Oracle® Adapters for Files, FTP, Databases, and Enterprise Messaging

User's Guide

10*g* Release 2 (10.1.2.)

B25307-01

November 8, 2005

ORACLE®

Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User's Guide, 10*g* Release 2 (10.1.2.)

B25307-01

Primary Author: Deanna Bradshaw, Mark Kennedy, Craig West

Contributors: Oracle BPEL Process Manager development, product management, and quality assurance teams

Contents

Preface xiii

Audience xiii
Documentation Accessibility xiii
Related Documentation xiv
Conventions xiv

1 Introduction to Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging

Overview of Oracle BPEL Process Manager Technology Adapters 1-1
Summary 1-3

2 Oracle Application Server Adapter for Files/FTP

Introduction to the File and FTP Adapters 2-1
File and FTP Adapter Features 2-1
File and FTP Adapter Architecture 2-3
File and FTP Adapter Integration with Oracle BPEL Process Manager 2-3
File and FTP Adapter Concepts 2-4
File Adapter Read File Concepts 2-4
Inbound Operation 2-4
Inbound File Directory Specifications 2-5
Specifying Inbound Physical or Logical Directory Paths 2-5
Archiving Successfully Processed Files 2-6
File Matching and Batch Processing 2-7
Specifying a Naming Pattern 2-7
Including and Excluding Files 2-8
Batching Multiple Inbound Messages 2-10
File Polling 2-10
File Processing 2-11
Postprocessing 2-11
Native Data Translation 2-11
Error Handling 2-12
rejectedMessageHandlers Property 2-12
fatalErrorFailoverProcess Property 2-13
uniqueMessageSeparator Property 2-14
Default Error Directory 2-15

Guaranteed Delivery and Recovery from Server Failures ... 2-15
Inbound Service Name WSDL File ... 2-15
Inbound Header WSDL File ... 2-16
Synchronous File Reading Capabilities ... 2-17
File Adapter Write File Concepts ... 2-18
Outbound Operation ... 2-19
Outbound File Directory Creation ... 2-19
Specifying Outbound Physical or Logic Directory Paths ... 2-20
Specifying the Outbound File Naming Convention ... 2-21
Specifying a Dynamic Outbound File Name ... 2-23
Batching Multiple Outbound Messages ... 2-24
Native Data Translation ... 2-25
Error Handling ... 2-25
Outbound Service Name WSDL File ... 2-26
Outbound Header WSDL File ... 2-27
FTP Adapter for Get File Concepts ... 2-27
FTP Adapter for Put File Concepts ... 2-32
Using Secure FTP with the FTP Adapter ... 2-34
Secure FTP Overview ... 2-34
Installing and Configuring OpenSSL ... 2-35
Installing and Configuring vsftpd ... 2-36
Creating an Oracle Wallet ... 2-38
Setting Up the FTP Adapter ... 2-38
Use Cases for the File and FTP Adapters ... 2-39
File Adapter Use Cases ... 2-39
File Reading ... 2-39
Message Debatching ... 2-40
Reading Delimited Content Files ... 2-40
Reading Positional (Fixed Length) Content Files ... 2-40
File Writing ... 2-40
FTP Adapter Use Case ... 2-40
Summary ... 2-41

3 Oracle Application Server Adapter for Advanced Queuing

Introduction to the AQ Adapter ... 3-1
AQ Adapter Features ... 3-1
Enqueue-Specific Features (Message Production) ... 3-2
Dequeue and Enqueue Features ... 3-3
Supported ADT Payload Types ... 3-4
Native Format Builder Wizard ... 3-5
Use Cases for the AQ Adapter ... 3-6
Adapter Configuration Wizard Walkthrough ... 3-6
Generated WSDL file ... 3-14
Dequeuing and Enqueuing Object and ADT Payloads ... 3-16
Dequeuing One Column of the Object/ADT Payload ... 3-16
Processing Large Numbers of Messages ... 3-16
Using Correlation ID for Filtering Messages During Dequeue ... 3-17

Enqueuing and Dequeuing from Multisubscriber Queues ... 3-17
Rule-Based Subscription for Multiconsumer Queues ... 3-19
Using AQ Headers in a BPEL Process ... 3-20
Header Variables in JDeveloper BPEL Designer ... 3-21
Configuring a Message Rejection Handler for Data Errors ... 3-22
Summary ... 3-22

4 Oracle Application Server Adapter for Databases

Introduction to the Database Adapter ... 4-1
Database Adapter Features ... 4-1
Design Overview ... 4-2
Database Adapter Concepts ... 4-4
Relational-to-XML Mapping ... 4-4
Relational Types to XML Schema Types ... 4-7
Mapping Any Relational Schema to Any XML Schema ... 4-8
SQL Operations as Web Services ... 4-8
DML Operations ... 4-8
Merge ... 4-9
querybyExample ... 4-9
Use Cases for Outbound Invoke Operations ... 4-9
Polling Strategies ... 4-10
Physical Delete ... 4-10
Logical Delete ... 4-11
Sequencing Table: Last-Read Id ... 4-13
Sequencing Table: Last Updated ... 4-14
Control Tables ... 4-14
Use Cases for Polling Strategies ... 4-16
Use Cases for the Database Adapter ... 4-16
The Adapter Configuration Wizard ... 4-18
Starting the Adapter Configuration Wizard ... 4-18
Connecting to a Database ... 4-19
Selecting the Operation Type ... 4-20
Selecting and Importing Tables ... 4-21
Defining Primary Keys ... 4-22
Creating Relationships ... 4-23
What Happens When Relationships Are Created or Removed ... 4-25
Different Types of One-to-One Mappings ... 4-25
Creating the Object Model ... 4-26
Defining a WHERE Clause ... 4-26
Choosing an After-Read Strategy ... 4-28
Delete the Rows that Were Read ... 4-29
Update a Field in the Table (Logical Delete) ... 4-29
Update a Sequencing Table ... 4-30
Internal Processes at Design Time ... 4-31
Importing Tables ... 4-31
Creating Relationships ... 4-32
Generating Design-Time Artifacts ... 4-32

Advanced Configuration 4-32
The OracleAS TopLink Mapping Workbench Project 4-32
Deleting a Descriptor 4-33
Returning Partial Objects When Querying 4-33
Renaming a Mapping 4-35
Configuring Offline Database Tables 4-35
Relational-to-XML Mappings (toplink_mappings.xml) 4-36
The Service Definition (WSDL) 4-38
DBWriteInteractionSpec 4-39
DBReadInteractionSpec 4-40
DBActivationSpec 4-41
XML Schema Definition (XSD) 4-43
Deployment 4-44
Location of the oc4j-ra.xml File 4-46
Advanced Properties 4-48
Performance 4-49
Outbound Write: Should You Use Merge, Write, or Insert? 4-49
The OracleAS TopLink Cache: When Should You Use It? 4-50
Existence Checking 4-50
Inbound (Polling): maxRaiseSize 4-50
Inbound (Polling): Choosing a Polling Strategy 4-50
Relationship Reading (Batch Attribute and Joined Attribute Reading) 4-50
Connection Pooling 4-50
Inbound Distributed Polling 4-51
Concurrency Control: Pessimistic Locking 4-51
Load Balancing: MaxTransactionSize and Pessimistic Locking 4-52
Third-Party Database Support 4-52
Design Time 4-53
Run Time 4-54
Stored Procedure and Function Support 4-54
Design Time: Using the Adapter Configuration Wizard 4-55
Using Top-Level Standalone APIs 4-55
Using Packaged APIs and Overloading 4-59
Design Time: WSDL and XSD Generation 4-61
The WSDL–XSD Relationship 4-61
Supported Primitive Datatypes 4-62
Generated XSD Attributes 4-63
User-Defined Types 4-64
Complex User-Defined Types 4-66
Object Type Inheritance 4-66
Object References 4-66
Run Time: Before Stored Procedure Invocation 4-67
Value Binding 4-67
Datatype Conversions 4-69
Run Time: After Stored Procedure Invocation 4-69
Datatype Conversions 4-69
Null Values 4-70

Function Return Values 4-70
Advanced Topics 4-70
Support for REF CURSOR 4-70
Support for PL/SQL BOOLEAN 4-71
Support for PL/SQL RECORD 4-71
Use Case for Creating and Configuring a Stored Procedure in JDeveloper BPEL Designer 4-71
Creating a Stored Procedure 4-72
Creating a Database Connection 4-72
Creating a Workspace and a Greeting Process 4-73
Creating a Partner Link 4-74
Creating an Invoke Activity 4-77
Creating an Initial Assign Activity 4-78
Creating a Second Assign Activity 4-80
Validating, Compiling, and Deploying the Greeting Process 4-81
Running the Greeting Process 4-82
Summary 4-83

5 Oracle Application Server Adapter for Java Message Service

Introduction to the JMS Adapter 5-1
JMS Adapter Features 5-1
Use Cases for the JMS Adapter 5-2
Concepts 5-2
Using the Adapter Configuration Wizard to Configure a JMS Adapter 5-4
Generated WSDL File 5-10
oc4j-ra.xml file 5-12
Produce Message Procedure 5-12
Configuring for OJMS 5-13
Configuring for OC4J JMS 5-15
Configuring for TIBCO JMS 5-16
Direct Connection 5-17
Configuring for IBM Websphere JMS 5-17
Summary 5-18

6 Native Format Builder Wizard

Creating Native Schema Files with the Native Format Builder Wizard 6-1
Supported Formats 6-2
Delimited (such as CSV files) 6-2
Fixed Length (Positional) 6-2
DTD 6-2
COBOL Copybook 6-2
User Inputs 6-3
COBOL Clauses 6-3
Native Format Builder Wizard Windows 6-6
Understanding Native Schema 6-7
Use Cases for the Native Format Builder 6-7
Defining a Comma-Separated Value File Structure 6-8

Native Data Format to Be Translated 6-8
Native Schema 6-8
Translated XML Using the Native Schema 6-9
Defining a * Separated Value File Structure 6-9
Native Data Format to Be Translated 6-9
Native Schema 6-9
Defining a Fixed Length Structure 6-9
Native Data Format to Be Translated 6-9
Native Schema 6-9
Defining a More Complex Structure - Invoice 6-10
Native Data Format to Be Translated 6-10
Native Schema 6-11
Translated XML Using the Native Schema 6-12
COBOL Copybook 6-13
Multiple Root Levels 6-13
Single Root Level, Virtual Decimal, Fixed Length Array 6-16
Variable Length Array 6-18
Numeric Types 6-20
Native Schema Constructs 6-21
Defining Fixed Length Data 6-21
Native Data Format to Be Translated: With Padding 6-21
Native Schema: With Padding 6-21
Translated XML Using the Native Schema: With Padding 6-22
Native Data Format to Be Translated: Without Padding 6-22
Native Schema: Without Padding 6-22
Translated XML Using the Native Schema: Without Padding 6-23
Native Data Format to Be Translated: Actual Length Also Being Read from the Native Data 6-23
Native Schema: Actual Length Also Being Read from the Native Data 6-23
Translated XML Using the Native Schema: Actual Length Also Being Read from the Native Data 6-23
Defining Terminated Data 6-24
Native Data Format to Be Translated: Optionally Quoted 6-24
Native Schema: Optionally Quoted 6-24
Translated XML Using the Native Schema: Optionally Quoted 6-24
Native Data Format to Be Translated: Not Quoted 6-24
Native Schema: Not Quoted 6-24
Translated XML Using the Native Schema: Not Quoted 6-25
Defining Surrounded Data 6-25
Native Data Format to Be Translated: Left and Right Surrounding Marks Are Different 6-25
Native Schema: Left and Right Surrounding Marks Are Different 6-25
Translated XML Using the Native Schema: Left and Right Surrounding Marks Are Different 6-26
Native Data Format to Be Translated: Left and Right Surrounding Marks Are the Same 6-26
Native Schema: Left and Right Surrounding Marks Are the Same 6-26

