this property is available, then the adapter creates its own processor threads rather than depend on the global thread pool processor threads (by default, 4 of them). In other words, this parameter partitions the in-memory queue and each composite application gets its own in-memory queue. • If the `ThreadCount` property is set to `0`, then the threading behavior equals that of the single-threaded model. • If the `ThreadCount` property is set to `-1`, then the global thread pool is used, which equals the default threading model. • The maximum value for the `ThreadCount` property is `40`.

Table A-1 (Cont.) JCA Properties for Oracle File and FTP Adapters

Property	Description
`TriggerFile`	The name of the trigger file that activates the inbound Oracle File or FTP Adapter.
`TriggerFilePhysicalDirectory`	The directory path where the Oracle File or FTP Adapter looks for the trigger files.
`TriggerFileStrategy`	This property defines the strategy that the Oracle File or FTP Adapter uses to look for the specified trigger file in the trigger file directory. The acceptable values are `EndpointActivation`, `EveryTime`, or `OnceOnly`.
`Type`	Set it to `COPY`, `MOVE`, or `DELETE` for the File IO interaction.
`UseRemoteErrorArchive`	This property defines where an error is archived during an Inbound Read. During an Inbound Read operation, if a malformed XML file is read, the malformed file results in an error. The errored file is by default sent to the remote file system for archival. The errored file can be archived at a local file system by specifying the useRemoteErrorArchive property in the jca file and setting that property to false. The default value for this property is true.
`UseHeaders`	This parameter can be set to `true` or `false`. If you must read file headers and skip reading the payload while using inbound Oracle File or FTP Adapter, then set the `UseHeader` property to `true`. This is typically used in large payload scenarios where the inbound adapter is used as a notifier.
`UseStaging`	If set to `true`, then the outbound Oracle File or FTP Adapter writes translated data to a staging file, and later it streams the staging file to the target file. If set to `false`, then the outbound Oracle File or FTP Adapter does not use an intermediate staging file.
`Xsl`	Set it to the `xsl` transformer between the source and the target.

Table A-2 JCA Properties Specific to Oracle FTP Adapter

Property	Description
`FileType`	Set this property to either `ascii` or `binary` depending on the requirement.
`SourceIsRemote`	Set this property to `false` if you want to notify the Oracle FTP Adapter that the source for the IO operation is a local file system as opposed to a remote FTP server.
`TargetIsRemote`	Set this property to `false` if you want to notify the Oracle FTP Adapter that the target for the IO operation is a local file system as opposed to a remote FTP server.
`UseNlst`	Set this property to `true` if you want the Oracle FTP Adapter to use the `NLST` FTP command instead of the `LIST` command that the adapter uses by default.
`UseRemoteArchive`	Set this property to `true` to notify the Oracle FTP Adapter that the archival directory is on the same FTP server. If set to `false`, the Oracle FTP Adapter uses a local file system folder for archival.

Table A-3 Binding Properties for Oracle File and FTP Adapters

Property	Description
`IgnoreListingErrors`	Lets you control the behavior of the inbound Oracle File Adapter during the polling operation. If set to `true`, the Oracle File Adapter does not complain if it cannot read from a nested folder.
`IgnoreZeroByteFile`	Set it to `true` if you do not want Oracle File and FTP Adapters to throw an exception during the outbound read operation, if the file could not be found. This property is ignored if the schema for the inbound file is anything other than `Opaque`.
`NestedFolderLimit`	Limits the number of levels the File/FTP adapter recurses into input directory to poll for files
`InMemoryTranslation`	This property is applicable only if `UseStaging` is set to `false` .If `UseStaging` is set to `true`, then the translation step occurs in-memory (that is, an in-memory byte array is created). If set to `false`, then the adapter creates an output stream to the target file (FTP, FTPS, and SFTP included) and allows the translator to translate and write directly to the stream.
`jca.message.encoding`	This property is used to override the encoding specified in the NXSD schema for inbound Oracle File and FTP Adapters.
`NotifyEachBatchFailure`	Setting to `true` causes the Oracle File or FTP Adapter to call the Notification Agent's `onBatchFailure` every time an error occurs in a debatching scenario. If set to `false`, Oracle File and FTP Adapter call `onBatchFailure` only once after all messages are debatched.
`oracle.tip.adapter.file.debatching.rejection.quantum`	This property lets you control the size of rejected messages for inbound Oracle File and FTP Adapter partner link. For example, if you set it to `100`, it causes the Oracke File or FTP Adapter to reject 100 lines from the file because the actual file is too large.
`oracle.tip.adapter.file.highavailability.maxRetry`	Number of times that inbound Oracle File and FTP Adapters retry to establish a database connection in distributed polling scenarios.
`oracle.tip.adapter.file.highavailability.maxRetryInterval`	Number of milliseconds after which inbound Oracle File and FTP Adapters retry to establish a database connection in distributed polling scenarios.
`oracle.tip.adapter.file.mutex`	Set it to the class name that specifies the mutex for the outbound write operation. This class must extend the `oracle.tip.adapter.file.Mutex` abstraction.
`oracle.tip.adapter.file.rejectOriginalContent`	Setting to `true` causes Oracle File or FTP Adapter to reject the original content. If set to `false`, the adapter rejects the XML data created because of the translation step.
`oracle.tip.adapter.file.timeout.recoverpicked.minutes`	This property is used by the inbound highly available adapter when using `FILEADAPTER_IN` as the coordinator. Remember that when a file is first claimed (enqueued) by a node for processing, the `FILE_PROCESSED` column in `FILEADAPTER_IN` is set to `0`. At a later point in time, when a decoupled processor threads picks up the file for processing, the value of the `FILE_PROCESSED` column is updated from `0` to `1`. And when the file is processed completely, the `FILE_PROCESSED` column is updated from `1` to `2`. However, if the processor thread picks up a file but the node crashes before processing the file, then the file is never processed. This property is used to *undo* the pick operation. The adapter does this by deleting the entries in the `FILEADAPTER_IN` table that have been picked up but not processed within the value specified here.

Table A-3 (Cont.) Binding Properties for Oracle File and FTP Adapters

Property	Description
`oracle.tip.adapter.file.timeout.recoverunpicked.minutes`	This property is used by the inbound highly available adapter when using `FILEADAPTER_IN` as the coordinator. Remember that when a file is first claimed by a node for processing, `FILE_PROCESSED` column in `FILEADAPTER_IN` is set to `0`. Later on, when the decoupled-processor thread picks up the file for processing, the value of the `FILE_PROCESSED` column is updated from `0` to `1`. And when the file is processed completely, the `FILE_PROCESSED` column is updated from `1` to `2`. If the node crashes when the `FILE_PROCESSED` is still `0`, it would mean that the file is enqueued by a node (no other nodes can pick this one up). However, it also means that the decoupled processor threads have still not picked this one for processing. This property is used to *undo* the claim(`enqueue_ operation`.) The adapter does this by deleting the entries in the `FILEADAPTER_IN` table that have been claimed (for example, `FILE_PROCESSED =="0"`), but not picked up till now.
`RecoveryInterval`	This property is used by the inbound adapter to configure the recovery interval in case of errors. For example, if the physical directory is nonexistent, then the adapter uses this value to perform periodic sleep or wakeup checks to check whether the physical directory has been created and is accessible.
`SerializeTranslation`	If set to `true`, then the translation step is serialized using a semaphore. The number of permits for semaphore (guarding the translation step) comes from the `ConcurrentThreshold` property. If `false`, then the translation step occurs outside the semaphore.
`UseFileSystem`	This property is used by inbound Oracle File and FTP Adapters during read-only polling in a clustered environment. Setting it to `true` causes the adapter to use the file system to store the metadata of the processed files. Setting it to `false` causes the adapter to use a database table.

Table A-4 Binding Properties Specific to Oracle FTP Adapter

Property	Description
`TimestampOffset`	This property is used by the Oracle FTP Adapter to handle time zone issues, typically to convert the time difference between the FTP server and the system on which the Oracle FTP Adapter is running to millisecond.
`control.so.timeout`	This property allows you the define time-out for the control socket to the FTP server. The parameter should specified in milliseconds.

Table A-5 JCA Properties for Oracle File Adapter: Normalized Properties

Property	Description
`jca.file.FileName`	This property specifies the name of the file read from the inbound directory or written to the outbound directory.
`jca.file.Directory`	This property specifies the name of the directory from which file is read from or written to.
`jca.file.Size`	This property specifies the size of the file published from the inbound Oracle File Adapter.
`jca.file.Batch`	This property is used to specify a unique identifier for the file being published from the inbound adapter.

Table A-5 (Cont.) JCA Properties for Oracle File Adapter: Normalized Properties

Property	Description
`jca.file.BatchIndex`	If a file has multiple messages and de-batching is used, then this normalized property specifies the message (record) number from the same batch. In this case, the `jca.file.Batch` remains the same but `jca.file.BatchIndex` increments by one for each publish from the same batch.
`jca.file.IsEOF`	This property is used with chunked interaction. It is used to find out if the end of file is reached.

Table A-6 JCA Properties for Oracle FTP Adapter: Normalized Properties

Property	Description
`jca.ftp.FileName`	This property specifies the name of the file read from the inbound directory or written to the outbound directory.
`jca.ftp.Directory`	This property specifies the name of the directory from which file is read from or written to.
`jca.ftp.Size`	This property specifies the size of the file published from the inbound Oracle File Adapter. The size can be zero.
`jca.ftp.Batch`	This property is used to specify a unique identifier for the file being published from the inbound Oracle FTP Adapter.
`jca.ftp.BatchIndex`	If a file has multiple messages and de-batching is used, then this normalized property specifies the message (record) number from the match batch. In this case, the `jca.ftp.Batch` remains the same but `jca.ftp.BatchIndex` increments by one for each publish from the same batch.

Oracle Socket Adapter Properties

This section describes the properties applicable to the Oracle Socket Adapter, including:

- Table A-7

For properties applicable to all Oracle JCA Adapters, see Generic Oracle JCA Adapter Properties.

For more information, see Section 33.1.2.7, "Oracle Socket Adapter", in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Table A-7 JCA Properties for Oracle Socket Adapter

Property	Description
`ByteOrder`	Byte order of the remote computer being communicated with.
`CustomImpl`	If `CustomImpl` is chosen as the `TransMode` property, then it is the name of the Java class defining the handshake. This property is a concrete implementation of the ICustomParser interface.
`Encoding`	Character encoding used by the remote computer.

Table A-7 (Cont.) JCA Properties for Oracle Socket Adapter

Property	Description
`Host`	In case of outbound, the computer name on which the socket server is running, to which you want to connect. In case of inbound, it is always `localhost`.
`Port`	In case of outbound, it is the port number on which a socket server is running, to which the adapter is connecting. In case of inbound, it is the port number on which the socket adapter listens for incoming connections.
`ReplyXslt`	If `XSLT` is chosen as the `TransMode` property, then it specifies the path to the style sheet defining the handshake for inbound reply.
`TransMode`	Mechanism for defining the protocol. Set to `XSLT` to use style sheets, set to `CustomImpl` to use custom Java code, and set to `NXSD` for plain schema translation. Set to `Script` for javascript handshake base protocol.
`Xslt`	If `XSLT` is chosen as the `TransMode` property, then it specifies the path to the style sheet defining the handshake for inbound request, in case of inbound and outbound request or reply.
`RequestScript`	Service side only. Name of the javascript file used for handling the request side communication. Use when `TransMode` is set to script.
`ReplyScript`	Service side only. Name of the javascript file used for handling the response side communication. Use when `TransMode` is set to script.
`OutboundScript`	Reference side only. Name of the javascript file used for outbound communication. Use when `TransMode` is set to script.
`SupportNIO`	By default this property is set to `true` (when it is not specified). NIO is supported only for the inbound direction and only for NXSD mode of translation. For other modes of translation such as XSLT, CustomImpl and Script, SupportNIO is set to `false`. Additionally, NIO is supported only for non-SSL connections.

Oracle AQ Adapter Properties

This section describes the properties applicable to the Oracle AQ Adapter, including:

- Table A-8
- Table A-9
- Table A-10

For properties applicable to all Oracle JCA Adapters, see Generic Oracle JCA Adapter Properties.