Translated XML Using the Native Schema: Left and Right Surrounding Marks Are the Same 6-26
Defining Lists 6-26
Native Data Format to Be Translated: All Items Separated by the Same Mark, But the Last Item Terminated by a Different Mark (Bounded) 6-27
Native Schema: All Items Separated by the Same Mark, But the Last Item Terminated by a Different Mark (Bounded) 6-27
Translated XML Using the Native Schema: All Items Separated by the Same Mark, But the Last Item Terminated by a Different Mark (Bounded) 6-27
Native Data Format to Be Translated: All Items Separated by the Same Mark, Including the Last Item (Unbounded) 6-27
Native Schema: All Items Separated by the Same Mark, Including the Last Item (Unbounded) 6-27
Translated XML Using the Native Schema: All Items Separated by the Same Mark, Including the Last Item (Unbounded) 6-28
Defining Arrays 6-28
Native Data Format to Be Translated: All Cells Separated by the Same Mark, But the Last Cell Terminated by a Different Mark (Bounded) 6-28
Native Schema: All Cells Separated by the Same Mark, But the Last Cell Terminated by a Different Mark (Bounded) 6-28
Translated XML Using the Native Schema: All Cells Separated by the Same Mark, But the Last Cell Terminated by a Different Mark (Bounded) 6-29
Native Data Format to Be Translated: 6-29
Native Schema: 6-29
Translated XML Using the Native Schema: All Cells Separated by the Same Mark, Including the Last Cell (Unbounded) 6-30
Native Data Format to Be Translated: Cells Not Separated by Any Mark, But the Last Cell Terminated by a Mark (Bounded) 6-30
Native Schema: Cells Not Separated by Any Mark, But the Last Cell Terminated by a Mark (Bounded) 6-31
Translated XML Using the Native Schema: Cells Not Separated by Any Mark, But the Last Cell Terminated by a Mark (Bounded) 6-31
Native Data Format to Be Translated: The Number of Cells Being Read from the Native Data 6-31
Native Schema: The Number of Cells Being Read from the Native Data 6-32
Translated XML Using the Native Schema: The Number of Cells Being Read from the Native Data 6-32
Conditional Processing 6-33
Native Data Format to Be Translated: Processing One Element within a Choice Model Group Based on the Condition 6-33
Native Schema: Processing One Element within a Choice Model Group Based on the Condition 6-33
Translated XML Using the Native Schema: Processing One Element Within a Choice Model Group Based on the Condition 6-35
Native Data Format to Be Translated: Processing Elements within a Sequence Model Group Based on the Condition 6-36
Native Schema: Processing Elements within a Sequence Model Group Based on the Condition 6-36
Translated XML Using the Native Schema: Processing Elements within a Sequence Model Group Based on the Condition 6-38

Defining Dates 6-40
Native Data Format to Be Translated 6-40
Native Schema 6-40
Translated XML Using the Native Schema 6-40
Using Variables 6-41
Native Data Format to Be Translated 6-41
Native Schema 6-41
Translated XML Using the Native Schema 6-42
Native Schema Constructs 6-42
Summary 6-44

A Troubleshooting and Workarounds

Troubleshooting the Oracle Application Server Adapter for Databases A-1
Could Not Create OracleAS TopLink Session Exception A-1
Could Not Find Adapter for eis/DB/*my_connection* A-1
Changes Through TopLink Mapping Workbench A-2
Redeploying from the Command Line A-2
Cannot Change Customers_table.xsd A-2
No Target Foreign Keys Error A-2
No Primary Key Exception A-3
dateTime Conversion Exceptions A-4
Issues with Oracle DATE A-5
Handling a Database Adapter Fault A-6
BPEL Process Does Not Run Against Another Database A-6
Only One Employee Per Department Appears A-7
Outbound SELECT on a CHAR(X) or NCHAR Column Returns No Rows A-7
ORA-00932: Inconsistent Datatypes Exception Querying CLOBs A-8
Merge Sometimes Does UPDATE Instead of INSERT, or Vice Versa A-8
Integrity Violation Occurs with Delete or DeletePollingStrategy A-9
Some Queried Rows Appear Twice or Not at All in the Query Result A-10
Importing a Same-Named Table, with Same Schema Name, but Different Databases A-10
Problems Creating a Relationship Manually for a Composite Primary Key A-11
Must Fully Specify Relationships Involving Composite Primary Keys A-11
Database Adapter Throws an Exception When Using a BFILE A-11
During Design-Time, Wizard Does Not Allow Deletion of a Table A-11
Changes to JDeveloper Project Are Made Even If Wizard Is Cancelled A-11
Problems Removing a Relationship, Then Adding a New Relationship with the Same Name A-11
Problems Importing Third-Party Database Tables with Unsupported Database Types A-12
Problems Importing Object Tables A-12
Relationships Not Autogenerated When Tables Are Imported Separately A-12
Primary Key Is Not Saved A-13
Troubleshooting the Oracle Application Server Adapter for Databases When Using Stored Procedures A-13
Design-Time Problems: Unsupported Parameter Types A-13
Run-Time Problems: Parameter Mismatches A-14
Run-Time Problems: Stored Procedure Not Defined in the Database A-15

Troubleshooting the Oracle Application Server Adapter for Files/FTP A-16
Changing Logical Names with the Adapter Configuration Wizard A-16
Creating File Names with Spaces with the Native Format Builder Wizard A-16
Common User Errors A-16
Troubleshooting the Oracle Application Server Adapter for Advanced Queuing A-18
Inbound Errors A-18
JNDI Lookup Failed A-18
During Initialization, I/O Exception: Network Adapter Did Not Establish the Connection A-19
Incorrect Username/Password A-19
Queue Not Found A-20
User Does Not Have DBMS_AQIN Privileges, Which Are Required by the AQ Java API A-20
Translation Error A-20
Subscriber Already Exists When Using MessageRuleSelector A-21
Outbound Errors A-22
JNDI Lookup Failed A-22
I/O Exception: Network Adapter Could Not Establish the Connection A-22
Queue Not Found A-23
Incorrect Username/Password A-23
User Does Not Have DBMS_AQIN Privileges, Which Are Required by the AQ Java API A-23
Translation Error A-24
JDeveloper BPEL Designer Errors A-24
Translation Error A-26
Other Problems A-27
Summary A-29

Index

Preface

This guide describes how to use the technology adapters that are provided with Oracle BPEL Process Manager.

This preface contains the following topics:

- Audience
- Documentation Accessibility
- Related Documentation
- Conventions

Audience

Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User's Guide is intended for anyone who is interested in using these adapters.

Documentation Accessibility

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

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documentation

For more information, see these Oracle resources:

- Oracle Application Server 10*g* Documentation Library
- *Oracle BPEL Process Manager Developer's Guide*

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

To download free release notes, installation documentation, white papers, or other collateral, visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://www.oracle.com/technology/membership/`

To download Oracle BPEL Process Manager documentation, technical notes, or other collateral, visit the Oracle Technology Network (OTN) at

`http://www.oracle.com/technology/bpel/`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://www.oracle.com/technology/documentation/`

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.

1

Introduction to Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging

This chapter describes the file, FTP, database, and enterprise messaging adapters that are provided with Oracle BPEL Process Manager. The adapters enable you to integrate BPEL processes with access to file systems, FTP servers, database tables, database queues (advanced queues, or AQ), Java Message Services (JMS), and Oracle Applications. See *Oracle BPEL Process Manager Developer's Guide* for information about BPEL processes.

This chapter contains the following topics:

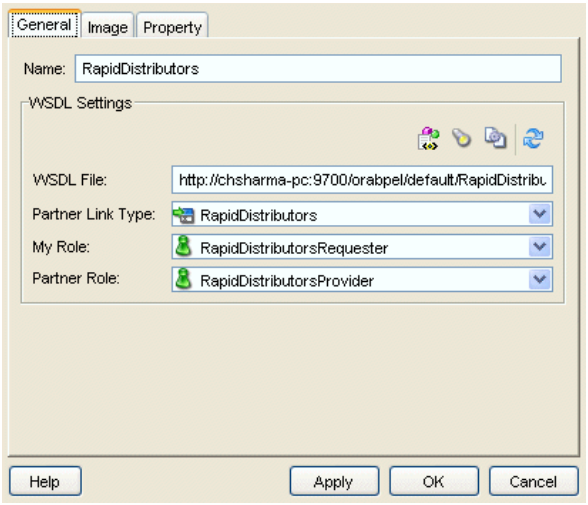
- Overview of Oracle BPEL Process Manager Technology Adapters
- Summary

See *Oracle Application Server Adapter Concepts* for information about application and mainframe adapters.

Overview of Oracle BPEL Process Manager Technology Adapters

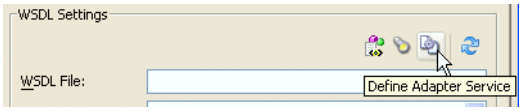
From the Partner Link Window, shown in Figure 1–1, you can access the adapters that are provided with Oracle BPEL Process Manager.

Figure 1–1 Partner Link Window



Click the **Define Adapter Service** icon, shown in Figure 1–2, to access the Adapter Configuration Wizard.

Figure 1–2 Defining an Adapter



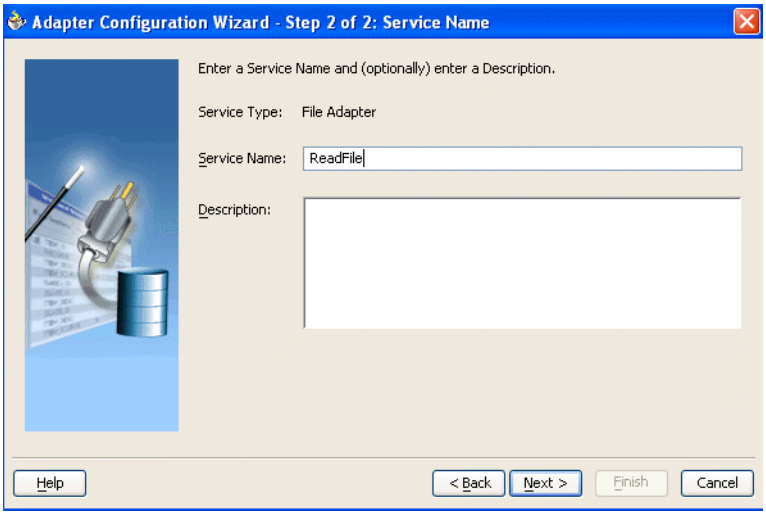
This wizard enables you to configure the types of adapters shown in Figure 1–3 for use with BPEL processes.

Figure 1–3 Adapter Types

When you select an adapter type, the Service Name window shown in Figure 1–4 prompts you to enter a name. For this example, **File Adapter** was selected in Figure 1–3. When the wizard completes, a WSDL file by this service name appears in the **Applications Navigator** for the BPEL process (for this example, named **ReadFile.wsdl**). This file includes the adapter configuration settings you specify with this wizard. Other configuration files (such as header files and files specific to the adapter) are also created and display in the **Applications Navigator**.

See *Oracle Application Server Adapter for Oracle Applications User's Guide* for information on using the Oracle Applications adapter listed in Figure 1–3.

Figure 1–4 Adapter Service Name



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.

Summary

This chapter introduces the file, FTP, database, and enterprise messaging adapters that are provided with Oracle BPEL Process Manager. The adapters enable you to integrate BPEL processes with access to file systems, FTP servers, database tables, database queues, Java Message Services (JMS), and Oracle Applications.

2

Oracle Application Server Adapter for Files/FTP

This chapter describes how to use the Oracle Application Server Adapter for Files/FTP (file and FTP adapters), which work in conjunction with Oracle BPEL Process Manager as an external service. References to use cases for the file and FTP adapters are also provided.

This chapter contains the following topics:

- Introduction to the File and FTP Adapters
- File and FTP Adapter Concepts
- Using Secure FTP with the FTP Adapter
- Use Cases for the File and FTP Adapters
- Summary

> **Note:** The term *Oracle Application Server Adapter for Files/FTP* is used for the file and FTP adapters, which are separate adapters with very similar functionality.

Introduction to the File and FTP Adapters

Oracle BPEL Process Manager includes the file and FTP adapters. The file and FTP adapters enable your BPEL process 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 contains the following topics:

- File and FTP Adapter Features
- File and FTP Adapter Architecture
- File and FTP Adapter Integration with Oracle BPEL Process Manager

File and FTP Adapter Features

The file and FTP adapters enable you to configure your BPEL process to interact with local and remote file system directories. The file and FTP adapters can read and write the following file formats and use the adapter translator component at both design time and run time:

- XML (both XSD- and DTD-based)

- Delimited
- Fixed positional
- Binary data
- COBOL Copybook data

The 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.

The translator enables the file and FTP adapters to convert native data in various formats to XML, and vice versa. The native data can be simple (just a flat structure) or complex (with parent-child relationships).

The FTP adapter supports the use of secure FTP on Solaris.

The file and FTP adapters exchange files in the inbound and outbound directions. Based on the direction, the file and FTP adapters perform a different set of tasks.