For more information, see Section 33.1.2.1, "Oracle AQ Adapter", in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Table A-8 JCA Properties for Oracle AQ Adapter

Property	Description
`QueueName`	The name of the AQ Queue being read from or written to.

Table A-8 (Cont.) JCA Properties for Oracle AQ Adapter

Property	Description
`DatabaseSchema`	The schema where the queue resides. If not specified, the schema of the current connection is used.
`SchemaValidation`	When this property is set to `"true"`, the payload is validated against the schema specified. If the validation fails, then the message is rejected.
`EnableStreaming`	When this property is set to `"true"`, the payload from the queue is streamed instead of being processed as an in-memory DOM. You must use this feature while handling large payloads. This property is applicable when processing RAW messages, XMLType messages, and ADT type messages for which a payload is specified through an ADT attribute.
`RecipientList`	Specify the consumer name or names that are the intended recipients for the messages enqueued by the adapter. The message remains in the queue until all recipients have dequeued the message. If the field is left empty, then all the currently active consumers are recipients.
`Consumer`	Applicable only for multiconsumer queues. If specified, only the messages targeted for the particular consumer are made available for processing.
`ObjectFieldName`	This property is used to identify the field containing the business payload if the queue is an ADT queue. You can specify an attribute of ADT to constitute a payload or an entire ADT to represent payload. In former case the 'ObjectFieldName' should be same as the attribute name of the ADT. In latter case this property is not specified.
`PayloadHeaderRequired`	Only applicable if the `ObjectFieldName` property specifies a value. If set to `"true"`, it ensures that all non payload attributes of ADT are available for processing as Normalized Message property `'jca.aq.HeaderDocument'`.
`MessageSelectorRule`	When a dequeue is performed from a multiconsumer queue, it is sometimes necessary to screen the messages and accept only those that meet certain conditions. These conditions can be based on payload or queue header values and is specified using `MessageSelectorRule` property of the adapter. These conditions may concern selecting messages of a certain priority, or some aspect of the message payload, such as in selecting only loan applications above $100,000.
`DequeueCondition`	This property is valid for dequeue operations only. Enter a Boolean expression similar to the `WHERE` clause of a SQL query. This expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions. If multiple messages satisfy the dequeue condition, then the order of dequeuing is indeterminate, and the sort order of the queue is not honored.
`Correlation`	You can assign an identifier to each message, thus providing a means to retrieve specific messages at a later time. The value to enter is agreed upon between the enqueuing sender and the dequeuing receiver for asynchronous conversations. This can be overridden on a per message basis through the normalized message property, `'jca.aq.Correlation'`. When specified for the dequeue operation, it only dequeues messages that match the value specified. If none is specified, all messages in the queue are available to the dequeue operation.
`PayloadSizeThreshold`	This property exposes a configurable control mechanism through which you can specify the payload size threshold in the adapter layer. The messages that have sizes beyond the configured threshold limit are rejected. If this property is not configured, it does not impose any restriction on the size of messages.

Table A-9 JCA Properties for Oracle AQ Adapter: Normalized Properties

Property	Description
`jca.aq.Attempts`	The number of failed attempts at dequeuing the message.
`jca.aq.Correlation`	User-assigned correlation ID.
`jca.aq.Delay`	The number of seconds after which the message is available for dequeuing.
`jca.aq.EnqueueTime`	The time at which the message was enqueued.
`jca.aq.ExceptionQueue`	The exception queue name.
`jca.aq.Expiration`	The number of seconds before the message expires. This parameter is an offset from the `Delay` parameter. Default value of `-1(never expires)` is used if not specified.
`jca.aq.MessageId`	The hexadecimal representation of the message ID for the dequeued message.
`jca.aq.OrigMessageId`	The hexadecimal representation of the original message ID.
`jca.aq.Priority`	Priority of the message. A smaller number indicates a higher priority. The priority can be any number. The default value is zero.
`jca.aq.RecipientList`	The list of recipients for this message, separated by commas. This overrides `RecipientList` in `InteractionSpec`.
`jca.aq.HeaderDocument`	Contains string or DOM of current headers (XML DOM representation of payload headers.)

Table A-10 Binding Properties for Oracle AQ Adapter

Property	Description
`ConnectionRetryDelay`	The time for which the Oracle AQ Adapter waits before trying to re-create a connection after a connection is lost. The default value is 15s.
`DequeueTimeOut`	It is the interval after which the `dequeue()` API times out if no message is received on the inbound queue. The default value is 1s.
`adapter.aq.dequeue.threads`	Specifies the number of poller threads that are created when an endpoint is activated. The default value is 1.

Oracle JMS Adapter Properties

This section describes the properties applicable to the Oracle JMS Adapter, including:

- Table A-11
- Table A-12
- Table A-13

For properties applicable to all Oracle JCA Adapters, see Generic Oracle JCA Adapter Properties.

For more information, see Section 33.1.2.5, "Oracle JMS Adapter", in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*

Table A-11 JCA Properties for Oracle JMS Adapter

Property	Description
`DestinationName`	The name of the queue or topic being read from or written to.
`UseMessageListener`	Deprecated. The only value of this property supported in the current version is false. A value of `false` ensures that the JMS adapter uses synchronous mechanism to poll queues or topics for messages
`PayloadType`	This property specifies the type of JMS message that is being dequeued or enqueued by the adapter. For Map messages the value is `'MapMessage'` and for Text messages the value is `'TextMessage'`.
`DurableSubscriber`	Name used to identify a durable subscription. When working with durable subscriptions ensure that ClientID is also specified in addition to `DurableSubscriber` property. `ClientID` is specified as part of the `FactoryProperties` property when defining a Managed Connection Factory instance. This property is only applicable when working with JMS Topic scenarios
`MessageSelector`	A string whose syntax is based on a subset of the SQL92 conditional expression syntax and lets you specify the messages adapter is interested in, by using header field references and property references. Only messages whose header and property values match the selector are delivered.
PayloadEntry	Only applicable when dealing with messages of type `MapMessage`. This property is used to identify the field containing the business payload when dealing with Map messages. All other Map message entries are made available as normalized message properties accessed using `jca.jms.Map.`*`MapMessage entry name`*
AttachmentList	Only applicable when dealing with message of type `MapMessage`. This property is used to identify the field containing the business payload when dealing with Map messages. Also, the payload in this case is published as an attachment. You can use either `PayloadEntry` or `AttachmentList`. All other Map message entries are made available as normalized message properties accessed using `jca.jms.Map.`*`MapMessage entry name`*
`RequestDestinationName`	This property is applicable for a synchronous request-reply scenario and specify the name of destination for sending a message.
`ReplyDestinationName`	This property is applicable for a synchronous request-reply scenario and specify the name of destination for receiving a reply.
`AllowTemporaryReplyDestination`	This property is applicable for a synchronous request-reply scenario. When set to `true` `ReplyDestinationName` is not required and JMS adapter in turn uses a temporary destination to recieve a reply from.
`EnableStreaming`	When this property is set to `"true"`, the payload from the queue or topic is streamed instead of being processed as an in-memory DOM. You must use this feature while handling large payloads.

Table A-11 (Cont.) JCA Properties for Oracle JMS Adapter

Property	Description
`DeliveryMode`	Represents the delivery mode to use. The message producer's default delivery mode is `PERSISTENT`. This can be overridden on a per message basis using normalized message property `jca.JMSDeliveryMode`
`TimeToLive`	Represents the message's lifetime (in milliseconds). The message producer's default time to live is unlimited; the message never expires. A value of `0` signifies that the message never expires.
`PayloadSizeThreshold`	This property exposes a configurable control mechanism through which you can specify the payload size threshold in the adapter layer. The messages that have sizes beyond the configured threshold limit are rejected. If this property is not configured, it does not impose any restriction on the size of messages
`Priority`	Represents priority for this message. The message producer's default priority is 4. This can be overridden on a per message basis using normalized message property `jca.jms.JMSPriority`
`UnitOfOrder`	Specifies the use of the Unit-of-Order feature. The WebLogic Server Unit-of-Order feature enables a message producer or group of message producers acting as one, to group messages into a single unit that is processed sequentially in the order the messages were created.

Table A-12 JCA Properties for Oracle JMS Adapter: Normalized Properties

Property	Description
`jca.jms.JMSDestinationName`	This property specifies the destination to which the message is sent, and is set by the JMS producer.
`jca.jms.JMSDestinationProperties`	This property represents the properties that define the context used to look up the destination object to which the message must be sent
`jca.jms.JMSCorrelationID`	This property is set by both producers and consumers for linking the response message with the request message. This is an optional attribute.
`jca.jms.JMSType`	This property specifies the JMS message type.
`jca.jms.JMSReplyTo`	This is an optional attribute that indicates the destination to which a message reply must be sent.
`jca.jms.JMSPriority`	This property is used by the consumer to set a priority number between 0 and 9. Larger numbers represent a higher priority.
`jca.jms.JMSExpiration`	This property specifies the duration of the message before the expiration. When a message's expiration time is reached, the JMS provider should discard it.
`jca.jms.JMSDeliveryMode`	This property is set to `persistent` or `nonpersistent` mode by the JMS client.

Table A-12 (Cont.) JCA Properties for Oracle JMS Adapter: Normalized Properties

Property	Description
`jca.jms.JMSMessageID`	This property is used to specify a unique message identifier. The exact scope of uniqueness is provider-defined.
`jca.jms.JMSRedelivered`	This property is used as an indication of whether a message is being re-delivered. If a client receives a message with the JMSRedelivered field set, it is likely, but not guaranteed, that this message was delivered earlier but that its receipt was not acknowledged at that time.
`jca.jms.JMSTimestamp`	This property is used to specify the time when the message was handed off to the JMS provider to be sent.
`jca.jms.JMSProperty.`*`name`*	This property represents any custom (application-specific) properties of the message. The supported properties conforms to the one allowed according to JMS specification. If an invalid property value is specified, the adapter warns the user (captured in the log files) and ignores the invalid property.
`jca.jms.Map.`*`name`*	This property represents any MapMessage element that is not transferred as payload.
`jca.jms.WeblogicUnitOfOrder`	This value overrides the value specified using the property `UnitOfOrder` for the `JmsProduceInteractionSpec`.
`jca.jms.JMSProperty.JMS_BEA_UnitOfOrder`	Use this Normalized Property to get the Unit of order value set on the Inbound JMS Message in the receive activity of BPEL.

Table A-13 Binding Properties for Oracle JMS Adapter

Property	Description
`adapter.jms.encoding`	Used to encode inbound text messages. This property is superseded by the newly supported property called `jca.message.encoding` that is applicable for both inbound and outbound messages.

Table A-13 (Cont.) Binding Properties for Oracle JMS Adapter

Property	Description
`adapter.jms.DistributedDestinationConnectionEveryMember`	This property takes boolean values [`true` | `false`]. Default value is `true`. When true, adapter will create a consumer/subscriber for each member of the Distributed Destinations. If set to false, adapter will create consumer/subscriber for only local members of the distributed destination.When the adapter is connecting to distributed destination on a remote cluster or a server on remote domain, property `adapter.jms.DistributedDestinationConnectionEveryMember` should always be set to *true*. When the adapter is connecting to distributed destination on a local cluster, the property can be set to either *true* or *false*. If set to *true*, the adapter behavior remains the same as before (consumer for each Distributed Destination is created). If set to *false*, adapter will now only create consumer/subscriber for the local members. You do not see the property in the JDeveloper User Interface as s binding property on the Service side. You must manually add the property in the composite.xml as below under the <service> /<binding.jca> tag : `<property name="adapter.jms.DistributedDestinationConnectionEveryMember" type="xs:string" many="false" override="may">false</property>`
`adapter.jms.receive.threads`	Specifies the number of poller threads that are created when an endpoint is activated. The default is 1.
`adapter.jms.receive.timeout`	Timeout value used for the synchronous receive call. It is the time after which receive() API times out if no message is received on the inbound queue. The default value is 1s.
`adapter.jms.registration.interval`	This property is not supported anymore.
`adapter.jms.retry.interval`	Used by the inbound connection retry layer. The time for which the Oracle JMS Adapter waits before trying to re-create a connection after a connection is lost. The default value is 30s.
`JMSReplyToDestinationProperties`	Declaratively impose custom property settings on Destination objects received during inbound request/reply scenarios.
`JMSReplyUseCorrelationIdForCorrelation`	Used to specify whether you want to use a correlation Id for correlation. Values are `true` and `false`. The default value is `false`.
`JMSReplyUseMessageIdForCorrelation`	Used to specify whether message Id for correlation. Values are `true` and `false`. The default value if none is specified is false.
`JMSReplyPropagateJMSExpiration`	The boolean property specifies if the reply message TTL is set to 0 (message never expires) or some specified value related to message expiration. The default value is false.