For inbound files sent to Oracle BPEL Process Manager, the file and FTP adapters perform the following operations:

- Poll the file system looking for matches
- Read and translate the file contents based on the translation logic defined at design time
- Publish the same as an XML message

This functionality of the file and FTP adapters is referred to as the file read operation and the component that provides this function as the file reader. This operation is known as a Java Connector Architecture (JCA) inbound interaction.

For outbound files sent from Oracle BPEL Process Manager, the file and FTP adapters perform the following operations:

- Receive messages from BPEL
- Format the XML contents as specified at design time
- Produce output files

This functionality of the file and FTP adapters is referred to as the file write operation and the component that provides this functionality as the file writer. This operation is known as a JCA outbound interaction.

For the inbound and outbound directions, the file and FTP adapters use a set of configuration parameters. For example:

- The inbound 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 file and FTP adapters have parameters for the outbound directory in which to write the file and the file naming convention to use.

The file reader supports polling conventions and offers several postprocessing options. After processing the file, the files can be deleted, moved to a directory, or left as is. The file reader can split the contents of a file and publish it in batches, instead of as a single message. This feature can be utilized for performance tuning of the file and FTP adapters. The file reader guarantees once and once-only delivery.

When a file contains multiple messages, you can select to publish messages in a specific number of batches. This is referred to as debatching. During debatching, the

file reader, upon restart, proceeds from where it left off in the previous run, thereby avoiding duplicate messages.

The file writer offers several conditions for output file creation. The output files can be created based on time elapsed, file size, and number of messages received.

See the following sections for details about the read and write functionality of the file and FTP adapters:

- "File Adapter Read File Concepts" on page 2-4
- "File Adapter Write File Concepts" on page 2-18
- "FTP Adapter for Get File Concepts" on page 2-27
- "FTP Adapter for Put File Concepts" on page 2-32

File and FTP Adapter Architecture

The file and FTP adapters are based on JCA 1.5 architecture. JCA provides a standard architecture for integrating heterogeneous enterprise information systems (EIS). The adapter framework of the file and FTP adapters exposes the underlying JCA interactions as services (WSDL with JCA binding) for Oracle BPEL Process Manager integration. See *Oracle Application Server Adapter Concepts* for details about OracleAS Adapter architecture.

File and FTP Adapter Integration with Oracle BPEL Process Manager

The file and FTP adapters are automatically integrated with Oracle BPEL Process Manager. When you create a partner link in JDeveloper BPEL Designer, you can invoke the Adapter Configuration Wizard, as shown in Figure 1–2 on page 1-2.

This wizard enables you to select and configure the file and FTP adapters or other OracleAS Adapters, as shown in Figure 1–3 on page 1-2. The Adapter Configuration Wizard then prompts you to enter a service name, as shown in Figure 1–4 on page 1-3. When configuration is complete, a WSDL file of the same name is created in the **Applications Navigator** section of JDeveloper BPEL Designer. This WSDL file contains the configuration information you specify with the Adapter Configuration Wizard.

After specifying a service name (as shown in Figure 1–4 on page 1-3), you are prompted to select an operation to perform. Based on your selection, different Adapter Configuration Wizard windows appear and prompt you for configuration information. Table 2–1 lists the available operations and provides references to sections that describe the configuration information you must provide.

Table 2–1 Supported Operations

Operation	See Section...
File Adapter	-
■ Read File (inbound operation)	"File Adapter Read File Concepts" on page 2-4
■ Write File (outbound operation)	"File Adapter Write File Concepts" on page 2-18
FTP Adapter	-
■ Get File (inbound operation)	"FTP Adapter for Get File Concepts" on page 2-27
■ Put File (outbound operation)	"FTP Adapter for Put File Concepts" on page 2-32

See Oracle Application Server Adapter Concepts for more information about OracleAS Adapter integration with Oracle BPEL Process Manager.

File and FTP Adapter Concepts

This section contains the following topics:

- File Adapter Read File Concepts
- File Adapter Write File Concepts
- FTP Adapter for Get File Concepts
- FTP Adapter for Put File Concepts

File Adapter Read File Concepts

In the inbound direction, the 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 file adapter. You use the Adapter Configuration Wizard to configure the file adapter for use with your BPEL process. This creates an inbound WSDL file named after the service name you specify with the Adapter Configuration Wizard. An inbound header file named `fileAdapterInboundheader.wsdl` is also created.

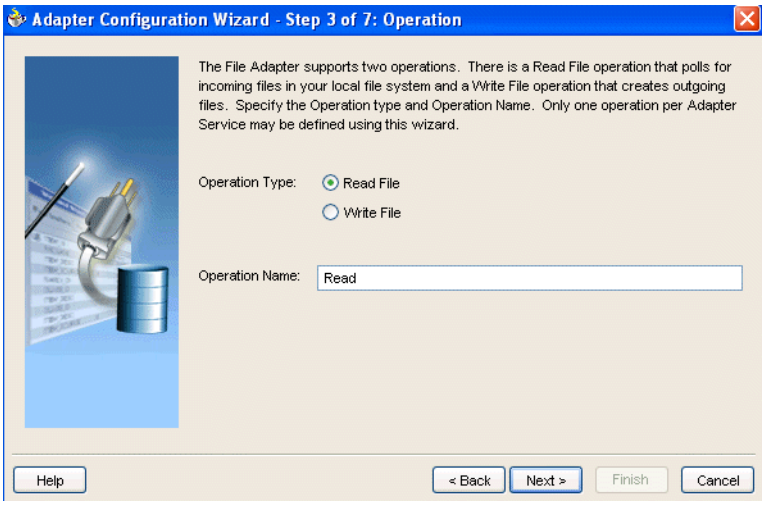
This section contains the following topics:

- Inbound Operation
- Inbound File Directory Specifications
- File Matching and Batch Processing
- File Polling
- File Processing
- Postprocessing
- Native Data Translation
- Error Handling
- Guaranteed Delivery and Recovery from Server Failures
- Inbound Service Name WSDL File
- Inbound Header WSDL File
- Synchronous File Reading Capabilities

Inbound Operation

For inbound operations with the file adapter, you select to perform an inbound **Read File** operation. Figure 2–1 shows this selection.

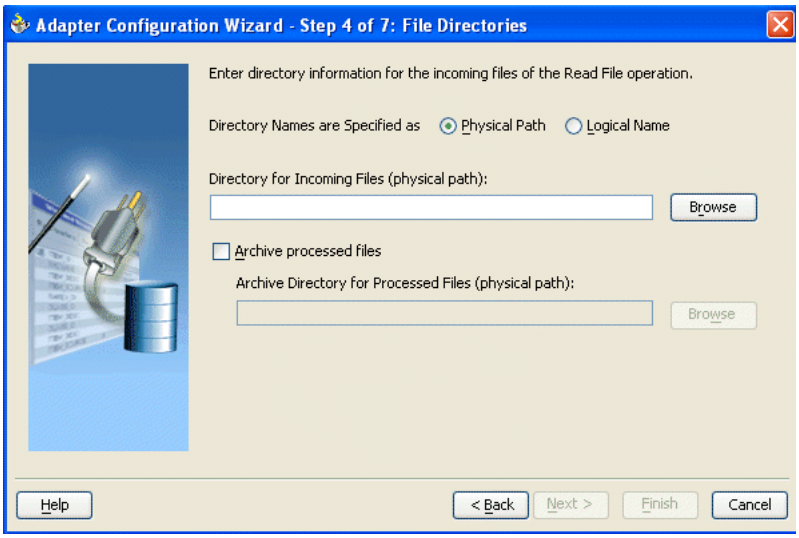
Figure 2–1 Selecting the Read File Operation



Inbound File Directory Specifications

The File Directories window of the Adapter Configuration Wizard shown in Figure 2–2 enables you to specify information about the directory to use for reading inbound files and the directories in which to place successfully processed files.

Figure 2–2 Adapter Configuration Wizard—Specifying Incoming Files



The following sections describe the file directory information to specify:

- Specifying Inbound Physical or Logical Directory Paths
- Archiving Successfully Processed Files

Specifying Inbound Physical or Logical Directory Paths You can specify inbound directory names as physical or logical paths. Physical paths are values such as `c:\inputDir`.

Logical properties are specified in the inbound WSDL file and their logical-physical mapping is resolved using partner link properties. You specify the logical parameters once at design time, and you can later modify the physical directory name as needed.

Provide a logical name. For example, the generated inbound WSDL file looks as follows for the logical input directory name `InputFileDir`.

```
<operation name="Read">
   <jca:operation
      LogicalDirectory="InputFileDir"
      ActivationSpec="oracle.tip.adapter.file.inbound.FileActivationSpec"
      IncludeFiles=".*"
      PollingFrequency="5"
      MinimumAge="0"
      DeleteFile="true"
      OpaqueSchema="true" >
   </jca:operation>
```

In the BPEL partner link of the `bpel.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.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<BPELSuitcase>
  <BPELProcess id="ComplexStructure" src="ComplexStructure.bpel">
    <partnerLinkBindings>
      <partnerLinkBinding name="InboundPL">
        <property name="wsdlLocation">ComplexStructureInbound.wsdl</property>
      </partnerLinkBinding>
      <partnerLinkBinding name="OutboundPL">
        <property name="wsdlLocation">ComplexStructureOutbound.wsdl</property>
      </partnerLinkBinding>
    </partnerLinkBindings>
      <activationAgents>
      <activationAgentclassName=
"oracle.tip.adapter.fw.agent.jca.JCAActivationAgent"partnerLink="InboundPL">
      <property name="InputFileDir">C:/ora_home/integration/bpm/samples/tutorials/
121.FileAdapter/ComplexStructure/InputDir/</property>
      <property name="portType">Read_ptt</property>
    </activationAgent>
   </activationAgents>
  /BPELProcess>
</BPELSuitcase>
```

> **Note:** Multiple BPEL processes or multiple file adapters polling one inbound directory are *not* supported. Ensure that all are polling their own unique directory.

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" on page 2-5.

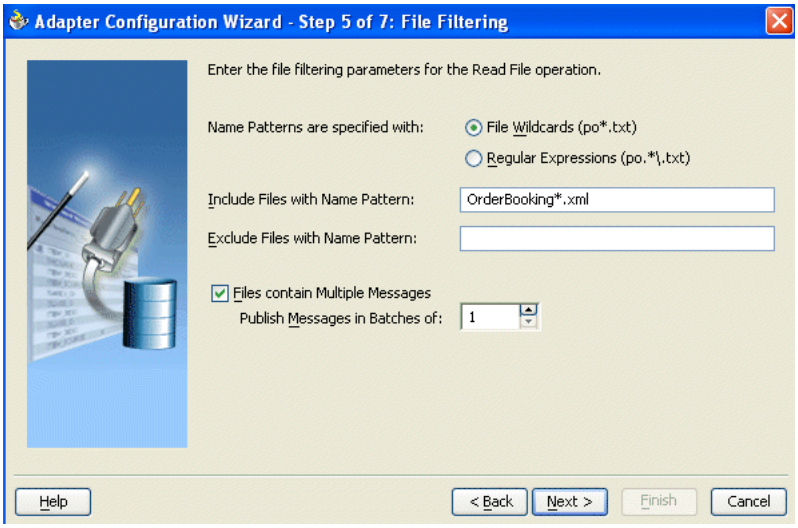
> **Note:** Files greater than or equal to 7 MB cannot be delivered. As an alternative, debatch large files (if they have multiple messages), and publish these files in messages of sizes less than 7 MB.

File Matching and Batch Processing

The File Filtering window of the Adapter Configuration Wizard shown in Figure 2–3 enables you to specify details about the files to retrieve or ignore.

The file adapter acts as a file listener in the inbound direction. The file adapter polls the specified directory on a local or remote file system and looks for files that match a specified naming criteria.

Figure 2–3 Adapter Configuration Wizard—File Filtering



The following sections describe the file filtering information to specify:

- Specifying a Naming Pattern
- Including and Excluding Files
- Batching Multiple Inbound Messages

Specifying a Naming Pattern Specify the naming convention that the 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 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.

Notes:

- 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 filename, 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 like `\`, `/`, `:`, `*`, `<`, `>`, or `|`.

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.

If you want the inbound 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, the exclude files expression takes precedence. For example, if **Include Files** is set to **abc*.txt** and **Exclude Files** is set to **abcd*.txt**, you receive any files prefixed with **abcd***.

Table 2–2 lists details of Java regex constructs.

Note: Do not begin JDK regex pattern names with the following characters: `+`, `?`, or `*`.

Table 2–2 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 newline (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]
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 whitespace 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*

Table 2–2 (Cont.) Java Regular Expression Constructs

Matches	Construct
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, go to

`http://java.sun.com/j2se/1.4.2/docs/api`

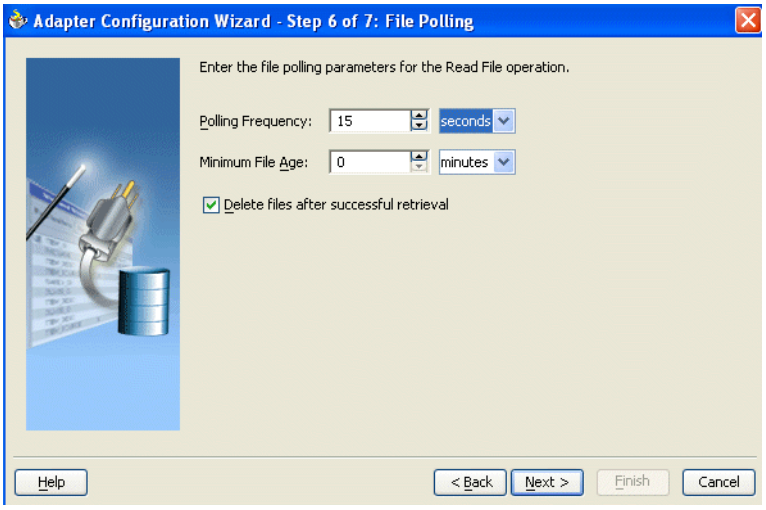
Batching Multiple Inbound Messages You can select if incoming files have more than one message, and specify the number of messages in one batch file to publish. When a file contains multiple messages and this check box is selected, this is referred to as debatching. Nondebatching is when the file contains only a single message and the check box is not selected.

File Polling

The File Polling window of the Adapter Configuration Wizard shown in Figure 2–4 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 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, set the minimum age of pollable files to five minutes. If a file is detected in the input directory and its modification time is less than 5 minutes older than the current time, the file is not retrieved because it is still potentially being written to.
- Whether or not 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 already been processed, but the modification date remains the same, that file is not retrieved.

Figure 2–4 Adapter Configuration Wizard—File Polling



File Processing

The file adapter prepares the files for processing and delivers them to the adapter translator for translation and debatching (if necessary).

If you have many inbound files to process or very large files of more than 1 MB, you may need to increase the `config timeout` value in the *`Oracle_Home`*`\integration\orabpel\system\appserver\oc4j\j2ee\home\server.xml` file:

```
<transaction-config timeout="30000"/>
```

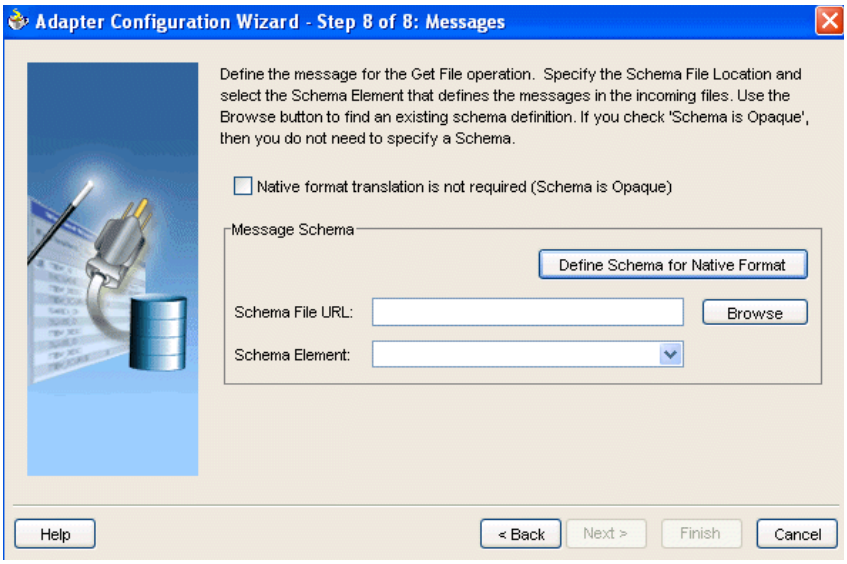
Postprocessing

The file adapter supports several postprocessing options. After processing the file, the files can be deleted if specified in the File Polling window shown in Figure 2–4. Files can also be moved to a completion (archive) directory if specified in the File Directories window shown in Figure 2–2 on page 2-5.

Native Data Translation

The next Adapter Configuration Wizard window that appears is the Messages window shown in Figure 2–5. This window enables you to select the XSD schema file for translation.

Figure 2–5 Specifying the Schema



If native format translation is not required (for example, a JPG or GIF image is being processed) 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. If you want 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 comma-separated value (CSV), fixed-length, DTD, and COBOL Copybook. After the native schema file is created, you are returned to this Messages window with the **Schema File URL** and **Schema Element** fields filled in. See "Creating Native Schema Files with the Native Format Builder Wizard" on page 6-1 for more information.

Note: Ensure that the schema you specify includes a namespace. If your schema does not have a namespace, an error message appears.

Error Handling

The file adapter provides several inbound error handling capabilities:

- rejectedMessageHandlers Property
- fatalErrorFailoverProcess Property
- uniqueMessageSeparator Property
- Default Error Directory

rejectedMessageHandlers Property You can configure your BPEL process to process the correct records of a file and write only the rejected records to an archive directory by setting the `rejectedMessageHandlers` property. For example, assume that you have a file with four records. If three records are processed successfully and one record is not, the file is processed with the three correct records. The record that errored is written to a rejected messages directory. The behavior of the `rejectedMessageHandlers` property in case a file has only one message is to reject the entire message.

You first define the `rejectedMessageHandlers` property as an `activationAgent` property in the `bpel.xml` file so that it applies to inbound WSDL operations only:

```
<BPELSuitcase>
  <BPELProcess src="ErrorTest.bpel" id="ErrorTest">
    <activationAgents>
       <activationAgent
          className="oracle.tip.adapter.fw.agent.jca.JCAActivationAgent"
                       partnerLink="inboundPL">
         <property name="rejectedMessageHandlers">
            file://C:/orabpel/samples/test/errorTest/rejectedMessages
         </property>
```

This causes messages that error to be written to the configured directory using the following naming pattern:

```
INVALID_MSG_ + process-name + operation-name + current-time
```

fatalErrorFailoverProcess Property If the file adapter (or any OracleAS Adapter) encounters an unrecoverable system error (such as no more memory or a full disk), it can instruct the adapter framework to shut down the BPEL process.

You can optionally configure a standby (or failover) BPEL process to be invoked when the adapter initiates the shutdown request of the (main) BPEL process.

You configure this failover BPEL process by setting the `fatalErrorFailoverProcess` property as an activationAgent property in the `bpel.xml` file.

```
<property name="fatalErrorFailoverProcess">
bpel://bpel-domain:password|process-name|operation-name|
input-message-part-name
</property>
```

where *`password`* (which can be omitted if it is `bpel`) can be encrypted.

For example:

```
<property name="fatalErrorFailoverProcess">

  bpel://default|JCA-FatalErrorHandler|handleError|message
</property>
```

or

```
<property name="fatalErrorFailoverProcess">

bpel://default:C23487CFA591952D3ED0B81F0961F65A|JCA-FatalErrorHandler|handleError|
message</property>
```

where the `bpel` password was specified in encrypted form (using the `encrypt.bat` (for Windows) or `encrypt.sh` (for UNIX) command line utility).

The fatal error BPEL process must use a specific input (WSDL) message type. This message type is defined in the system-provided WSDL file `FatalErrorMessageWSDL.wsdl`. This WSDL can be referenced (imported) using the following:

```
<import namespace="http://xmlns.oracle.com/pcbpel/fatalErrorHandler"
location="http://localhost:9700/orabpel/xmllib/jca/FatalErrorMessage.wsdl"/>
```

The XML schema type for this message is as follows:

```
<complexType name="FatalErrorMessageType">
  <sequence>
    <element name="Reason" type="string"/>
    <element name="Exception" type="string"/>
    <element name="StackTrace" type="string"/>
  </sequence>
</complexType>
```

The purpose of the failover BPEL process can be to undertake error compensating actions, alert someone through e-mail or short message service (SMS), or restart some other process.

uniqueMessageSeparator Property In the case of debatching (multiple messages in a single file), the typical behavior is to reject the messages from the first bad message to the end of the file. 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` 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, all records that follow the record that errored are also rejected.

The following schema file provides an example of using `uniqueMessageSeparator`.

```
<?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:-->
```

Default Error Directory If you do not set the `rejectedMessageHandlers` property, records that error during translation are placed by default in

Oracle_Home\integration\orabpel\domains*domain_name*\archive\jca*default_directory*

Guaranteed Delivery and Recovery from Server Failures

The file adapter guarantees once-only delivery of inbound files. This includes guaranteed delivery of large files through FTP. If your system goes down, the read functionality of the file adapter upon restart avoids creating duplicate messages.

If your system server crashes while inbound messages are being processed, you must manually perform recovery when the server restarts to ensure that all message records are recovered. For example, a file has ten messages and the server crashes after three messages have been processed. This causes the fourth message to go undelivered. When the server restarts and begins processing with message five (the offset of the last successfully rejected message), you must manually recover message four to ensure that all messages are preserved.

Perform the following procedures to recover the rejected message record.

1. Log in to Oracle BPEL Console.
2. Select the **BPEL Processes** tab.
3. Click **Perform Manual Recovery** under the **Related Tasks** section.
4. Click **Recover**.

Inbound Service Name WSDL File

When you finish configuring the file adapter, a WSDL file is generated for the inbound direction. The file is named after the service name you specified on the Service Name window of the Adapter Configuration Wizard shown in Figure 1–4 on page 1-3. You can rerun the wizard at any time to change your operation definitions.

The `ActivationSpec` parameter holds the inbound configuration information. The `ActivationSpec` and a set of inbound file adapter properties are part of the inbound WSDL file.

The inbound WSDL contains the following information:

```
<pc:inbound_binding />
  <operation name="Read">
    <jca:operation
      ActivationSpec="oracle.tip.adapter.file.inbound.FileActivationSpec"
      PhysicalDirectory="C:/ora_
home/integration/bpm/samples/tutorials/121.FileAdapter/ComplexStructure/inputDir/"
      PhysicalArchiveDirectory="C:/ora_
home/integration/bpm/samples/tutorials/121.FileAdapter/ComplexStructure/archiveDir
/"
      IncludeFiles=".+\.txt"
      PollingFrequency="5"
      MinimumAge="0"
      DeleteFile="true"
```

```
      OpaqueSchema="false" >
    </jca:operation>
    <input>
      <jca:header message="hdr:InboundHeader_msg" part="inboundHeader"/>
    </input>
  </operation>
</binding>
```

The `ActivationSpec` parameters are specified in the Adapter Configuration Wizard during design time and appear in the binding element of the inbound WSDL. The inbound file adapter uses the following configuration parameters:

- `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.

- `PhysicalDirectory`

  This parameter specifies the physical input directory 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.

- `LogicalDirectory`

  This parameter specifies the logical input directory to be polled. The parameter is of type `String`.

- `PublishSize`

  This parameter indicates if 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`.

- `PhysicalArchiveDirectory`

  This parameter specifies where to archive successfully processed files. The parameter is of type `String` and is not mandatory.

- `LogicalArchiveDirectory`

  This parameter specifies the logical directory in which to archive successfully processed files. The parameter is of type `String` and is not mandatory.

- `IncludeFiles`

  This parameter specifies the pattern for types of files to pick up during polling. The parameter is of type `String` and is not mandatory.

- `ExcludeFiles`

  This parameter specifies the pattern for types of files to be excluded during polling. The parameter is of type `String` and is not mandatory.

Inbound Header WSDL File

The WSDL file shown in "Inbound Service Name WSDL File" on page 2-15 includes two attributes that indicate which message and part define the operation headers:

```
<jca:header message="hdr:InboundHeader_msg" part="inboundHeader"/>
```

The `fileAdapterInboundHeader.wsdl` file defines these attributes. This file also provides information such as the name of the file being processed and its directory path. This file is created along with the service name WSDL file, and displays in the **Applications Navigator** of JDeveloper BPEL Designer.