Table A-13 (Cont.) Binding Properties for Oracle JMS Adapter

Property	Description
`RequestReply.cacheReceivers`	If the same small number of JMS receivers are used for the same request destination repeatedly, then set this property to `true` to improve performance. The default value if none is specified is false.
`RequestReply.useCorrelation`	Applicable for a synchronous request-reply scenario. If set to `true`, then it applies a JMS Message selector for obtaining the reply message. If the request Normalized Message property, `jca.jms.JMSCorrelationID` is specified, then it is used for correlation, otherwise the JMS Message ID property is used. The JMS adapter uses the following message selector: `"JMSCorrelationID = '<corrId>' [ AND (<wsdlSelector>) ]"` where the AND branch is only included if the user specifies a `MessageSelector` property. The default value is `true`.
`SuppressHeaders`	Used to bypass headers. For scenarios in which a composite does not use or produce the headers, the value of `true` can be used. It may improve performance for such scenarios. The default value is `false`.

Oracle Database Adapter Properties

This section describes the properties applicable to the Oracle Database Adapter, including:

- Table A-14
- Table A-15

For properties applicable to all Oracle JCA Adapters, see Generic Oracle JCA Adapter Properties.

For more information, see:

- Section 33.1.2.2, "Oracle Database Adapter", in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.
- Appendix H, "Normalized Message Properties", in the *Developing SOA Applications with Oracle SOA Suite*.

Table A-14 JCA Properties for Oracle Database Adapter: Instance Properties

Property	Description
`DataSourceName`	Either this property or xADataSource name is a mandatory property, or both. Refers to the JNDI name (`jdbc/...`) of the `tx-level="local"` data source connecting to. All operations using this pool are locally transacted, independent on the global transaction. If both `xADataSourceName` and `dataSourceName` are specified, then the latter is used for `READ` operations.
`LogTopLinkAll`	The default value is `FALSE`. You must increase DB Adapter logging to include all underlying `TopLink` log messages at maximum granularity. This property provides maximum visibility, but adapter logging is tuned to show the most relevant `TopLink` SQL logging.

Table A-14 (Cont.) JCA Properties for Oracle Database Adapter: Instance Properties

Property	Description
`PlatformClassName`	This is a mandatory property. This points to the type of database being connected to. The suggested values for this property are: • Oracle11Platform • Oracle10Platform • Oracle9Platform • Oracle8Platform • DB2Platform • InformixPlatform • SybasePlatform • SQLServerPlatform • MySQLPlatform • DatabasePlatform You also can give the full package and class name of a subclass of `oracle.toplink.platform.database.DatabasePlatform`. For DB2 on AS/400, Oracle recommends that you give the value of `oracle.tip.adapter.db.toplinkext.DB2AS400Platform`.
`UsesBatchWriting`	The default value is `TRUE`. Multiple identical statements are executed as a single batch statement. You must only disable this property for certain JDBC drivers that have known issues.
`UsesSkipLocking`	The default value is `TRUE`. Oracle Database polling statements using `SELECT FOR UPDATE` are automatically be upgraded to `SELECT FOR UPDATE SKIP LOCKED`, which provides better concurrent performance. Disable this property only for certain cases where skip locking is not compatible with another feature or your use case.
`UsesNativeSequencing`	The default value is `TRUE`. If any SOA services are configured to automatically assign sequence numbers on `INSERT` operation, then a `TRUE` value indicates that the sequence values are coming from a database native sequence.
XADataSourceName	This is a mandatory property. It specifies the JNDI name (`jdbc/...`) of the `tx-level="global"` data source connecting to the database. All operations using this pool bind to the global transaction and commit or roll back as a unit.

Table A-15 JCA Properties for Oracle Database Adapter: Normalized Message Properties

Property	Description
`jca.db.CursorName`	Inbound/Outbound.
`jca.db.DataSourceName`	Outbound.
`jca.db.Password`	Outbound. You cannot assign values to the `jca.db.password` property on Oracle Containers for Java EE because its data source does not support setting password dynamically to the `getConnection` method. Consider using `jca.db.ProxyPassword` instead.
`jca.db.ProxyCertificate`	Outbound. When set, specifies `OracleConnection.PROXYTYPE_CERTIFICATE` as the proxy type. The value is a `base64Binary` encoded `byte[]` array that contains the certificate. If set, do not set `jca.db.ProxyDistinguishedName`, `jca.db.ProxyUser`, and `jca.db.ProxyPassword`.

Table A-15 (Cont.) JCA Properties for Oracle Database Adapter: Normalized Message Properties

Property	Description
`jca.db.ProxyDistinguishedName`[1]	Outbound. When set, specifies `OracleConnection.PROXYTYPE_DISTINGUISHED_NAME` as the proxy type. The value should be the proxy distinguished name as a `java.lang.String`. If set, then set none of `jca.db.ProxyCertificate`, `jca.db.ProxyUserName` and `jca.db.ProxyPassword`.
`jca.db.ProxyIsThickDriver`[1]	Outbound. Values are `true` and `false`. Set to `true` if using the OCI driver, as there is some discrepancy in the JDBC-level API between the thick and thin drivers.
`jca.db.ProxyPassword`[1]	Outbound. When set, specifies `OracleConnection.PROXYTYPE_USER_PASSWORD` as the proxy type. The value should be the password for the proxy user as a `java.lang.String`. If set, you must also set `jca.db.ProxyUserName`. If set, then set neither `jca.db.ProxyCertificate` nor `jca.db.ProxyDistinguished` name.
`jca.db.ProxyRoles`[1]	Outbound. Set to define the roles associated with the proxy user. The value should be a `String[]` array where each `java.lang.String` corresponds to a role name. This property is applicable when using any of `OracleConnection.PROXYTYPE_USER_PASSWORD`, `OracleConnection.PROXYTYPE_CERTIFICATE`, and `OracleConnection.PROXYTYPE_DISTINGUISHED_NAME`.
`jca.db.ProxyUserName`[1]	Outbound. When set, specifies `OracleConnection.PROXYTYPE_USER_PASSWORD` as the proxy type. The value should be the user name of the proxy user as a `java.lang.String`. If set, you must also set `jca.db.ProxyPassword`. If set, then set neither `jca.db.ProxyCertificate` nor `jca.db.ProxyDistinguished` name.
`jca.db.UserName`	Outbound. You cannot assign values to the `jca.db.userName` property on Oracle Containers for Java EE because its data source does not support setting user name dynamically to the `getConnection` method. Consider using `jca.db.ProxyUserName` instead.
`jca.db.XADataSourceName`	Outbound.

[1] For more information, see Proxy Authentication Support.

Oracle MQ Series Adapter Properties

This section describes the properties applicable to the Oracle MQ Series Adapter, including:

- Table A-16
- Table A-17
- Table A-18
- Table A-19

For properties applicable to all Oracle JCA Adapters, see Generic Oracle JCA Adapter Properties.

For more information, see Section 33.1.2.6, "Oracle MQ Adapter", in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Table A-16 JCA Properties for Oracle MQ Series Adapter

Property	Description
`QueueName`	This property specifies the name of the MQ Queue for sending or retrieving messages.
`MessageType`	This property specifies the type of message: Normal, Request, Reply, or Report
`MessageFormat`	This property specifies the type of MQ message format, such as Default, and Request/Reply.
`Priority`	This property specifies the message priority. Its value ranges from `0` to `9`. The default value is `AS_Q_DEF`, which uses the value defined in the destination queue.
`Persistence`	This property is used to set the message persistence. The message persists when this property is set to `true`. The defaults value is `AS_Q_DEF`, which uses the value defined in the destination queue.
`OnDeliveryFailure`	This property is used when message delivery fails. The default value is `DeadLetterQueue`, which sends messages to a dead letter queue. If the value is set to `DISCARD`, then messages are discarded.
`PartialDeliveryForDL`	This property is used for partial delivery to a distribution list. The default value is `false`.
`SegmentIfRequired`	This property is used when the size of the message is larger than the maximum limit set on the queue. The default value is `true`.
`Expiry`	This property specifies the time after which the message would be removed by the Queue Manager. The default value is `NEVER`.
`ReplyToQueueName`	This property specifies the name of the queue to which the reply or report must be sent.
`ReportCOA`	If this property is set, a confirmation on arrival report is sent to the `replyto` queue on arrival of a message in the destination queue. The default value is `WITH_NO_DATA`, and no data is sent in this case.
`ReportCOD`	If this property is set, a confirmation on delivery report is sent to the replyto queue on arrival of a message in the destination queue. The default value is `WITH_NO_DATA`, and no data is sent in this case.
`ReportException`	If this property is set, an exception report is sent to the replyto queue when message delivery to the destination queue fails. The default value is `WITH_NO_DATA`, and no data is sent in this case.
`ReportExpiry`	If this property is set, an expiry report is sent to the replyto queue when a message sent to the destination queue expires. The default value is `WITH_NO_DATA`, and no data is sent in this case.
`WaitInterval`	This property specifies the waiting interval for dequeuing the message in outbound MQ queue.
`MessageId`	This property is used to generate a Message Id for a reply or a report message. By default a new Message Id is generated.

Table A-16 (Cont.) JCA Properties for Oracle MQ Series Adapter

Property	Description
`CorrelationId`	This property is used to generate a correlation Id for a reply or a report message. By default the message Id of the request message is used as the correlation Id.
`QueueOpenOptions`	This property specifies the queue open options to use while accessing the queue.
`SecondaryQueueManagerName`	This property specifies the queue manager for the enqueue queue. Use this property only when the outbound enqueue queue resides outside the inbound queue manager.
`BackoutQueue`	This property is used to specify a backout queue to which rejected messages from an inbound queue are to be sent.
`BackoutQueueManagerName`	This property is used to specify the queue manager for the backout queue. Use this property only when the Backout Queue resides outside the inbound queue manager.
`MaximumBackoutCount`	This property is used to specify the maximum backout retry count after which rejected message is sent to the backout queue.
`BackoutInterval`	This property is used to specify intervel between the backout retries. The default value is 5 seconds.
`BackoutRetries`	This property is used to specify the number of backout retries. The default value is 3.
`FallbackReplyToQueueName`	This property is used for sending the report to the Normal Message Queue.
`FallbackReplyToQueueManagerName`	This property is used when the Primary Queue Manager specified in JNDI connection cannot access the queue.
`DistributionList`	This property is used to specify the elements of the distribution list for enqueuing the message.
`MessageSelector`	This property specifies which messages must be retrieved from the MQ queue. Only messages whose headers and properties match the selection criteria are retrieved.
`AsAttachment`	This property, set to true, configures the adapter to process all messages as attachment.
`CharacterSet`	This property is optional and should only be specified if the "AsAttachment" property is set to true. It defines the character set of the attachment and is meant for third-party applications that process the attachments published by the Oracle MQ Series Adapter.
`Encoding`	This property is optional and should only be specified if the "AsAttachment" property is set to true. It defines the encoding of the attachment and is meant for third-party applications that process the attachments published by the Oracle MQ Series Adapter.
`ContentType`	This property is optional and should only be specified if the "AsAttachment" property is set to true. It defines the content type of the attachment and is only meant for third-party applications that process the attachments published by the Oracle MQ Series Adapter.

Table A-16 (Cont.) JCA Properties for Oracle MQ Series Adapter

Property	Description
`MaximumSegmentLength`	This property is optional and a value should only be provided if you want to segment the message into a custom size. The value should be in the following format - integerValue[M/K/B], where M stands for Megabytes, K for Kilobytes and B for Bytes. For example, 230K, 4M, 512B.
`ServerHeartbeat`	This property defines the amount of time in milliseconds which the adapter should wait before trying to reconnect once the MQ Queue Manager is down.
`ExponentialBackOff`	This property should always be used in conjunction with the `MaxExponentialBackOffCount` property. This property (in seconds) defines the base of the exponentially increasing time interval after which the adapter tries to reconnect to the queue manager. For example, if the value is 2, the adapter will try to reconnect at intervals of 2 seconds, 4seconds, 8 seconds, 16 seconds.
`MaxExponentialBackOffCount`	This property should always be used in conjunction with the "ExponentialBackOff" property. This property is used for capping the "ExponentialBackOff" time interval. For example, if the "ExponentialBackOff" is set to 2 and "MaxExponentialBackOffCount" to 7 then the adapter retries at intervals of 2, 4, 8, 16, 32, 64, 128, 128, 128… seconds.
`UseDynamicResponseQueue`	The boolean property can be used to specify whether dynamic response queues should be used rather than a permanent ReplyToQueue. If set to true, then `ResponseModelQueueName` should always be specified.
`ResponseModelQueueName`	The model queue name based on which the dynamic response queues should be created for retrieving the reply message.
`ResponseWaitInterval`	This property specifies the wait interval for which the adapter waits for the response message to arrive on the reply queue.
`LimitHeaderLength`	This property is used to reduce the MQ header information. The property can be added to `oracle.tip.adapter.mq.inbound.ActivationSpecImpl` in the corresponding MQ adapter JCA file. The default value is false.
`PollingFrequency`	This parameter specifies how often the inbound MQ Adapter must poll messages on the MQ queue. The parameter is of type `int` in milliseconds. The default is 3000 milliseconds if the parameter is not specified.

Table A-17 JCA Properties for Oracle MQ Series Adapter: Normalized Properties

Property	Description
`jca.mqAccountingToken`	Inbound/Outbound. Accounting token information of the message. A hexadecimal-encoded string.
`jca.mq.ApplIdentityData`	Inbound/Outbound. Provides additional information about the Identity of the message or its originator. Accepts any string.
`jca.mq.ISpec.ResponseWaitInterval`	Outbound. Wait interval for the response message to arrive on the reply queue. Accepts integer value in miliseconds.

Table A-17 (Cont.) JCA Properties for Oracle MQ Series Adapter: Normalized Properties

Property	Description
jca.mq.MQMD.ApplOriginData	Inbound/Outbound. Provides additional information about the origin of this message. Accepts any string.
jca.mq.MQMD.BackoutCount	Inbound/Outbound.Count of the number of times the message has previously been returned by an MQQueue.get() call as part of a unit of work, and subsequently backed out. Accepts zero and positive integer values.
jca.mq.MQMD.CorrelId	Inbound/Outbound. Correlation identifier of the message to be retrieved/ to be put. Accepts a hexadecimal-encoded string.
jca.mq.MQMD.Encoding.Decimal	Inbound/Outbound. Representation used for numeric values in the application message data. Accepts NORMAL and REVERSED.
jca.mq.MQMD.Encoding.Float	Inbound/Outbound. representation used for numeric values in the application message data. Accepts NORMAL, REVERSED and S390
jca.mq.MQMD.Expiry	Inbound/Outbound. A message's expiry time has elapsed, and it is eligible to be discarded by the queue manager. Accepts NEVER or a non- Inbound/ Outbound.negative integer value
jca.mq.MQMD.Feedback	Inbound/Outbound. Used with a message of type MQC.MQMT_REPORT to indicate the nature of the report. Accepts any string.
jca.mq.MQMD.Feedback.ApplicationDefined	Inbound/Outbound. application defined feedback.Accepts any string.
jca.mq.MQMD.Format	Inbound/Outbound. Format name used by the sender of the message to indicate the nature of the data in the message to the receiver.Accepts following formats NONE, ADMIN, CHANNEL_COMPLETED, CICS, CMD1, CMD2, DEAD_LETTER_HDR, DIST_HDR, EVENT, IMS, IMS_VAR_STRING, MD_EXTN, PCF, REF_MSG_HDR, RF_HDR_1, RF_HDR_2, STRING, TRIGGER, WORK_INFO_HDR, XMIT_Q_HDR
jca.mq.MQMD.GroupId	Inbound/Outbound. Byte string that identifies the message group to which the physical message belongs. Accepts hexadecimal-encoded string.

Table A-17 (Cont.) JCA Properties for Oracle MQ Series Adapter: Normalized Properties

Property	Description
jca.mq.MQMD.MsgFlags.IsMsgInGroup	Inbound/Outbound. Specifies if the message belongs to a group. Accepts true, false.
jca.mq.MQMD.MsgFlags.IsLastMsgInGroup	Inbound/Outbound. Specifies if the message is the last message of the group. Accepts true, false
jca.mq.MQMD.MsgFlags.IsSegment	Inbound/Outbound. Specifies if the message is a segment. Accepts true, false
jca.mq.MQMD.MsgFlags.IsLastSegment	Inbound/Outbound. Specifies if message is the last segment. Accepts true, false
jca.mq.MQMD.MsgId	Inbound/Outbound. Message identifier of the message to be retrieved/ to be put. Accepts hexadecimal encoded string
jca.mq.MQMD.MsgSeqNumber	Inbound/Outbound. Sequence number of a logical message within a group. Accepts non- Inbound/Outbound.negative integer value
jca.mq.MQMD.MsgType	Inbound/Outbound. Indicates the type of the message. Accepts any string.
jca.mq.MQMD.MsgType.ApplicationDefined	Inbound/Outbound. Application -defined message type. Accepts any string.
jca.mq.MQMD.Offset	Inbound/Outbound. The offset of data in a physical message from the start of a logical message. Accepts non- Inbound/ Outbound.Negative integer value.
jca.mq.MQMD.OriginalLength	Inbound/Outbound. Original length of a segmented message. Accepts non-Negative integer value Inbound/Outbound
jca.mq.MQMD.Persistence	Inbound/Outbound. message persistence. Accepts true, false, AS_Q_DEF
jca.mq.MQMD.Priority	Inbound/Outbound. Message priority. Accepts 0- Inbound/Outbound.9, AS_Q_DEF
jca.mq.MQMD.PutApplName	Inbound/Outbound. Name of the application that Put the message. Accepts any string
jca.mq.MQMD.PutApplType	Inbound/Outbound. Type of application that Put the message. Accepts any string.
jca.mq.MQMD.PutApplType.UserDefined	Inbound/Outbound. User-defined Put application type. Accepts any string.

Table A-17 (Cont.) JCA Properties for Oracle MQ Series Adapter: Normalized Properties

Property	Description
`jca.mq.MQMD.PutDateTime`	Inbound/Outbound. Time and date that the message was Put. Accepts year:month:date, year:month:date:hour:minute, year:month:date:hour:minute:second
`jca.mq.MQMD.ReplyToQMgr`	*I*nbound/Outbound. Name of the queue manager to which reply or report messages should be sent. Accepts any sting.
`jca.mq.MQMD.ReplyToQ`	Inbound/Outbound. Name of the queue to which reply or report messages should be sent. Accepts any string
`jca.mq.MQMD.Report.Generate.CorrelId`	Inbound/Outbound. scheme to generate the CorrelationId of reply or report message. Accepts PASS_CORREL_ID, COPY_MSG_ID
`jca.mq.MQMD.Report.Generate.MsgId`	Inbound/Outbound. Scheme to generate the MessageId of reply or report message. Accepts NEW_MSG_ID, PASS_MSG_ID
`jca.mq.MQMD.Report.Generate.COA`	Inbound/Outbound. Specifies the content of COA report. Accepts WITH_NO_DATA, WITH_PARTIAL_DATA, WITH_FULL_DATA
`jca.mq.MQMD.Report.Generate.COD`	Inbound/Outbound. Specifies the content of COD report. Accepts WITH_NO_DATA, WITH_PARTIAL_DATA, WITH_FULL_DATA
`jca.mq.MQMD.Report.Generate.Exception`	Inbound/Outbound. Specifies the content of the Exception report. Accepts WITH_NO_DATA, WITH_PARTIAL_DATA, WITH_FULL_DATA
`jca.mq.MQMD.Report.Generate.Expiry`	Inbound/Outbound. Specifies the content of the Expiry report. Accepts WITH_NO_DATA, WITH_PARTIAL_DATA, WITH_FULL_DATA
`jca.mq.MQMD.Report.Generate.NAN`	Inbound/Outbound. Specifies if the incoming/outgoing message is NAN or not. Accepts true, false
`jca.mq.MQMD.Report.Generate.PAN`	Inbound/Outbound. specify if incoming/ outgoing message is PAN or not. Accepts true, false
`jca.mq.MQMD.Report.TakeAction.OnMsgDeliveryFailure`	Inbound/Outbound.

Table A-17 (Cont.) JCA Properties for Oracle MQ Series Adapter: Normalized Properties

Property	Description
jca.mq.MQMD.Report.TakeAction.OnMsgDeliveryFailure	Accepts DISCARD, DEADLETTERQUEUE
jca.mq.MQMD.StrucId	Inbound/Outbound. Struct id of MQMD. Accepts any string
jca.mq.MQMD.UserIdentifier	Inbound/Outbound. User who originated this message. Accepts any string.
jca.mq.MQMD.Version	Inbound/Outbound. Version of MQMD. Accepts VERSION_1, VERSION_2
jca.mq.Inbound.MQMD.CorrelId	Outbound. Correlation identifier of the message retrieved in Async req-reply scenario. Accepts hexadecimal encoded string
jca.mq.Inbound.MQMD.MsgId	Outbound. Message identifier of the messageretrieved in Async req-reply scenario. Accepts hexadecimal-encoded string.
jca.mq.Inbound.MQMD.MsgType	Outbound. Message Type of the message retrieved in Async req-reply scenario. Accepts any string.
jca.mq.Inbound.MQMD.Nan	Outbound. NAN report option of the message retrieved in the Async req-reply scenario. Aceepts true, false
jca.mq.Inbound.MQMD.Pan	Outbound. PAN report option of the message retrieved in the Async req-reply scenario. Accepts true, false
jca.mq.Inbound.MQMD.ReplyToQMgr	Outbound. ReplyToQueueManager of the message retrieved in the Async req-reply scenario. Accepts any string.
jca.mq.Inbound.MQMD.ReplyToQ	Outbound. ReplyToQueue of the message retrieved in the Async req-reply scenario. Accepts any string.
jca.mq.Inbound.MQMD.Report.Generate.CorrelId	Outbound. Correlation scheme, for generation of CorrelationId, of the message retrieved in Async req-reply scenario. Accepts PASS_CORREL_ID, COPY_MSG_ID
jca.mq.Inbound.MQMD.Report.Generate.MsgId	Outbound. Correlation scheme, for generation of MessageId, of the message retrieved in Async req-reply scenario. Accepts NEW_MSG_ID, PASS_MSG_ID
jca.mq.ISpec.EnqueueMsgToQMgr	Outbound. Queue Manager for outbound queue. Accepts any string
jca.mq.ISpec.EnqueueMsgToQ	Outbound. Queue name of outbound queue. Accepts any string.

Table A-18 Connection Properties for Oracle MQ Series Adapter

Property	Description
HostNam	Name of the host computer.
PortNumber	Port number to be used.
ChannelName	Set it to the server connection channel to be used.
QueueManagerName	A valid queue manager name.
CipherSuite	Set CipherSuite to the name matching the CipherSpec set on the SVRCONN channel. If set to null (default), then no SSL encryption is performed.
ClientEncoding	Character encoding used by the client.
ConnectionFactoryLocation	Location of the connection factory.
HostOSType	Operating system used by the host computer.
KeyStoreAlgorithm	Algorithm used by the key store.
KeyStoreLocation	This value is the keystore where Oracle MQ Series Adapter has its private keys. This is required when an adapter must authenticate itself to the MQ Series server.
KeyStorePassword	This value is the password that is required to access keystore.
KeyStoreProviderName	The name of the keystore provider.
TrustStoreLocation	This is the location where the adapter keeps its trusted certificates information. This information is required when an adapter must authenticate to the MQ Series server.
TrustStorePassword	This property specifies the password of the Trust Store location.
KeyStoreType	This property specifies the type of the key store.
Protocol	Key Management Algorithm.
SSLPeerName	A distinguished name pattern. If CipherSuite is set, you can use this variable to ensure that the correct queue manager is used. If set to null (default), then the DN of the queue manager is not checked. This variable is ignored if sslCipherSuite is null.
SSLEnable	The true or false value for this property means that the Oracle MQ Series Adapter is SSL enabled or SSL disabled.
UserID	This property is used if credential mapping is not set.
Password	This is the password to connect to the queue manager. This property is used if credential mapping is not set.
XATransaction	This property is used to enable or disable XA transactions. If set to true, then XA transaction is enabled.
ReceiveExit	This is the Receive Exit java class, which gets triggered when you receive a message on a particular queue.