```
<definitions
     name="fileAdapter"
     targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/"
     xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/file/"
     xmlns="http://schemas.xmlsoap.org/wsdl/" >
    <types>
        <schema attributeFormDefault="qualified" elementFormDefault="qualified"
                targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/"
                xmlns="http://www.w3.org/2001/XMLSchema"
                xmlns:FILEAPP="http://xmlns.oracle.com/pcbpel/adapter/file/">
          <element name="InboundFileHeaderType">
                <complexType>
                    <sequence>
                        <element name="fileName" type="string"/>
                        <element name="directory" type="string"/>
                    </sequence>
                </complexType>
            </element>
        </schema>
    </types>

    <!-- Header Message -->
    <message name="InboundHeader_msg">
      <part element="tns:InboundFileHeaderType" name="inboundHeader"/>
   </message>
</definitions>
```

See the online Help that is included with the **Adapter** tab in JDeveloper BPEL Designer for more information.

Synchronous File Reading Capabilities

The file adapter can synchronously read a file using an invoke activity. If the specified file does not exist, the read invoke activity returns nothing. You must manually edit the WSDL file to enable this functionality.

1. Create a WSDL file for the *write* file operation using the Adapter Configuration Wizard.
2. Select the schema of the file to be read in the Adapter Configuration Wizard Messages window.
3. Change the value of the parameters in the `<jca:operation> element` section of the created WSDL file to:

   ```
   InteractionSpec = oracle.tip.adapter.file.outbound.FileReadInteractionSpec
   PhysicalDirectory=Directory_from_which_to_read
   FileName=file_name_to_read
   DeleteFile=true | false
   ```

   where either `true` or `false` is selected for the `DeleteFile` parameter. Do not use a regular expression for the `FileName` parameter value. For synchronous reads, the file name must be known.
4. Review the `operation` section of the WSDL file, which includes information similar to the following:

   ```
   <operation name="Write">
   <input message="..."/>
   </operation>
   ```

5. Rename the `input message` to `output message`. The adapter uses this to return the translated file contents. This now points to the native format definition you created with the Adapter Configuration Wizard.
6. Create an `empty_msg` message type:

```
<element name="empty"><complexType/></element>
```

7. Add an `input message` to the `<operation>` pointing to an empty element as defined in step 6.
8. Modify the `<invoke>` activity in the BPEL file with appropriate `inputVariable` (pointing to the `empty_msg` type) and `outputVariable` attributes.

   A sample WSDL section looks as follows. Note that the dots below (. . .) represent areas you must complete with information appropriate to your environment.

```
<types>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="...">
<import..../>
<element name="empty"><complexType/></element>
</schema>
</types>

<message name="Address-List_msg"> /* The output message */
<part name="..." element="...:..."/>
</message>

<message name="Empty_msg">
<part name="Empty" element="...:empty"/>
</message>

<portType name="SynchronousRead_ptt">
<operation name="SynchronousRead">
<input message="tns:Empty_msg"/>
<output message="tns:Address-List_msg"/>
</operation>
</portType>

<binding name="SynchronousRead_binding" type="tns:SynchronousRead_ptt">
<jca:binding />
<operation name="SynchronousRead">
<jca:operation
PhysicalDirectory="D:\work\jDevProjects"
InteractionSpec="oracle.tip.adapter.file.outbound.FileReadInteractionSpec"
FileName="address_csv.txt"
DeleteFile="true"
OpaqueSchema="false" >
</jca:operation>
```

File Adapter Write File Concepts

In the outbound direction, the file adapter receives messages from the BPEL process and writes the messages to a file in a file system. This section provides an overview of the outbound file writing capabilities of the file adapter. You use the Adapter Configuration Wizard to configure the file adapter for use with your BPEL process. This creates an outbound WSDL file named after the service name you specify with the Adapter Configuration Wizard. An outbound header file named `fileAdapterOutboundheader.wsdl` is also created.

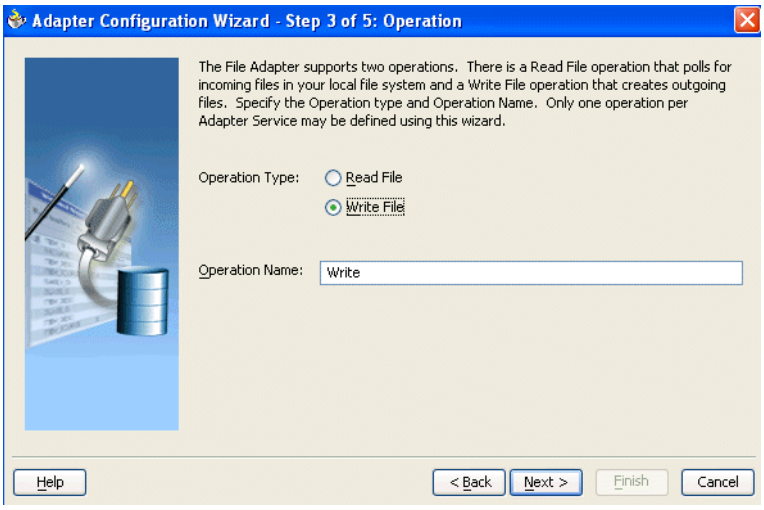
This section contains the following topics:

- Outbound Operation
- Outbound File Directory Creation
- Native Data Translation
- Error Handling
- Outbound Service Name WSDL File
- Outbound Header WSDL File

Outbound Operation

For outbound operations with the file adapter, you select to perform an outbound **Write File** operation. Figure 2–6 shows this selection.

Figure 2–6 Selecting the Write File Operation

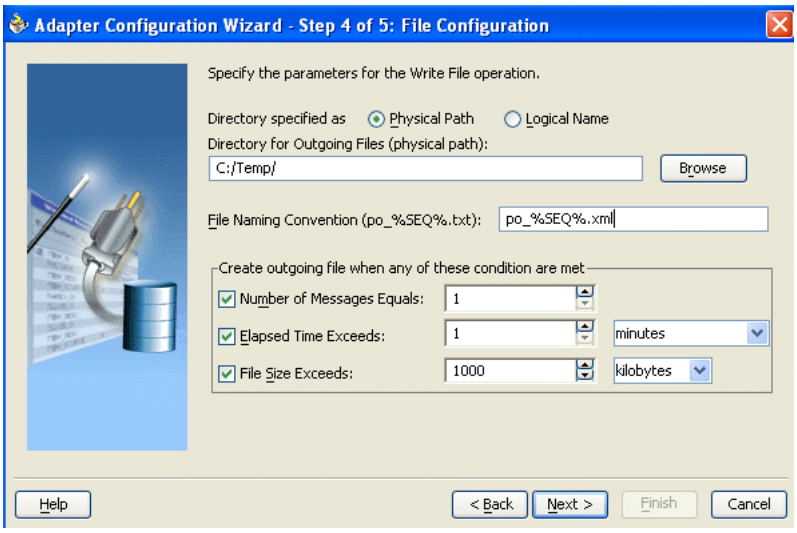


Outbound File Directory Creation

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 window of the Adapter Configuration Wizard shown in Figure 2–7 enables you to specify the directory for outgoing files and the outbound file naming convention.

Figure 2–7 Adapter Configuration Wizard—Parameters for Outgoing Files



The following sections describe the file configuration information to specify:

- Specifying Outbound Physical or Logic Directory Paths
- Specifying the Outbound File Naming Convention
- Specifying a Dynamic Outbound File Name
- Batching Multiple Outbound Messages

Specifying Outbound Physical or Logic Directory Paths You can specify outbound directory names as physical or logical paths. Physical paths are values such as `c:\outputDir`.

If you specify logical parameters, the generated WSDL file looks as follows for the logical outbound directory name `OutputFileDir`.

```
<jca:binding />
  <operation name="Write">
    <jca:operation
      InteractionSpec="oracle.tip.adapter.file.outbound.FileInteractionSpec"
      LogicalDirectory="OutputFileDir"
      FileNamingConvention="po_%SEQ%.xml">
    </jca:operation>
  .....
```

In the BPEL partner link in the `bpel.xml` file, you then specify an `outbound partner link binding` property through the **Property** tab of the partner link. This resolves the mapping between the logical directory name and the actual physical directory name.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<BPELSuitcase>
  <BPELProcess id="ComplexStructure" src="ComplexStructure.bpel">
    <partnerLinkBindings>
      <partnerLinkBinding name="InboundPL">
        <property name="wsdlLocation">ComplexStructureInbound.wsdl</property>
      </partnerLinkBinding>
    <partnerLinkBinding name="OutboundPL">
      <property name="wsdlLocation">ComplexStructureOutbound.wsdl</property>
      <property name="OutputFileDir">C:/ora_
home/integration/bpm/samples/tutorials/
```

```
121.FileAdapter/ComplexStructure/outputDir/</property>
  </partnerLinkBinding>
</partnerLinkBindings>
    <activationAgents>
    <activationAgentclassName=
"oracle.tip.adapter.fw.agent.jca.JCAActivationAgent"partnerLink="InboundPL">
    <property name="portType">Read_ptt</property>
   </activationAgent>
  </activationAgents>
 </BPELProcess>
</BPELSuitcase>
```

> **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 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 of outgoing files names, 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 2–7 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).

The sequence number is written to a file if the system goes down. 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 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`.

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 quotes (`'`) to avoid interpretation. The characters `"''"` represent a single quote. All other characters are not interpreted.

The Java pattern letters are defined in Table 2–3.

Table 2–3 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`

Table 2–3 (Cont.) Java Pattern Letters

Letter	Date or Time Component	Presentation	Examples
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
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, 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 amount. For parsing, the number of pattern letters is ignored unless it is needed to separate two adjacent fields.

- Year

  For formatting, if the number of pattern letters is two, 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, 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, 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 You can also use variables to specify dynamic outbound file names through use of the `OutboundHeader_msg` message name in the `fileAdapterOutboundHeader.wsdl`.

```
<definitions
     name="fileAdapter"
     targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/"
     xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/file/"
     xmlns="http://schemas.xmlsoap.org/wsdl/" >
    <types>
        <schema attributeFormDefault="qualified" elementFormDefault="qualified"
                targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/"
                xmlns="http://www.w3.org/2001/XMLSchema"
                xmlns:FILEAPP="http://xmlns.oracle.com/pcbpel/adapter/file/">
            <element name="OutboundFileHeaderType">
                <complexType>
                    <sequence>
                        <element name="fileName" type="string"/>
                    </sequence>
```

```
            </complexType>
          </element>
        </schema>
    </types>
    <!-- Header Message -->
    <message name="OutboundHeader_msg">
      <part element="tns:OutboundFileHeaderType" name="outboundHeader"/>
   </message>
</definitions>
```

Create a variable of message type `OutboundHeader_msg`, assign it a value, and use it in the **Adapter** tab of an invoke activity. For example:

```
<variables>
     <variable name="fileHeader" messageType="file:OutboundHeader_msg"/>
    [..]

    <assign>
      <copy>
        <from>*testfile.txt*</from>
        <to variable="fileHeader" part="outboundHeader"
            query="/file:OutboundFileHeaderType/file:fileName"/>
      </copy>
    </assign>

     <invoke name="FileSend" partnerLink="outboundPL"
             portType="out:FileWrite_PortType" operation="Write"
             inputVariable="payload"
    bpelx:inputHeaderVariable="fileHeader"/>
```

See the online Help that is included with the **Adapter** tab in JDeveloper BPEL Designer for more information.

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 enables you to specify the number of messages in one batch file to publish. The following batch file settings are provided in the File Configuration window shown in Figure 2–7 on page 2-20:

- Number of messages equals

  Specify a value that, when equaled, causes a new outgoing file to be created.

- Elapsed time exceeds

  Specify a time that, when equaled, causes a new outgoing file to be created.

- File size exceeds

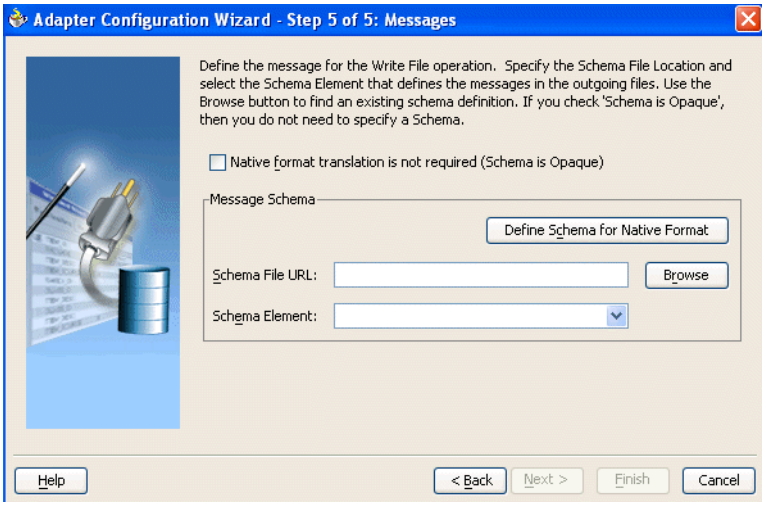
  Specify a file size that, when equaled, causes a new outgoing file to be created. For example, assume you specify a value of three for the number of messages received and a value of 1 MB for the file size. When you receive two messages that when combined equal or exceed 1 MB, or three messages that are less than 1 MB, an output file is created.

If the file adapter goes down during batching, it starts batching at the point at which it left off upon recovery.

Native Data Translation

The next Adapter Configuration Wizard window that appears is the Messages window shown in Figure 2–8. This window enables you to select the XSD schema file for translation.

Figure 2–8 Specifying the Schema



As with specifying the schema for the inbound direction, you can perform the following tasks in this window:

- 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

See "Native Data Translation" on page 2-11 for more information about the Messages window.

Error Handling

As with inbound files, the file adapter guarantees once-only delivery of outbound files. This includes guaranteed delivery of large files through FTP. If your system goes down, the write functionality of the file adapter upon restart has the knowledge to proceed from where it left off in the previous run, thereby avoiding duplicate messages. If the target host is unavailable, the file adapter supports retrying to send documents. For example, if the directory to which it is trying to write is read only, the file adapter tries to write again. You must configure two partner link retry properties from the **Property** tab of the partner link (updates the `bpel.xml` file):

```
<partnerLinkBinding name="WriteFile">
  <property name="wsdlLocation">WriteFile.wsdl</property>
  <property name="retryMaxCount">10</property>
  <property name="retryInterval">60</property>
```

The write operation eventually succeeds (if the problem is resolved in a timely fashion). If the problem is not resolved within the retry period, a binding fault is sent back to the BPEL process.

Outbound Service Name WSDL File

When you complete configuration of the file adapter with the Adapter Configuration Wizard, a WSDL file is generated for the outbound direction. The file is named after the service name you specified on the Service Name window of the Adapter Configuration Wizard shown in Figure 1–4 on page 1-3. You can rerun the wizard at any time to change your operation definitions.

The `InteractionSpec` parameters in the WSDL file contain the outbound configuration information that you specified with the Adapter Configuration Wizard during design time. The `InteractionSpec` parameters and a set of outbound file adapter properties are part of the outbound WSDL file. The outbound WSDL includes the following information:

```
<jca:binding />
  <operation name="Write">
    <jca:operation
      InteractionSpec="oracle.tip.adapter.file.outbound.FileInteractionSpec"
      PhysicalDirectory="C:/ora_
home/integration/bpm/samples/tutorials/121.FileAdapter/ComplexStructure/outputDir/
"
      FileNamingConvention="po_%SEQ%.xml"
      NumberMessages="1"
      ElapsedTime="60"
      FileSize="1000000"
      OpaqueSchema="false" >
    </jca:operation>
    <input>
      <jca:header message="hdr:OutboundHeader_msg" part="outboundHeader"/>
    </input>
  </operation>
</binding>
```

The outbound file adapter uses the following configuration parameters:

- `PhysicalDirectory`

  This parameter specifies the physical directory in which to write output files. The parameter is of type `String`. The outbound directory where the outgoing files are written is mandatory. You must specify the physical directory or logical directory.

- `LogicalDirectory`

  This parameter specifies the logical directory in which to write output files. The parameter is of type `String`.

- `NumberMessages`

  This parameter is used for outbound batching. The outgoing file is created when the number of messages condition is met. The parameter is of type `String` and is not mandatory. The default value is `1`.

- `ElapsedTime`

  This parameter is used for outbound batching. When the time specified elapses, the outgoing file is created. The parameter is of type `String` and is not mandatory. The default value is `1` minute.

- `FileSize`

  This parameter is used for outbound batching. The outgoing file is created when the file size condition is met. The parameter is of type `String` and is not mandatory. The default value is `1000` kb.

- `FileNamingConvention`

  This parameter is for the naming convention for the outbound `write` operation file. The parameter is of type `String` and is mandatory.

Outbound Header WSDL File

The WSDL file shown in "Outbound Service Name WSDL File" on page 2-26 includes two attributes that indicate which message and part define the operation headers:

```
<jca:header message="hdr:OutboundHeader_msg" part="outboundHeader"/>
```

The `fileAdapterOutboundHeader.wsdl` file defines these attributes, as well as information about the outbound file name. This file is created along with the service name WSDL file, and displays in the **Applications Navigator** of JDeveloper BPEL Designer.

```
<definitions
     name="fileAdapter"
     targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/"
     xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/file/"
     xmlns="http://schemas.xmlsoap.org/wsdl/" >
    <types>
        <schema attributeFormDefault="qualified" elementFormDefault="qualified"
                targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/"
                xmlns="http://www.w3.org/2001/XMLSchema"
                xmlns:FILEAPP="http://xmlns.oracle.com/pcbpel/adapter/file/">
            <element name="OutboundFileHeaderType">
                <complexType>
                    <sequence>
                        <element name="fileName" type="string"/>
                    </sequence>
                </complexType>
            </element>
        </schema>
    </types>

    <!-- Header Message -->
    <message name="OutboundHeader_msg">
      <part element="tns:OutboundFileHeaderType" name="outboundHeader"/>
   </message>
</definitions>
```

FTP Adapter for Get File Concepts

In the inbound direction, the FTP adapter works the same as the Read File operations of the file adapter in that it polls and gets files from a file system for processing. The major difference is that the FTP adapter is used for remote file exchanges. Because of this, the Adapter Configuration Wizard asks for connection information to an FTP server to be used later, as shown in Figure 2–9.

Figure 2–9 Specifying FTP Server Connection Information

To create the FTP server connection that you specify in Figure 2–9, you must edit the *Oracle_Home*`\integration\orabpel\system\appserver\oc4j\j2ee\home\application-deployments\default\FtpAdapter\oc4j-ra.xml` deployment descriptor file for adapter instance JNDI name and FTP server connection information. A sample of `oc4j-ra.xml` is as follows:

```
<?xml version="1.0"?>
<!DOCTYPE oc4j-connector-factories PUBLIC "-//Oracle//DTD Oracle Connector
  9.04//EN" "http://xmlns.oracle.com/ias/dtds/oc4j-connector-factories-9_04.dtd">

<oc4j-connector-factories>
    <connector-factory location="eis/Ftp/FtpAdapter" connector-name="FTP Adapter">
        <config-property name="host" value="stada55.us.oracle.com"/>
        <config-property name="port" value="21"/>
        <config-property name="username" value="anonymous"/>
        <config-property name="password" value="password"/>
    </connector-factory>
</oc4j-connector-factories>
```

The adapter instance JNDI name is specified as **`eis/Ftp/FtpAdapter`** and connection information such as host, port, username, and password are provided as configuration properties.

Notes:

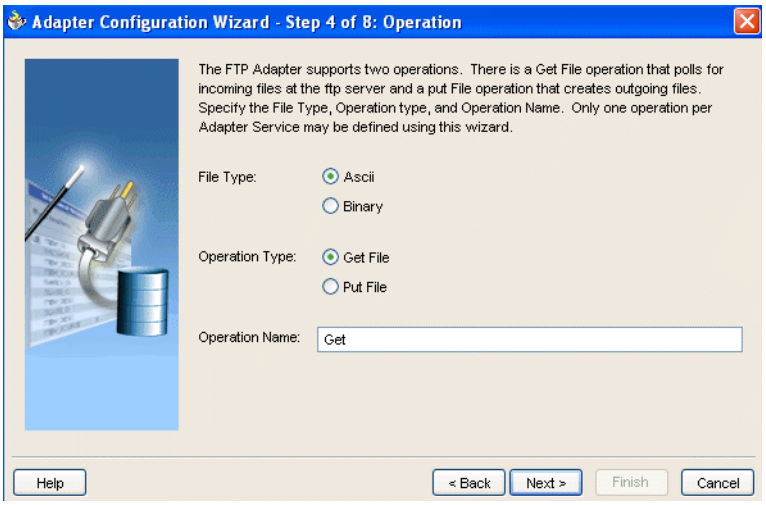
- The directory path to the `oc4j-ra.xml` file shown above is for the BPEL Process Manager for Developers installation type. If you selected the BPEL Process Manager for OracleAS Middle Tier installation type, this file is located in:

  Oracle_Home`\j2ee\OC4J_BPEL\application-deployments\default\FtpAdapter`

- The 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 2–10 shows this selection.

Figure 2–10 Selecting the Get File Operation



For inbound and outbound binary file transfers, the `Oracle_Home\integration\orabpel\system\appserver\oc4j\j2ee\home\connectors\FtpAdapter\FtpAdapter\META-INF\ra.xml` file includes a `serverLineSeparator` property. This property is automatically used to determine line separators when you transfer data in binary mode only.

```
<config-property-name>serverLineSeparator</config-property-name>
<config-property-type>java.lang.String</config-property-type>
<config-property-value>\n</config-property-value>
```

This property is not used when transferring data in ASCII mode. This is because in ASCII mode, the line separators are determined by the operating system on which the FTP server is running.

From this point onward, the windows of the Adapter Configuration Wizard window for the Get File operation are the same as those for the Read File operation of the FTP adapter. Table 2–4 lists the windows that display and provides references to sections that describe their functionality.

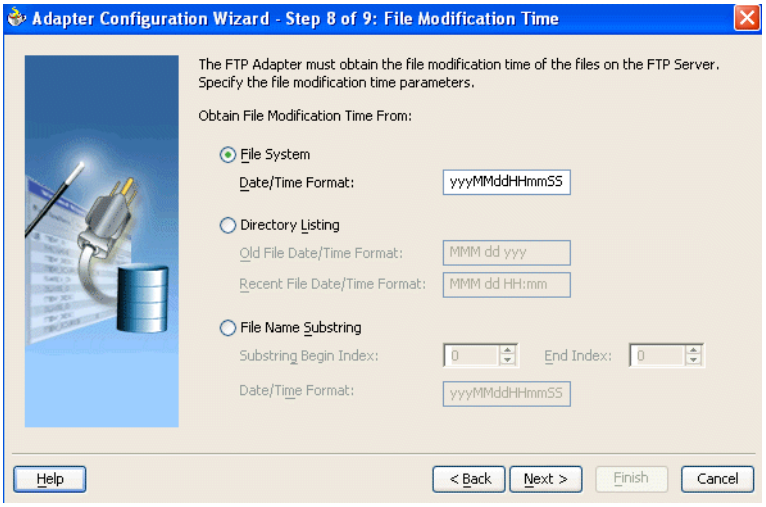
Table 2–4 Adapter Configuration Wizard Windows for Get File Operation

Window	See Section...
File Directories (Figure 2–2)	"Inbound File Directory Specifications" on page 2-5
File Filtering (Figure 2–3)	"File Matching and Batch Processing" on page 2-7
File Polling (Figure 2–4)	"File Polling" on page 2-10
Messages (Figure 2–5)	"Native Data Translation" on page 2-11

An additional Adapter Configuration Wizard window is also available for advanced users. This window is shown in Figure 2–11 and only appears after you make either or both of the following selections on the File Polling window shown in Figure 2–4 on page 2-11:

- 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 2–11 File Modification Time



This window enables you to specify one of the following methods for obtaining the modification time of files on the remote FTP server:

- **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 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 *`service_name`*`.wsdl` file includes the following parameters and values:

  ```
  FileModificationTime="FileSystem"
  ModificationTimeFormat="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 showing in the Windows task bar). You must then convert the time difference between the FTP server and the system on which the FTP adapter is running to milliseconds and add the value as a property in the `bpel.xml` file:

  ```
  <activationAgents>
  ```

```
<activationAgent ...>
  <property name="timestampOffset">2592000000</property>
```

- **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 data/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 *`service_name`*`.wsdl` file includes the following parameters and values:

  ```
  FileModificationTime="DirListing"
  ModificationTimeFormat="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 showing in the Windows task bar). You must then convert the time difference between the FTP server and the system on which the FTP adapter is running to milliseconds and add the value as a property in the `bpel.xml` file:

  ```
  <activationAgents>
    <activationAgent ...>
      <property name="timestampOffset">2592000000</property>
  ```

- **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 **Substring 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 *`service_name`*`.wsdl` file includes the following parameters and values:

  ```
  FileModificationTime="Filename"
  ```

```
    FileNameSubstringBegin="12"
    FileNameSubstringEnd="20"
    ModificationTimeFormat="yyyyMMdd"
```

When the Adapter Configuration Wizard completes, configuration files are created in the **Applications Navigator** section of JDeveloper BPEL Designer.