Table A-18 (Cont.) Connection Properties for Oracle MQ Series Adapter

Property	Description
SecurityExit	This is the `Security Exit` java class, which enables you to customize the security flows that occur when an attempt is made to connect to a queue manager.
`SendExit`	This is the `Send Exit` java class, which is triggered when you send a message to a particular queue.
`CCDTurl`	The value is a path to the CCDT file which should be used by the MQ Series Adapter for client connection details.
`KeyStoreName`	The name of the key store under the given stripe. Used when KeyStoreType is set to "kss".
`StripeNameForKeyStore`	The name of the stripe within which key store is created. Used when KeyStoreType is set to `kss`.
`StripeNameForTrustStore`	The name of the stripe within which trust store is created. Used when TrustStoreType is set to `kss`
`TrustStoreAlgorithm`	Algorithm used by the trust store.
`TrustStoreName`	The name of the trust store under the given stripe. Used when TrustStoreType is set to kss.
`TrustStoreProviderName`	The name of the trust store provider.
`TrustStoreType`	This property specifies the type of the trust store.

Table A-19 Binding Properties for Oracle MQ Series Adapter

Property	Description
`adapter.mq.inbound.queueName`	This property is used to specify the name of the inbound MQ queue.
`adapter.mq.inbound.binaryNulls`	This property is used for dequeuing the messages with binary zero value. The default value for this property is `true`.

LDAP Adapter Properties

This section describes the properties applicable to the LDAP Adapter.

- Table A-20
- Table A-21

Table A-20 LDAP Adapter Connection Properties

Property	Allowed Values	Description
`HostName`	Name or IP address of the host machine.	Machine where the ldap server is hosted.

Table A-20 (Cont.) LDAP Adapter Connection Properties

Property	Allowed Values	Description
BindDN	Cn=admin	
InboundDataSource	JDBC datasource JNDI where RCU schema is installed. This data store is used to persist the last change number. This property value is useful for last incremental synchronization scenario. (It is required to use if the LDAP adapter must be used on the inbound direction).	JNDI string location pointing to a valid XA data source.
OperationTimeout		Sets the operation timeout. If the response is not received by the Directory Server in the timeout period, the operation will be abandoned and an exception will be raised.
Password	secret	Not required
SecurityProvider		
SSLContextProtocol		Key management protocol, not required.
TrustAll	False	If mentioned, the client will blindly trust any certificated that the server might present. Required only if useSSL or startTLS is selected.
TrustStoreProviderName		The jsse provider name.
P	Integer value	Port where the LDAP service is running on the host.
UseSSL	True/False with default value as false.	Indicates that LDAP adapter should use SSL to secure communication with the directory server. The server must be configured to listen in SSL mode and the value for the port argument must be one where the server is listening for SSL based connections.
UseStartTLS	True/False with default value as false. If useSSL is true, then this must be false.	Use start TLS extended operation to secure communication with a directory server over a non-encrypted channel. Port in this case is listening for clear-text LDAP Connections.

Table A-20 (Cont.) LDAP Adapter Connection Properties

Property	Allowed Values	Description
`TrustAll`	True/false with default value as false.	If mentioned, the client will blindly trust any certificated that the server might present. TrustStorePath is redundant and ignored if this option is selected.
`TrustStorePath`	Must be mentioned if one of useSSL or useStartTLS is set as true.	Specifies path to the JKS trust store file that the client use for server authentication.
`TrustStorePassword`		Specifies the password needed to access the trust store contents. In most cases, no trust store password is required. This should not be used in conjunction with the trust store password file option.
`Protocol`		Set the protocol version enabled for secure connections with the directory server. The protocol must be supported by the SSL Context.
`EnableCipherSuites`	A String. Uses the default set of SSL cipher suites provided by the server's JVM. List of comma separated cipher suites allowed by the jsse provider.	Specifies the names of the SSL cipher suites that are allowed for use in SSL or StartTLS communication.

The following table lists LDAP Adapter Binding properties. Note that some of the jca properties are applicable on multiple operations. For example, `TimeLimit` can be used with `EntryChangeNotification`, `ChangelogNotification` or search operation. But this same property has the same meaning across these operations.

Table A-21 LDAP Adapter Binding Properties (Common Across All Adapters)

Property Name	Allowed	Description
Jca.retry.count		Indicates maximum number of retries before rejection.
Jca.retry.interval	Measured in Seconds	Indicates time interval between retries.
Jca.retry.backoff	Positive Integer	Indicates the retry interval growth factor.
Jca.retry.maxInterval		Indicates maximum value of retry interval
Jca.retry.maxPeriod		Is the upper limit of the entire accumulated duration of all retries; that is, a hard limit..

Table A-21 (Cont.) LDAP Adapter Binding Properties (Common Across All Adapters)

Property Name	Allowed	Description
Jca.retry.count		Indicates maximum number of retries before rejection.
Jca.retry.interval	Measured in Seconds	Indicates time interval between retries
Jca.retry.backoff	Positive Integer	Indicates the retry interval growth factor
Jca.retry.maxInterval		Indicates max value of retry interval
Jca.retry.count		Indicates maximum number of retries before rejection.

Table A-22 LDAP Adapter JCA Artifacts for Entry Change Notification

Property Name	Allowed Values	Description
BaseDN	Valid DN under which events should be reported.	Notify events under this entry.
EventScope	Restricted string with value in: { "baseObject", "singleLevel", "subordinates", "wholeSubtree"}	Scope of event source under the configured basedn.
EventType	Value in Unknown macro: {"add", "modify", "all" }	Unknown macro: {"add", "modify", "all" } If all is selected, then all the events will be published
SearchFilter	Valid string representation of LDAP filter	Advanced filter condition. Events that satisfy the given filter condition only will be published by the adapter.
ReturnAttributes	Csv list of all the attributes to return as part of the event.	-
ReturnAttributesDelimiter	String	Attributes delimiter can be explicitly defined in case the default delimiter ',' is not appropriate.
TypesOnly	"true" or "false" with default value as "false"	When set to true, only the attribute names will be returned. Otherwise, both attribute names and values will be returned.

Table A-22 (Cont.) LDAP Adapter JCA Artifacts for Entry Change Notification

Property Name	Allowed Values	Description
SizeLimit	+ve integer. With default value of 1000. <= 0 implies no sizeLimit will be enforced.	Maximum number of entries returned as part of a search op. This can be configured on the DS side as well. Lower of the two values will take effect. SizeLimit is enforced within a single page.
PageSize	Integer value specifying a pageSize. Note, `sizeLimit` implies a size within a page and does not pertain to the total number of entries returned across all the pages in a search. Default value of -1 implying unlimited pageSize.	Max number of events that should be published in a page.
TimeLimit	Integer value in seconds with default value as 10 seconds.	Maximum time the server should wait before returning the results.
NotificationStrategy	String value: one of "TimeStampBased"	Persistent search option has been deprecated in the first release.
PollingInterval	Integer with default value set to 10 seconds.	Polling interval before conducting subsequent searches for new events.

Table A-23 LDAP Adapter JCA Artifacts for Change Log Notification

Property Name	Allowed Values	Description
PollingInterval	Integer with default value set to 10 seconds.	Polling interval before conducting subsequent searches for new events.
EventType	Value in Unknown macro: {"add", "delete", "modify", moddn", "all"} Note that eventType value might vary from Directory Server implementation. For instance, moddn might be modrdn in the case of oid.	If all is selected, all events are published, Otherwise a comma-separated list of event types to publish will be sent. For example: "add,delete"
NotificationStrategy	String value in Unknown macro: {"OJD", "OID" or fully qualified name of the user defined changelog implementation.}	Unknown macro: {"OJD", "OID" or "fully qualified name of the user defined changelog implementation} Oracle.tip.adapter.ldap.inbound.<customStrategy Impl> .

Table A-23 (Cont.) LDAP Adapter JCA Artifacts for Change Log Notification

Property Name	Allowed Values	Description
TimeLimit	Integer value in seconds with default value as 0 for no time limit.	Maximum time the server should wait before returning the results.
SizeLimit	Integer value with default value as 1000.	Maximum number of events that should be published in one trip.
PageSize	Integer value specifying a pageSize. Note that sizeLimit implies within a page and not on the total number of entries returned across all the pages in a search.	-

Table A-24 LDAP Adapter JCA Artifacts for Search Operation

Property Name	Allowed Values	Description
SearchScope	Restricted string with value in: { "baseObject", "singleLevel", "subordinates", "wholeSubtree"} - Subordinates is not a part of the LDAP specification and is only supported by selected DS. Check your DS server before using this option.	Defines the search scope. This has to be defined at service deployment to control malignant search request from controlling server resources.
AliasDereferencing	Restricted string with value in: Unknown macro: { "never","search", "find", "always"}	Behavior for handling alias entries while procssing the search.
FollowReferrals	Restricted string with value in: Unknown macro: {"ignore","throwException","follow"} By default, ldap adapter ignores all referral suggestions unless explicitly asked to follow or throw exception.	Alternate locations where additional entries may be searched/ matched as part of a search request.
HopLimit	Positive integer. Default value of 1.	Number of alternate locations participating in search referrals.
SizeLimit	Integer with default value as 1000.	Maximum number of entries returned as part of a search op. This can be configured on the DS side as well. Lower of the two values will take effect.

Table A-24 (Cont.) LDAP Adapter JCA Artifacts for Search Operation

Property Name	Allowed Values	Description
TimeLimit	Integer value in seconds with default value as 0 for no time limit.	Maximum time in seconds the server should wait before returning the results.
TypesOnly	"true" or "false" with default value as "false"	When set to true, only the attribute names will be returned. Otherwise, both attribute names and values will be returned.
ReturnAttributes	String value, delimited list of all the attributes that must be returned as part of this search response.	The default delimiter is ',' Only names attributes will be included as part of search response.
ReturnAttributesDelimiter	String	attributes delimiter can be explicitly defined in case the default delimiter ',' is not appropriate

Table A-25 LDAP Adapter DSML JCA Artifacts

Property Name	Allowed Values	Description
MaxDSMLRequestSize	Integer value with default value set to 0 for unlimited.	DSML contains a batch request. This batch request can potentially contain millions of ldap operation requests. This property is used to control the maximum number of permissible operation requests that can be passed to the ldap adapter through a single dsml batch request.

Table A-26 LDAP Adapter Delete Operation JCA Artifacts

Property	Allowed values	Description
Request Controls	String	To pass in one or more controls along with the operation.
SuppressEntryDoesNotExist	True | false	A delete operation fails if the entry to delete doesn't exist in the target DS. This exception can be suppressed by specifying this condition on the interaction spec.

Table A-27 LDAP Adapter Add Operation jca Artifacts

Property	Allowed values	Description
SuppressEntryAlreadyExists	True | false	Add operation fails if the entry to add already exist in the target DS. This exception can be suppressed by specifying this property on the interaction spec.
objectClass	CSV String	Comma-separated list of all objectClasses that constitute the structure of the entry to be added in the target directory server.

Table A-28 LDAP Adapter Modify Operation jca Artifacts

Property	Allowed values	Description
Request Controls	String	To pass in one or more controls along with the operation.

Table A-29 LDAP Adapter ModifyDN Operation jca Artifacts

Property	Allowed values	Description
RequestControls	String	To pass in one more controls along with the operation.
DeleteOldRDN	N "true" or "false" with default value as "true"	-

Coherence Adapter Properties

This section lists Coherence Adapter Properties.

- Table A-30
- Table A-31
- Table A-32

The following table lists Coherence Adapter JCA Properties

Table A-30 Coherence Adapter JCA Properties

Property	Description	Default Value
CacheName	Identifies a particular Coherence cache to be used for a particular operation. A non-retriable error occurs when the Cache as specified by the entry `CacheName` in any Coherence Adapter operation does not exist.	None
Key	Identifier for a Cache Entry.	None
KeyType	Identifies the java type of the value to be used as the key (currently any simple type from java.lang package).	None

Table A-30 (Cont.) Coherence Adapter JCA Properties

Property	Description	Default Value
ValueType	Fully qualified class name that identifies the object stored to/ retreived from cache. When you indicates to store XML into cache, you do not need to specify ValueType property. If not specified, the Coherence Adapter will convert the payload to a binary XML representation and store/ retreive it to/from cache.	None
Filter	Used to specify the subset of cache that the operation should be applied to. The filter expression is based on CohQL and support use of bind variables.	None
PageSize	Specifies the number of items to be returned when a query operation is executed. It is only applicable for cache operations associated with remote caches	None
TimeToLive	The number of milliseconds until the cache entry will expire. It is only applicable for cache operations associated with non-transactional caches.	`EXPIRY_NEVER`
IndexName	The name of the index to be added to the cache. To create an index on the attribute 'price' in the cache that stores Book (author.firstName, author.LastName, price) objects, `IndexName` should be set to value `price`.For index on the attribute firstName, `IndexName` should be set to `author.firstName`	None
Ordered	Boolean that specified whether contents of indexed information is ordered or not	False
MappingsMetadataFile	Indicates the name of the Metadata mapping file that helps in POJO to. XML and XML to POJO conversion	None

Table A-30 (Cont.) Coherence Adapter JCA Properties

Property	Description	Default Value
CacheOperation	The operation to be carried out on the Cache identified by CacheName. The supported values for CacheOperation are 'get','put','query' and 'remove'.	None
ReturnCacheKeysOnly	Boolean that allows for only Cache identifiers (keys) to be returned when a Query operation is performed	False

The following table lists Coherence Adapter Normalized properties.

Table A-31 Coherence Adapter Normalized Properties

Property	Description
jca.coherence.CacheName	Name of the Cache to be used for Coherence operations. This overrides 'CacheName' in CoherenceInteractionSpec
jca.coherence.Key	Identifies a cache entry. This overrides 'Key' in CoherenceInteractionSpec.
jca.coherence.Filter	Specify the filter to be used when performing Coherence operations. This overrides 'Filter' in CoherenceInteractionSpec.

The following table lists Coherence Adapter Connection Factory Properties

Table A-32 Coherence Adapter Connection Factory Properties

Name	Description	Value
`ConnectionFactoryLocation`	Specifies the JNDI name of connection factory provided by coherence-transaction.rar	`tangosol.coherenceTx`
`ServiceName`	Name of the cache service to be associated with the connection. ServiceName is only required for TransactionalCaches. For the non-transaction/extend case it is not required.	XATransactionalCache - for XA TransactionalCache - for Local cache Leave blank for Remote cache
`CacheConfigLocation`	Location of cache configuration file that defines the cache and extend client settings	-

Table A-32 (Cont.) Coherence Adapter Connection Factory Properties

Name	Description	Value
`ClassLoaderMode`	Specifies if the classloader used for cache operations is based on the domain classes as bundled with the composite or based on the PojoJarFile property of the JCA managed connection factory. Values are CUSTOM or COMPONENT. If CUSTOM, the association is based on the PojoJarFile property. If COMPONENT, it is based on the domain classes bundled along with the composite. CUSTOM is used when the same cache needs to be used across composite boundaries or from different OSB processes. COMPONENT is used when all cache operations are embedded within the same composite or OSB process.	CUSTOM
`WLSExtendProxy`	Specifies if the Extend Proxy is a standalone or running as part of WLS server. If false, it is a standalone Extend Proxy; otherwise not. Applicable only for the Remote Case.	`True`
`PojoJarFile`	Specifies the location of domain classes that will be used for the cache operations.	-

MSMQ JCA Adapter Properties

This section describes MSMQ JCA Adapter properties.

- Table A-33
- Table A-34
- Table A-35
- Table A-36

Table A-33 MSMQ Adapter JCA Properties

Property	Description	Default Value	Required

`DestinationType`	Indicates if the message will be sent to a public queue, private queue or a group of queues as identified by the distribution list name. The values are:
	`PUBLIC_QUEUE`
	`PRIVATE_QUEUE`
	`DISTRIBUTION_LIST`
		None	Yes

`DestinationName`	Name of the MSMQ queue.	None	Yes, if UseActiveDirectoryPath is False
`DestinationPath`	The string that identifies a DistributionList or Public queue as represented in ActiveDirectory. An example string is listed below; LDAP://MyLDAPServer/CN=MyQueue,CN=msmq,CN=MyComputer,CN=Computers,DC=MyDomain,DC=MyCompany,DC=COM	None	Yes, if UseActiveDirectoryPath is True
`UseActiveDirectoryPath`	Boolean that allows for Active Directory Path to be used to identify a public queue instead of queue name. This property is applicable when DestinationType is '`DISTRIBUTION_LIST`' or '`PUBLIC_QUEUE`'	False	Applicable only to outbound(Enqueue) operation. This property is not applicable for inbound(Dequeue). Dequeue is not supported for Distribution lists.
`UseDirectFormatName`	Boolean that allows for Direct Format name to be used for public and private queues.	False	-
`Priority`	Priority can be set to any integer value between 7 and 0 (the default is 3). 7 means higher priority. 0 means lowest priority. Highest priority implies faster processing. Applicable to outbound Enqueue only	3	-

Table A-33 (Cont.) MSMQ Adapter JCA Properties

Property	Description	Default Value	Required
`TimeToLive`	This property specifies a time limit (in seconds) for the message to be retrieved from the target queue. The value is assigned to `MaxTimeToReceive` property for a given MSMQ message. Applicable to inbound and outbound Queue.	-1 (infinite) This implies that the message never expires.	-
`Delivery`	The property is used to specify express (non-persistent) or recoverable (persistent) messaging. Express messaging provides faster throughput. Recoverable messaging guarantees that the message will be delivered even if a computer crashes while the message is en route to the queue. The values are: Persistent, Non-Persistent, Applicable to outbound Enqueue only	Persistent	-
`OperationType`	The operation to be carried out. The supported values for OperationType are `enqueue` and `dequeue`.	None	Yes
`BodyType`	Values are `String` (default) and `ByteArray`. When opaque (schema) processing option is selected that would imply BodyType value of ByteArray.	String	Yes
`EnableStreaming`	Boolean that enables payload to be streamed. Applicable only for inbound dequeue process	False	-

The following table lists MSMQ Adapter Normalized Properties.

Table A-34 MSMQ Adapter Normalized Properties

Property Name	Description	Direction
`jca.msmq.message.SentTime`	The property indicates when a message is sent.	Inbound

Table A-34 (Cont.) MSMQ Adapter Normalized Properties

Property Name	Description	Direction
`jca.msmq.message.Priority`	The property specifies a message's priority. This overrides `Priority` in the MSMQInteractionSpec.	Inbound/Outbound
`jca.msmq.message.TimeToLive`	The property specifies a time limit (in seconds) for the message to be retrieved from the target queue. This overrides `TimeToLive` in the MSMQInteractionSpec	Inbound/Outbound
`jca.msmq.message.MaxTimeToReachQueue`	The property specifies a time limit (in seconds) for the message to reach the queue. Message Queuing uses the enterprise-wide setting for the time-to-reach-queue interval if none is specified here.	Inbound/Outbound
`jca.msmq.message.Id`	Identifies the message using an MSMQ-generated message identifier.	Inbound
`jca.msmq.message.Delivery`	Values can be `Persistent` or `NonPersistent`. The property specifies how Message Queuing delivers the message. This property overrides `Delivery` in MSMQInteractionSpec	Inbound/outbound
`jca.msmq.message.BodyLength`	The property provides the length of the message body in bytes.	Inbound
`jca.msmq.message.ArrivedTime`	The property indicates when the message arrived at the queue.	Inbound

The following table lists MSMQ Adapter Binding properties.

Table A-35 MSMQ Adapter Binding Properties

Property Name	Description
`adapter.msmq.receive.timeout`	The time (in milliseconds) that Message Queuing will wait for a message to arrive before starting another poll-cycle. The default value is one second (1).
`adapter.msmq.dequeue.threads`	Number of poller threads that will be initialized when endpoint activation occurs. When specified, the value of `adapter.msmq.dequeue.threads` is used to spawn multiple inbound poller threads; multiple inbound threads can be used to improve performance. The default value is 1.

The following table lists MSMQ Adapter Connection Factory Properties.

Table A-36 MSMQ Adapter Connection Factory Properties

Property Name	Description	Default
`Host`	IP Address of the MSMQ host.	This value can be the host name and the IP address can be given as the value for the Host property.
`AccessMode`	Identifies if the connection factory will allow for native access or not. Values are [`Native	DCOM`]. If Native, the Oracle WebLogic Server should be installed on the same host as MSMQ.	DCOM
`TransactionMode`	Indicates if the connection will participate in a transaction when sending and receiving a message. Values are [Single	None].If Single, the local and remote MSMQ queues to which the message is sent should be transactional. If Single, the local MSMQ queue from which the message is retrieved should be Transactional.If None and the queues to which the message is sent are transactional, there will be a ResourceException.	`None`
User	Identifies a user	-
Password	Password for the user specified.	-
`Domain`	Domain of the MSMQ host.	-

UMS JCA Adapter Properties

This section describes the properties applicable to the UMS Adapter.

- Table A-37
- Table A-38
- Table A-39
- Table A-40
- Table A-41

The following table lists UMS Adapter activation spec properties.

Table A-37 UMS Adapter Activation Spec Properties

Property Name	Description
JavaCalloutImpl	Name of the Java class that defines custom logic for a message filtering or any other check. This class is a concrete implementation of the ICustomCallout interface.
Name-InboundThreadCount	Specifies the number of inbound poller or listener threads.
ConsumeMode	Specifies how the adapter will receive messages from UMS. Set to poller for polling mode Or set to listener for listener mode.
To	Address from which to receive incoming messages. One or more comma separated email address for the Email delivery type.
DeliveryType	Email support is provided for receiving and sending outgoing messages.
PollingInterval	Polling interval in seconds for poller consume mode.
MessageFilters	Specify one or more message filters. A single filter would comprise of a Java Pattern String to match the incoming message against, along with the field type and the action (Accept or reject) to be taken.

The following table describes UMS Adapter Interaction Spec Properties.

Table A-38 UMS Adapter Interaction Spec Properties

Property Name	Description
DeliveryType	Email support for only receive and sending outgoing messages.
Subject	Subject of Outgoing Message
From	Sender addresses of outgoing message
To	One or more recipient addresses
Replyto	Reply-To address
Cc	One or more cc addresses for email delivery.
Bcc	One or more Bcc addresses for email delivery
SendEmailAsAttachment	True, to send email as an attachment

The following table describes UMS Adapter message headers.

Table A-39 UMS Adapter Message Headers

Header Field Name	Minimum Occurrence	Maximum Occurrence	Mapped Adapter Header Field Name
Return-path	0	1	jca.ums.return-path
Received	0	unlimited	jca.ums.received
Resent-Date	0	unlimited	jca.ums.resent-date
Resent-From	0	unlimited	jca.ums.resent-from
Resent-Sender	0	1	jca.ums.resent-sender
Resent-To	0	unlimited	jca.ums.resent-to
Resent-Cc	0	unlimited	jca.ums.resent-cc
Resent-Bcc	0	unlimited	jca.ums.resent-bcc
Resent-Message-ID	0	unlimited	jca.ums.resent-message-id
Date	1	1	jca.ums.date
From	1	1	jca.ums.from (applies to SMS and IM also)
Sender	0	1	jca.ums.sender
Reply-to	0	1	jca.ums.reply-to
To	0	1	jca.ums.to (applicable to SMS and IM channels)
Cc	0	1	jca.ums.cc
Bcc	0	1	jca.ums.bcc
Message-ID	0	1	jca.ums.message-id
In-Reply-To	0	1	jca.ums.in-reply-to
References	0	1	jca.ums.references

Table A-39 (Cont.) UMS Adapter Message Headers

Header Field Name	Minimum Occurrence	Maximum Occurrence	Mapped Adapter Header Field Name
`Subject`	0	1	`jca.ums.subject` (Applicable to SMS channel)
`Comments`	0	unlimited	`jca.ums.comments`
`Keywords`	0	unlimited	`jca.ums.keywords`

The following table describes UMS Adapter mime-part message headers.