The inbound service WSDL file name that is created is also similar to that of the file adapter. The main differences include the operation type and the file type.

```
<pc:inbound_binding />
   <operation name="Get">
<jca:operation
   FileType="binary"
```

The inbound header WSDL file named `ftpAdapterInboundHeader.wsdl` looks as follows:

```
<definitions
     name="fileAdapter"
     targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/ftp/"
     xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/ftp/"
     xmlns="http://schemas.xmlsoap.org/wsdl/" >
    <types>
        <schema attributeFormDefault="qualified" elementFormDefault="qualified"
                targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/ftp/"
                xmlns="http://www.w3.org/2001/XMLSchema"
                xmlns:FTPAPP="http://xmlns.oracle.com/pcbpel/adapter/ftp/">
            <element name="InboundFTPHeaderType">
                <complexType>
                    <sequence>
                        <element name="fileName" type="string"/>
                        <element name="ftpHost" type="string"/>
                        <element name="ftpPort" type="string"/>
                     </sequence>
                </complexType>
            </element>
        </schema>
    </types>

    <!-- Header Message -->
    <message name="InboundHeader_msg">
      <part element="tns:InboundFTPHeaderType" name="inboundHeader"/>
   </message>
</definitions>
```

See "Error Handling" on page 2-12 and "Guaranteed Delivery and Recovery from Server Failures" on page 2-15 for more information about error handling and guaranteed delivery capabilities.

FTP Adapter for Put File Concepts

In the outbound direction, the FTP adapter works the same as the Write File operations of the file adapter in that it receives messages from the BPEL process and writes the messages in a file to a file system (in this case, remote). Because of this, the Adapter Configuration Wizard prompts you to connect to the FTP server with the adapter instance JNDI name, as shown in Figure 2–9 on page 2-28.

After logging in, you select the Put File (write) operation and the type of file to deliver. Figure 2–12 shows this selection.

Figure 2–12 Selecting the Put File Operation

From this point onward, the windows of the Adapter Configuration Wizard window for the Put File operation are the same as those for the Write File operation of the file adapter. Table 2–5 lists the windows that display and provides references to sections that describe their functionality.

Table 2–5 Adapter Configuration Wizard Windows for Put File Operation

Window	See Section...
File Configuration (Figure 2–7)	"Outbound File Directory Creation" on page 2-19
Messages (Figure 2–8)	"Native Data Translation" on page 2-25

When the Adapter Configuration Wizard completes, configuration files are created in the **Applications Navigator** section of JDeveloper BPEL Designer.

The outbound service WSDL file name that is created is also similar to that of the file adapter. The main differences include the operation type and the file type.

```
<pc:inbound_binding />
   <operation name="Put">
<jca:operation
   FileType="binary"
```

The outbound header WSDL file named `ftpAdapterOutboundHeader.wsdl` looks as follows:

```
<definitions
     name="fileAdapter"
     targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/ftp/"
     xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/ftp/"
     xmlns="http://schemas.xmlsoap.org/wsdl/" >
    <types>
        <schema attributeFormDefault="qualified" elementFormDefault="qualified"
                targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/ftp/"
                xmlns="http://www.w3.org/2001/XMLSchema"
                xmlns:FTPAPP="http://xmlns.oracle.com/pcbpel/adapter/ftp/">
            <element name="OutboundFTPHeaderType">
                <complexType>
                    <sequence>
                        <element name="fileName" type="string"/>
```

```
                    </sequence>
                </complexType>
            </element>
        </schema>
    </types>

    <!-- Header Message -->
    <message name="OutboundHeader_msg">
      <part element="tns:OutboundFTPHeaderType" name="outboundHeader"/>
   </message>
</definitions>
```

See "Error Handling" on page 2-25 for more information about error handling capabilities.

Using Secure FTP with the FTP Adapter

The FTP adapter supports the use of the secure FTP feature on Solaris. This section provides an overview of secure FTP functionality and describes how to install and configure this feature.

This section contains the following tasks:

- Secure FTP Overview
- Installing and Configuring OpenSSL
- Installing and Configuring vsftpd
- Creating an Oracle Wallet
- Setting Up the FTP Adapter

> **Note:** The FTP adapter supports the secure FTP feature on Solaris only.

Secure FTP Overview

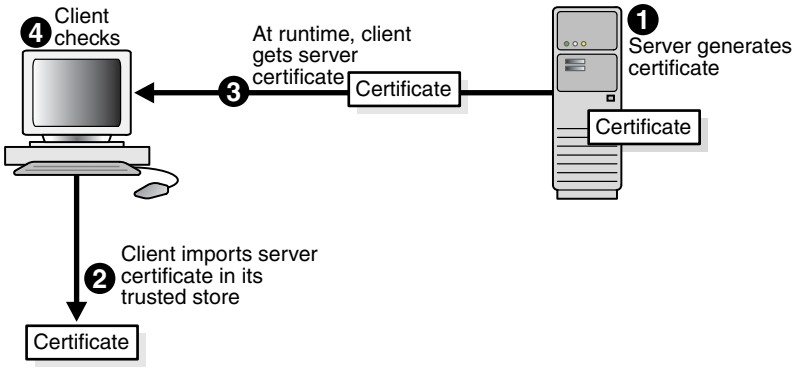
In environments in which sensitive data is transferred to remote servers (for example, sending credit card information to HTTP servers), the issue of security is very important. Security in these cases primarily refers to two requirements:

- 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 in the context of 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 vsftpd in "Installing and Configuring vsftpd" on page 2-36, you use openSSL to create a certificate for the server. Every client trusts a few parties to begin. If the server is one of these trusted parties, or if the server's certificate was issued by one of these parties, 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 2–3, the server's certificate is directly imported into the client's certificate store (or Oracle Wallet) as a trusted certificate.

Figure 2–13 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 is typically done by the client 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 using its private key and subsequently uses it to encrypt file data before sending it to the client.

The remaining subsections describe how to install and configure secure FTP.

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 host to use 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 not already using it):

```
sh
```

9. Configure and export the `PATH`:

```
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 host to use as the FTP server.

1. Go to the following location:

```
ftp://vsftpd.beasts.org/users/cevans/
```

2. Download `vsftpd-2.0.3` (you need the tar and signature file (`.asc` file)). For example:

```
[BINARY]     vsftpd-2.0.3.tar.gz. . . . . .    [Mar 19 21:26]    149K
[FILE]       vsftpd-2.0.3.tar.gz.asc. . . . . .    [Mar 19 21:26]    189B
```

3. Unzip the following file using `gunzip`.

```
gunzip vsftpd-2.0.3.tar.gz
```

4. Unzip the tar file:

```
tar xvf vsftpd-2.0.3.tar
```

5. Change directories to the following location:

```
cd vsftpd-2.0.3
```

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-2.0.3` directory structure. If you use one of those files to create the vsftpd.conf file in the `/etc` directory, ensure that it only includes the parameters and settings described above.

---

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-2.0.3` directory:

```
./vsftpd
```

Creating an Oracle Wallet

Oracle Wallet Manager is an application for managing and editing security credentials in Oracle wallets. A wallet is a password-protected container that stores authentication and signing credentials, including private keys, certificates, and trusted certificates, all of which are used by SSL for strong authentication.

1. Create a new wallet in Oracle Wallet Manager.
2. Import `vsftpd.pem` from step 11 on page 2-38 as a trusted certificate in this wallet.
3. Save this wallet in PKCS # 12 (`.p12`) format.

See the *Oracle Application Server Administrator's Guide* for details about using Oracle Wallet Manager.

Setting Up the FTP Adapter

1. Perform the following tasks on your Solaris host:

```
mkdir /var/ftp/inDir
mkdir /var/ftp/outDir
chmod 777 /var/ftp/inDir /var/ftp/outDir
```

2. Specify the FTP connection parameters in the FTP adapter `oc4j-ra.xml` file. The location of this file is based on the installation type you selected.

   - If you selected BPEL Process Manager for Developers, edit the *`Oracle_Home`*`\integration\orabpel\system\appserver\oc4j\j2ee\home\application-deployments\default\FtpAdapter\oc4j-ra.xml` file.
   - If you selected BPEL Process Manager for OracleAS Middle Tier, edit the *`Oracle_Home`*`\j2ee\OC4J_BPEL\application-deployments\default\FtpAdapter\oc4j-ra.xml` file.

   A sample `oc4j-ra.xml` is as follows:

```
<connector-factory location="eis/Ftp/FtpAdapter" connector-name="FTP Adapter">
  <config-property name="host" value="usunnbf29.us.oracle.com"/>
  <config-property name="port" value="21"/>
  <config-property name="username" value="ftp"/>
  <config-property name="password" value="password"/>
```