Table A-40 UMS Adapter Mime-Part Message Headers

Header Field Name	Mapped Adapter Header Field Name	Notes
Content-Type	`jca.ums.msg.content-type`	Applicable to all channels. For SMS and IM, the current supported type is `text/plain` only. For voice it is `text/vxml`, `text/x-vxml`
Content-Transfer-Encoding	`jca.ums.msg.content-transfer-encoding`	-
Content-ID	`jca.ums.msg.content-id`	-
Content-Description	`jca.ums.msg.content-description`	-
Content-Disposition	`jca.ums.msg.content-disposition`	-
Content-Language	`jca.ums.msg.content-language`	-
Mime-Extension-field	`jca.ums.mime-extension-headers`	Any other mime header field hat begins with the string "Content-". You can add more than one header as `Content-* : value CRLF Content*- : value` `CRLF - \r\n`
MIME-Version	`jca.ums.mime-version`	-

The following table describes UMS Adapter proprietary headers.

Table A-41 UMS Adapter Proprietary Headers

Header Name	Description
`jca.ums.msg.proprietary-headers`	More than one proprietary header can be added in the following format: Header Name : value CRLF Header Name : value (Header Name – should be similar to ums.adapter.xxxxx CRLF - \r\n)

Generic Oracle JCA Adapter Properties

This section describes the properties applicable to all Oracle JCA Adapters, including:

- Table A-42

For properties specific to each of the Oracle JCA Adapters, see:

- Properties
- Properties
- Properties
- Properties
- Properties
- Properties

Table A-42 JCA Properties for all Oracle JCA Adapters

Property	Description

`WsdlLocation`	An optional `adapter-config` attribute of type `xs:string`.
	When set to the name of the WSDL associated with the adapter's JCA file, the Oracle Service Bus can automatically resolve the WSDL to allow bulk import of JCA files and related WSDL and schemas.
	The example below shows the `FulfillmentBatch_jms.jca` file with the `adapter-config` attribute `wsdlLocation` set to the name of the corresponding WSDL file `FulFillmentBatch.wsdl`.
	`<adapter-config name="FulfillmentBatch" adapter="Jms Adapter" xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata" wsdlLocation="FulfillmentBatch.wsdl" >`
	`...`
	`</adapter-config>`

Generic Oracle Adapter Binding Properties

This section describes the properties applicable to all Oracle JCA Adapters, including:

- Table A-43

For properties specific to each of the Oracle JCA Adapters, see:

- Properties

- Properties
- Properties
- Properties
- Properties
- Properties

Table A-43 Binding Properties for all Oracle JCA Adapters

Property	Description
`UseWorkManager`	Default is `False`. Supplies a custom work manager rather than the default work manager that is currently used with adapters. You can configure an inbound JCA endpoint (SCA service endpoint) to use a specific WebLogic WorkManager, using the binding property `useWorkManager`. The JCA binding delegates the WorkManager name to the SOA Infrastructure Executor, which accepts the work manager in its constructor. The value you enter into the `useWorkManager` property should be just the "name" part; the rest is added. For example, the name could be fileRead. If this JNDI lookup fails, the SOA Executor falls back to the default SOA work manager normally used if no work manager specified).
`MinimumDelayBetweenMessages`	Default is `False`, or no delay. Inbound-only. This property is configured in milliseconds. Ensures that there at least will be MILLI_SECONDS delay between two consecutive messages posted to the downstream composite application. Note that `minimumDelayBetweenMessages` is effective per adapter polling thread. If you have configured multiple adapter polling threads, this setting controls the delay between messages processed by each thread only.
`ActivationInstances`	Deprecated, because each adapter now natively supports multi threading. However, the property still works and controls how many instances of a particular inbound JCA activation agent will be spawned.The property can thus be used to increase concurrency. The activationInstances property can be dynamically updated by using the BPEL console (Descriptor page) and makes the adapter framework either spawn more instances or shutdown existing instances on the fly (in the shutdown case, in reverse order, that is, the highest instance number will be shutdown first).
`UseRejectedMessageRecovery`	If set to true, you can recover or abort a Rejected message.
`Singleton`	In a Clustered High Availability environment, setting this property to true ensures that only one instance is created at one point of time.
`EnableReports`	Set to true, enables Adapter Reports to be generated in Fusion Middleware Control for the configured Inbound Adapter.
`EnableSnapshots`	Set to true. enables Snapshot Reports to be generated in Fusion Middleware Control for the configured Inbound Adapter.
`SnapshotInterval`	The interval between snapshot reports.

B

Oracle JCA Adapter Valves

This appendix includes sample valves used by Oracle File and FTP Adapters. A valve is the primary component of execution in an FTP or File Adapter processing pipeline. A valve processes the content it receives and forwards the processed content to the next valve.

This chapter contains the following sections:

A Simple Unzip Valve

The following sample is a simple Unzip Valve:

```
package valves;

import java.io.*;
import java.util.zip.*;

import oracle.tip.pc.services.pipeline.AbstractValve;
import oracle.tip.pc.services.pipeline.InputStreamContext;
import oracle.tip.pc.services.pipeline.PipelineException;

/**
 * A simple valve to process zip files.
 * The valve processes the first entry from the zip file.
 * If you need to process multiple files, you will need
 * a re-entrant valve
 **/
public class SimpleUnzipValve extends AbstractValve {

      public InputStreamContext execute(InputStreamContext inputStreamContext)
                  throws IOException, PipelineException {
            // Get the input stream that is passed to the Valve
            InputStream originalInputStream = inputStreamContext.getInputStream();

            // Create a new ZIP input stream
            ZipInputStream zipStream = null;
            try {
                  zipStream = new ZipInputStream(originalInputStream);
                  ZipEntry entry = null;
                  // In this sample valve, lets pick up the first entry
                  if ((entry = zipStream.getNextEntry()) != null) {
                        System.out.println("Unzipping " + entry.getName());
```

```
                        ByteArrayOutputStream bos = new ByteArrayOutputStream();
                        byte[] buf = new byte[4096];
                        int len = 0;
                        while ((len = zipStream.read(buf)) > 0) {
                              bos.write(buf, 0, len);
                        }
                        bos.close(); // no-op but still ...
                        ByteArrayInputStream bin = new ByteArrayInputStream(bos
                                    .toByteArray());
                        // This is where the Valve returns the inputstream to the
caller
                        // Example, Adapter
                        // return the newly created inputstream as a part of the
context
                        inputStreamContext.setInputStream(bin);

                        return inputStreamContext;
                  }
            } finally {
                  if (zipStream != null) {
                        zipStream.close();
                  }
            }
            // return null if no data
            return null;

      }

      @Override
      // Not required for this simple valve
      public void finalize(InputStreamContext in) {

      }

      @Override
      // Not required for this simple valve
      public void cleanup() throws PipelineException, IOException {

      }


}
```

A Simple Decryption Valve That Uses Staging File

The following is a sample decryption valve that uses a staging file:

```
package valves;

import java.io.*;
import javax.crypto.*;
import javax.crypto.spec.*;

import oracle.tip.pc.services.pipeline.AbstractStagedValve;
import oracle.tip.pc.services.pipeline.InputStreamContext;
import oracle.tip.pc.services.pipeline.PipelineException;
import oracle.tip.pc.services.pipeline.PipelineUtils;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

/**
```

```
 * Simple Decryption valve that uses DES algorightm
 *
 * You must note that this class uses AbstractStagedValve. By using the
 * AbstractStagedValve, the valve notifies the pipeline that the valve will take
 * care of its own staging and cleanup
 *
 */
public class SimpleDecryptValve extends AbstractStagedValve {

      // Staging file where the intermediate decrypted content is kept
      private File stagingFile = null;

      /**
       * Called by the adapter. All the binding/reference properties in the
       * composite are available to the pipeline via the pipeline context For
       * example <service name="FlatStructureIn"> <interface.wsdl
       * interface="http://xmlns.oracle.com/pcbpel/adapter/file/FlatStructureIn/
#wsdl.interface(Read_ptt)"/>
       * <binding.jca config="FlatStructureIn_file.jca"> <property
       * name="myCipherKey" source="" type="xs:string" many="false"
       * override="may">somekey</property> </binding.jca> </service>
       *
       */
      public InputStreamContext execute(InputStreamContext inputStreamContext)
                  throws IOException, PipelineException {

            // Read the cipher key from the adapter binding property 'myCipherKey'
            String cipherKey = (String) getPipeline().getPipelineContext()
                        .getProperty("myCipherKey");

            // If key is blank, default to some hard-coded value
            if (PipelineUtils.isBlank(cipherKey)) {
                  System.out.println("using default ciper key");
                  cipherKey = "desvalve";
            }
            // Create an instance of the Cipher
            byte key[] = cipherKey.getBytes();
            SecretKeySpec secretKey = new SecretKeySpec(key, "DES");
            Cipher decrypt = null;
            try {
                  decrypt = Cipher.getInstance("DES/ECB/PKCS5Padding");
            } catch (NoSuchPaddingException nspe) {
                  throw new PipelineException("Unable to get cipher instance", nspe);
            } catch (NoSuchAlgorithmException nsae) {
                  throw new PipelineException("Invalid cipher algorithm", nsae);
            }
            try {
                  decrypt.init(Cipher.DECRYPT_MODE, secretKey);
            } catch (InvalidKeyException ike) {
                  throw new PipelineException("Invalid secret key", ike);
            }
            // original input stream from caller. For example, adapter
            InputStream originalInputStream = null;
            CipherInputStream cis = null;
            try {
                  originalInputStream = inputStreamContext.getInputStream();
                  cis = new CipherInputStream(originalInputStream, decrypt);
            } catch (Exception e) {
                  throw new PipelineException("Unable to create cipher stream", e);
            }
            // Since we're using a staged valve, we will store the decrypted content
```

```
                // in a staging file
                // In this case, we're leveraging the File/Ftp Adapter control directory
                // to store the content, but, the staging file can be placed anywhere
                this.stagingFile = PipelineUtils.getUniqueStagingFile(getPipeline()
                            .getPipelineContext().getStagingDirectory());

                // Write the decrypted content to the staging file
                OutputStream os = new FileOutputStream(this.stagingFile);
                byte[] b = new byte[8];
                int i = cis.read(b);
                while (i != -1) {
                      os.write(b, 0, i);
                      i = cis.read(b);
                }
                os.flush();
                os.close();
                cis.close();

                // Open a stream to the staging file and return it back to the caller
                InputStream in = new FileInputStream(this.stagingFile);
                // close the input stream passed in this invocation
                inputStreamContext.closeStream();
                // set the input stream to staging file and return
                inputStreamContext.setInputStream(in);
                return inputStreamContext;

          }

          /*
           * (non-Javadoc)
           *
           * @see oracle.tip.pc.services.pipeline.AbstractStagedValve#getStagingFile()
           */
          public File getStagingFile() {
                return stagingFile;
          }

          /*
           * Delete the staging file if there is one (non-Javadoc)
           *
           * @see
    oracle.tip.pc.services.pipeline.AbstractValve#finalize(oracle.tip.pc.services.pipelin
    e.InputStreamContext)
           */
          public void finalize(InputStreamContext ctx) {
                try {
                      cleanup();
                } catch (Exception e) {
                }
          }

          /*
           * Use this method to delete the staging file (non-Javadoc)
           *
           * @see oracle.tip.pc.services.pipeline.AbstractStagedValve#cleanup()
           */
          public void cleanup() throws PipelineException, IOException {

                if (stagingFile != null && this.stagingFile.exists()) {
                      this.stagingFile.delete();
                }
```

```
            this.stagingFile = null;
      }

}
```

A Valve for Encrypting Outbound Files

The following is a simple encryption valve that extends `AbstractValve`.

```
package valves;

import java.io.*;
import javax.crypto.*;
import javax.crypto.spec.*;

import oracle.tip.pc.services.pipeline.AbstractValve;
import oracle.tip.pc.services.pipeline.InputStreamContext;
import oracle.tip.pc.services.pipeline.PipelineException;
import oracle.tip.pc.services.pipeline.PipelineUtils;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

/**
 * Simple Encryption valve that uses DES algorightm
 *
 */
public class SimpleEncryptValve extends AbstractValve {


/**
      * Called by the adapter. All the binding/reference properties
      * in the composite are available to the pipeline via
      * the pipeline context
      * For example
      * <service name="FlatStructureOut">
      *     <interface.wsdl interface="http://xmlns.oracle.com/pcbpel/adapter/file/
FlatStructureOut/#wsdl.interface(Write_ptt)"/>
      *     <binding.jca config="FlatStructureOut_file.jca">
      *         <property name="myCipherKey" source="" type="xs:string" many="false"
override="may">somekey</property>
      *   </binding.jca>
      * </service>
      *
      */
     public InputStreamContext execute(InputStreamContext inputStreamContext)
               throws IOException, PipelineException {

          //Read the cipher key from the adapter binding property 'myCipherKey'
          String cipherKey = (String) getPipeline().getPipelineContext()
                    .getProperty("myCipherKey");

          //If key is blank, default to some hard-coded value
          if (PipelineUtils.isBlank(cipherKey)) {
               System.out.println("using default ciper key");
               cipherKey = "desvalve";
          }
          //Create an instance of the Cipher

pt.init(Cipher.ENCRYPT_MODE, secretKey);
          } catch (InvalidKeyException ike) {
               throw new PipelineException("Invalid secret key", ike);
```

```
            }
            //original input stream from caller. For example, adapter
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            try {

                 encryptStream(inputStreamContext.getInputStream(), bos, encrypt);
            } catch (Exception e) {
                 throw new PipelineException("Unable to encrypt", e);
            }
            byte[] bytes = bos.toByteArray();
            InputStream in = new ByteArrayInputStream(bytes);
            //close the input stream passed in this invocation
            inputStreamContext.closeStream();
            //set the input stream and return
            inputStreamContext.setInputStream(in);
            return inputStreamContext;

       }

       private static void encryptStream(InputStream in, OutputStream out, Cipher
encrypt) {
            try {
                 byte[] buf = new byte[4096];
                 // Bytes written to out will be encrypted
                 out = new CipherOutputStream(out, encrypt);

                 // Read in the cleartext bytes and write to out to encrypt
                 int numRead = 0;
                 while ((numRead = in.read(buf)) >= 0) {
                      out.write(buf, 0, numRead);
                 }
                 out.close();
            } catch (java.io.IOException e) {
            }
       }

       /*
        * Delete the staging file if there is one
        * (non-Javadoc)
        * @see
oracle.tip.pc.services.pipeline.AbstractValve#finalize(oracle.tip.pc.services.pipelin
e.InputStreamContext)
        */
       public void finalize(InputStreamContext ctx) {
            try {
                 cleanup();
            } catch (Exception e) {
            }
       }

       /*Use this method to delete the staging file
        * (non-Javadoc)
        * @see oracle.tip.pc.services.pipeline.AbstractStagedValve#cleanup()
        */
       public void cleanup() throws PipelineException, IOException {

       }

       public static void main(String[] args) throws Exception{
            String cipherKey = "desvalve";
```

```
            //Create an instance of the Cipher
            byte key[] = cipherKey.getBytes();
            SecretKeySpec secretKey = new SecretKeySpec(key, "DES");
            Cipher encrypt = null;
            try {
                 encrypt = Cipher.getInstance("DES/ECB/PKCS5Padding");
            } catch (NoSuchPaddingException nspe) {
                 throw new PipelineException("Unable to get cipher instance", nspe);
            } catch (NoSuchAlgorithmException nsae) {
                 throw new PipelineException("Invalid cipher algorithm", nsae);
            }
            try {
                 encrypt.init(Cipher.ENCRYPT_MODE, secretKey);
            } catch (InvalidKeyException ike) {
                 throw new PipelineException("Invalid secret key", ike);
            }
            //original input stream from caller. for example, adapter

            FileInputStream fin = new FileInputStream(args[0]);
            FileOutputStream fout = new FileOutputStream(args[1]);
            try {

                 encryptStream(fin, fout, encrypt);
            } catch (Exception e) {
                 throw new PipelineException("Unable to encrypt", e);
            }
            fin.close();
            fout.close();

       }

}
```

An Unzip Valve for processing Multiple Files

The following is the sample of an unzip valve for processing multiple files:

```
package valves;

import java.io.*;
import java.util.zip.*;
import java.util.*;

import oracle.tip.pc.services.pipeline.AbstractStagedValve;
import oracle.tip.pc.services.pipeline.InputStreamContext;
import oracle.tip.pc.services.pipeline.PipelineException;
import oracle.tip.pc.services.pipeline.PipelineUtils;

/**
 * A re-entrant valve is one that can be invoked multiple times
 * and on each invocation it must return a new stream.
 * This concept is used here in this sample to process
 * a zipped file containing multiple entries.
 *
 * If a valve is marked as re-entrant, then the caller (adapter),
 * calls hasNext() on the valve to check if there are more
 * streams available
 */
public class ReentrantUnzipValve extends AbstractStagedValve {

      //member variables
```

```
	private boolean initialized = false;

	private List<String> files = null;

	private File currentFile = null;

	private File unzipFolder = null;

	/**
	 * On the first invocation, this valve unzips the zip file into
	 * a staging area and returns a stream the first unzipped file
	 * On subsequent iterations, the valve returns streams to
	 * subsequent files.
	 */
	public InputStreamContext execute(InputStreamContext inputStreamContext)
			throws IOException, PipelineException {
		String fileName = "";
		//the first time that the valve is invoked, unzip the file into
		//the staging area
		if (!initialized) {
			files = new ArrayList<String>();
			//Get hold of the File/Ftp adapter control directory
			File controlDirectory = getPipeline().getPipelineContext()
					.getStagingDirectory();
			//Create if required
			if (!controlDirectory.exists()) {
				controlDirectory.mkdirs();
			}
			//Generate a unique folder to store the staging files
			String digestPath = "";
			try {
				digestPath = PipelineUtils.genDigest(inputStreamContext
						.getMessageOriginReference());
			} catch (Exception e) {
				digestPath = String.valueOf(inputStreamContext
						.getMessageOriginReference().hashCode());
			}
			unzipFolder = new File(controlDirectory, digestPath);
			if (!unzipFolder.exists())
				unzipFolder.mkdirs();
			//unzip the files into the staging folder
			unzipToDirectory(inputStreamContext.getInputStream(), unzipFolder);
			//store the file names into the list
			PipelineUtils.listFiles(unzipFolder, files);
			//close the input stream
			inputStreamContext.closeStream();
		}
		initialized = true;
		//return the next one in the list
		if (files != null && files.size() > 0) {
			fileName = files.remove(0);
			currentFile = new File(fileName);
			System.out.println("Returning file[" + fileName + "]");
			//Open a stream to the file and return to caller. For example,
adapter
			FileInputStream fis = new FileInputStream(currentFile);
			inputStreamContext.setInputStream(fis);
			/*For re-entrant valves, setting the message key is
			 important since this allows the caller to distinguish
			 between parts for the same message. for example, in the
			 case of zip file in this example, the
```

```
                     messageOriginReference will be same, but, the individual
                     message keys will vary. For example, the messageOriginReference
                     will be "/input/in.zip", whereas message key might be something
                     like "dir1/address-csv1.txt", "dir1/address-csv2.txt" and so on
                     */
                    inputStreamContext.setMessageKey(fileName);
                    return inputStreamContext;
              } else {
                    //return null if no more files
                    return null;
              }
        }

        /*
         * Adapter calls this to check if there are more files
         * @see oracle.tip.pc.services.pipeline.AbstractValve#hasNext()
         */
        public boolean hasNext() {
              return (files != null && files.size() > 0);
        }

        /*
         * Returns the current file being processed
         * @see oracle.tip.pc.services.pipeline.AbstractStagedValve#getStagingFile()
         */
        public File getStagingFile() {
              return currentFile;
        }

        /*
         * delete the current file once the entry has been published to binding
component
         * @see
oracle.tip.pc.services.pipeline.AbstractValve#finalize(oracle.tip.pc.services.pipelin
e.InputStreamContext)
         */
        public void finalize(InputStreamContext ctx) {

              if (currentFile != null && currentFile.exists()) {
                    currentFile.delete();
              }
        }

        /*
         * Cleanup intermediate files
         * @see oracle.tip.pc.services.pipeline.AbstractStagedValve#cleanup()
         */
        public void cleanup() throws PipelineException, IOException {
              PipelineUtils.deleteDirectory(unzipFolder);
              initialized = false;
              if (currentFile != null && currentFile.exists()) {
                    currentFile.delete();
              }
              files = null;
        }

        /*
         * Unzip to the directory
         */
        private void unzipToDirectory(InputStream in, File directory)
                    throws IOException {
```

```
            ZipInputStream zin = new ZipInputStream(in);
            ZipEntry entry = null;
            if ((entry = zin.getNextEntry()) != null) {
                  do {
                        String entryName = entry.getName();
                        if (!entry.isDirectory()) {
                              File file = new File(directory, entryName);
                              unzipFile(zin, file);
                        }
                  } while ((entry = zin.getNextEntry()) != null);
            }
            zin.close();
      }

      private void unzipFile(InputStream in, File file) throws IOException {
            if (!file.getParentFile().exists()) {
                  file.getParentFile().mkdirs();
            }
            OutputStream os = new FileOutputStream(file);
            byte[] buf = new byte[4096];
            int len = 0;
            while ((len = in.read(buf)) > 0) {
                  os.write(buf, 0, len);
            }
            os.close();
      }

}
```

C

Oracle MQ Series Adapter Supported Encodings

This appendix provides a list of supported MQ Series Adapter Supported Encodings..

By default, Oracle MQ Series Adapter supports a list of encodings. It displays a list of MQ Series message encodings and Java encoding, and also the mapping between the MQ Series message encoding and Java encoding. The list of supported encodings for Oracle MQ Series Adapter follows..

Oracle MQ Series Adapter Encodings

Following is a list of encodings.

- ibm037
- ibm437
- ibm500
- ibm819
- Unicode
- UTF8
- ibm273
- ibm277
- ibm278
- ibm280
- ibm284
- ibm285
- ibm297
- ibm420
- ibm424
- ibm737
- ibm775
- ibm813
- ibm838

- ibm850
- ibm852
- ibm855
- ibm856
- ibm857
- ibm860
- ibm861
- ibm862
- ibm863
- ibm864
- ibm866
- ibm868
- ibm869
- ibm870
- ibm871
- ibm874
- ibm875
- ibm912
- ibm913
- ibm914
- ibm915
- ibm916
- ibm918
- ibm920
- ibm921
- ibm922
- ibm930
- SJIS
- ibm933
- ibm935
- ibm937
- ibm939

- ibm942
- ibm948
- ibm949
- ibm950
- EUCJIS
- ibm964
- ibm970
- ibm1006
- ibm1025
- ibm1026
- ibm1089
- ibm1097
- ibm1098
- ibm1112
- ibm1122
- ibm1123
- ibm1124
- Cp1250
- Cp1251
- Cp1252
- Cp1253
- Cp1254
- Cp1255
- Cp1256
- Cp1257
- Cp1258
- ibm1381
- ibm1383
- JIS
- KSC5601
- ibm33722813
- GB18030

.

Adding Support for Other Standard Java Encodings

You can add support for the other standard Java encodings that are not provided in this list, as follows:

1. Extract the `MQSeriesAdapter.jar` file from the `MQSeriesAdapter.rar` file.
2. Extract the `mq.properties` file from the `MQSeriesAdapter.jar` file.
3. Add the entry in the `mq.properties` file. For each new encoding, you must add two lines (properties) to the `mq.properties` file. One line for the MQ Series encoding to the corresponding Java encoding and other line for the Java encoding to the corresponding MQ Series encoding.

   For example, to add support for the following `ibm037` Java encoding:ibm037 (Java encoding)<->37 (MQ Series message encoding), you must add the following two lines to the `mq.properties` file:

   ```
   oracle.tip.adapter.mq.encoding.37=ibm037
   ```

   ```
   oracle.tip.adapter.mq.encoding.ibm037=37
   ```