```
  <config-property name="useFtps" value="true"/>
  <config-property name="walletLocation" value="D:\wallet\ewallet.p12"/>
  <config-property name="walletPassword" value="welcome1"/>
  <config-property name="channelMask" value="both"/>
  <config-property name="securePort" value="990"/>
</connector-factory>
```

Where...	Is...
`useFtps`	Set to `true`. This setting is required to use FTP over SSL. The default is `false`.
`walletLocation`	The location of the wallet created in "Creating an Oracle Wallet" on page 2-38
`walletPassword`	The password of the wallet
`channelMask`	The type of channel: control channel or data channel. Possible values are `both`, `control`, `data`, or `none`. The default is `both`. ***Author's Note: I lower cased the values, to match the code sample above.***
`securePort`	The port for FTP over SSL. The default is `990`.

3. Restart Oracle BPEL Server after changing the `oc4j-ra.xml` file.

   You have now installed and configured secure FTP and are ready to use this feature with the FTP adapter.

Use Cases for the File and FTP Adapters

Oracle BPEL Process Manager includes a number of demonstrations of file and FTP adapters. Some of these demonstrations are described in Readme files. Others are described in the Oracle BPEL Process Manager documentation set. This section provides an overview of these demonstrations and a reference to documentation that more fully describes the scenarios.

This section contains the following topics:

- File Adapter Use Cases
- FTP Adapter Use Case

File Adapter Use Cases

This section provides an overview of file adapter demonstrations.

This section contains the following topics:

- File Reading
- Message Debatching
- Reading Delimited Content Files
- Reading Positional (Fixed Length) Content Files
- File Writing

File Reading

Several file reading demonstrations are available:

- A complex structure demonstration shows how to use the file read and write functionality of the file adapter. For a demonstration of a complex structure, go to

  Oracle_Home\integration\orabpel\samples\tutorials\121.FileAdapter\ComplexStructure

- A simple file reading demonstration is provided as part of a larger tutorial that guides you through the design and execution of a sophisticated process that uses synchronous and asynchronous services, parallel flows of execution, conditional branching logic, fault handling and exceptions management, transformations, the file adapter, the database adapter, and human workflow, notification, and sensor functionality. In the file reading portion, you configure the file adapter to read an inbound purchase order request from a file in a directory. See Oracle BPEL Process Manager Order Booking Tutorial for a tutorial that uses the file reading functionality of the file adapter.

Message Debatching

This demonstration shows how the file adapter processes native data containing multiple messages defined in a custom format. The file adapter takes a single inbound file of two records and outputs each record to its own file. For a demonstration of message debatching, go to

Oracle_Home\integration\orabpel\samples\tutorials\121.FileAdapter\Debatching

Reading Delimited Content Files

This demonstration shows how the file adapter reads CSV-formatted entries in an address book, transforms the file contents using XSLT, and stores the data in a fixed-length formatted file. For a demonstration of delimited content files, go to

Oracle_Home\integration\orabpel\samples\tutorials\121.FileAdapter\FlatStructure

Reading Positional (Fixed Length) Content Files

This demonstration shows how the file adapter reads CSV-formatted entries in an address book, transforms the file contents using XSLT, and stores the data in a fixed-length formatted file. For a positional (fixed length) demonstration, go to

Oracle_Home\integration\orabpel\samples\tutorials\121.FileAdapter\FlatStructure

File Writing

A simple file writing demonstration is provided as part of the larger tutorial described in "File Reading" on page 2-39. In the file writing portion, you configure the file adapter to write an outbound purchase order acknowledgment to a file in a directory.

See Oracle BPEL Process Manager Order Booking Tutorial for a demonstration that uses the file writing functionality of the file adapter.

FTP Adapter Use Case

This demonstration shows how the file adapter processes native data containing multiple messages defined in a custom format. The native data instance contains an invoice and purchase order.

In the inbound direction, the FTP adapter retrieves a remote file, processes the file, and publishes the invoice and purchase order separately to the debatching BPEL process.

In the outbound direction, only purchase orders are generated. The debatching BPEL process transforms an invoice to a purchase order. The purchase record is simply

copied over. All purchase orders are then written in separate remote output files. For an FTP demonstration, go to

Oracle_Home\integration\orabpel\samples\tutorials\129.FTPAdapter\FTPDebatching

Summary

This chapter describes how you use the file and FTP adapters so that your BPEL process can exchange (read and write) files (both XML and non-XML) on local file systems and remote file systems (through FTP).

3

Oracle Application Server Adapter for Advanced Queuing

This chapter describes how to use the Oracle Application Server Adapter for Advanced Queuing (AQ adapter), which enables a BPEL process to interact with a queue.

This chapter contains the following topics:

- Introduction to the AQ Adapter
- AQ Adapter Features
- Use Cases for the AQ Adapter
- Summary

Introduction to the AQ Adapter

Oracle Streams Advanced Queuing (AQ) provides a flexible mechanism for bidirectional, asynchronous communication between participating applications. Because advanced queues are an Oracle database feature, they are scalable and reliable. Backup and recovery (including any-point-in-time recovery), logging, transactional services, and system management are all inherent features of the database and, therefore, 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. See *Oracle Streams Advanced Queuing User's Guide and Reference* for more information.

AQ Adapter Features

The AQ adapter is both a producer and 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 AQ adapter supports both ADT (Oracle object type) and RAW queues as payloads. It also supports extracting a payload from one ADT member column. The AQ adapter does not support the AQ XML type.

The AQ adapter supports headers for enqueue and dequeue options. Headers are comprised of an AQ header and a payload header. The AQ header consists of the standard message properties that are present in every queue.

You access the AQ adapter from the Adapter Configuration Wizard, which you use to browse queues and expose the underlying metadata as a WSDL with JCA extensions.

For AQ adapter samples, go to

Oracle_Home\integration\orabpel\samples\tutorials\124.AQAdapter

Enqueue-Specific Features (Message Production)

The 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.

- Multiconsumer queue

  In Oracle Streams AQ, more than one consumer can process and consume a single message. To use this feature, you must create multiconsumer queues and enqueue the messages into these queues. In this configuration, a single message can be consumed by more than one 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 AQ adapter enqueue header 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 multiconsumer queue must have unique values for the `Recipient` parameter. This means that 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, 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 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. The delay

interval, expiration time, and exception queue property are properties of the AQ adapter enqueue header.

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 AQ adapter supports a polling mechanism for consuming AQ messages.

The AQ adapter supports the following features of Oracle Streams AQ:

- Multiconsumer queue

  The AQ adapter can retrieve messages from a multiconsumer 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 AQ adapter dequeue header. If the number of retries exceeds the limit, the message is moved to an exception queue that you specify. The exception queue is a property of the 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 (currently priority and correlation ID), user data properties (object payloads only), and functions (as specified in the `WHERE` clause of a SQL query).

- AQ headers

  Table 3–1 summarizes the AQ header properties for the enqueue and dequeue operations.

Table 3–1 AQ Header Message Properties

Message Property	Datatype	Default If Not Specified	Description	Include in Message Header for Enqueue	Include in Message Header for Dequeue
`Priority`	Integer	1	Priority of the message. A smaller number indicates a higher priority.	Yes	Yes
`Delay`	Integer	0	The number of seconds after which the message is available for dequeuing	Yes	No
`Expiration`	Integer	-1 (never expires)	The number of seconds before the message expires. This parameter is an offset from the delay.	Yes	No
`Correlation`	String	-	User-assigned correlation ID	Yes	Yes
`RecipientList`	String	-	The list of recipients for this message, separated by commas. This overrides `RecipientList` in the `InteractionSpec`	Yes	No
`ExceptionQueue`	String	-	The exception queue name	Yes	No
`EnqueueTime`	String	-	The time at which the message was enqueued	No	Yes
`MessageID`	String	-	The hexadecimal representation of the message ID	No	Yes
`OrigMessageId`	String	-	The hexadecimal representation of the original message ID	No	Yes
`Attempts`	Integer	-	The number of failed attempts at dequeuing the message	No	Yes

See Oracle Application Server Adapter Concepts for information on AQ adapter architecture, adapter integration with Oracle BPEL Process Manager, and adapter deployments.

Supported ADT Payload Types

The AQ adapter supports the following RAW types:

- `BLOB`
- `CHAR`
- `CLOB`
- `DATE`
- `DECIMAL`
- `DOUBLE PRECISON`
- `FLOAT`
- `INTEGER`
- `NUMBER`
- `REAL`
- `SMALLINT`

- TIMESTAMP
- VARCHAR2

In addition, the primitive types, varrays of objects, or primitives are also supported.

If choosing a payload field instead of the whole ADT, choose only one of the following datatypes as the payload field:

- CLOB, either XSD or opaque schema
- VARCHAR2, either XSD or opaque schema
- BLOB, opaque schema only

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. See Chapter 6, "Native Format Builder Wizard" for more information. For Native Format Builder examples, go to

Oracle_Home\integration\orabpel\samples\tutorials\124.AQAdapter\RawQueuePayloadUsingNativeFormat

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 the ADT with BLOB selected as the payload case, an opaque schema must be used, which is defined as:

```
<element name="opaqueElement" type="base64Binary" />
```

In all other cases, you can either provide a schema or use an opaque schema, as shown in Table 3–2.

Table 3–2 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

If you do not have an XSD, but the payload data is formatted in some way (for example, in a comma-separated value (CSV) format), then the Native Format Builder Wizard can be used 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.

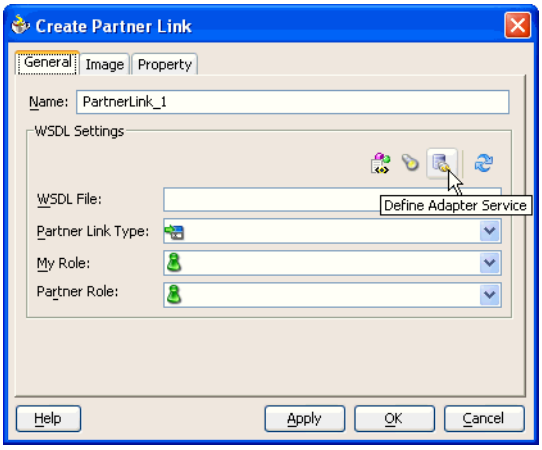
Use Cases for the AQ Adapter

The following use cases include a general walkthrough of the Adapter Configuration Wizard, followed by examples of how you modify the general procedure in different situations. Each example shows relevant parts of the generated WSDL file.

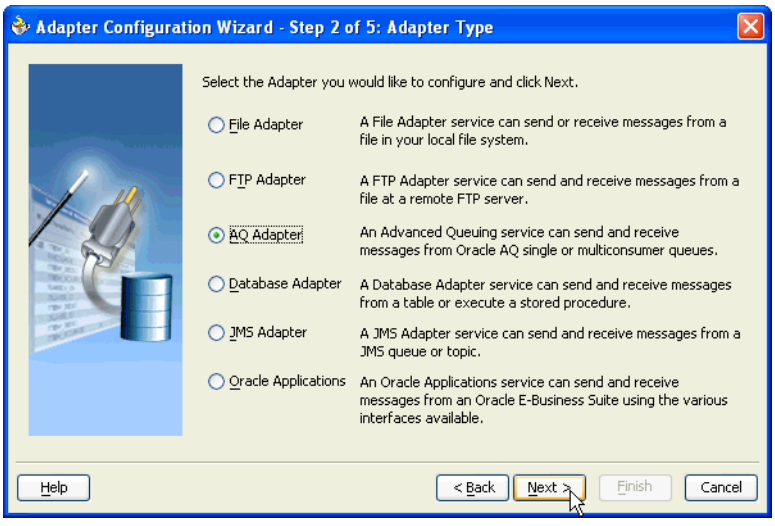
Adapter Configuration Wizard Walkthrough

This procedure shows how to use the Adapter Configuration Wizard. The example creates an adapter service that enqueues messages to the `CUSTOMER_IN_QUEUE` queue, with a payload that is one field within the `CUSTOMER_TYPE` object (the customer's e-mail address), and with a user-defined schema.

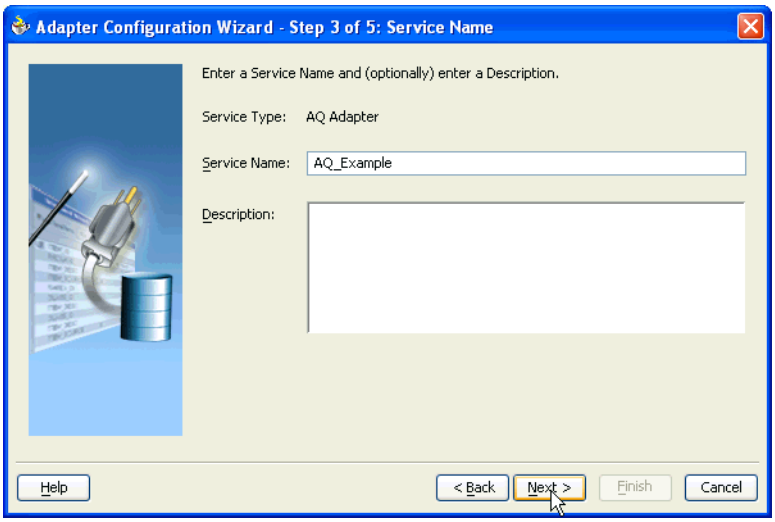
1. Drag and drop a **PartnerLink** activity onto the right side of the designer window.
2. Select **Define Adapter Service** (third icon) in the Create Partner Link window.



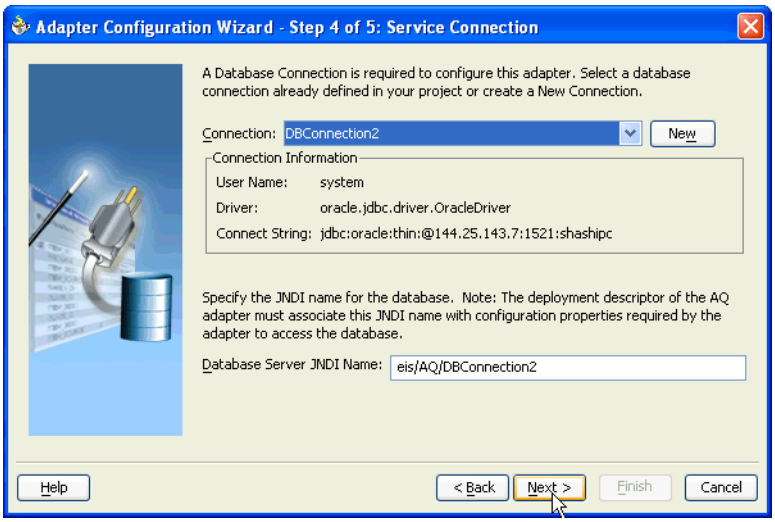
3. Select **AQ Adapter** from the Adapter Type window and click **Next**.



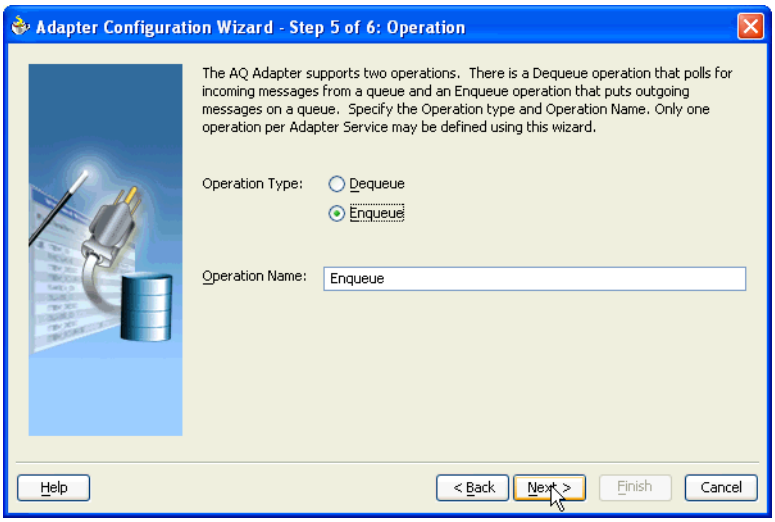
4. Enter a service name (this name can be anything) and click **Next**.



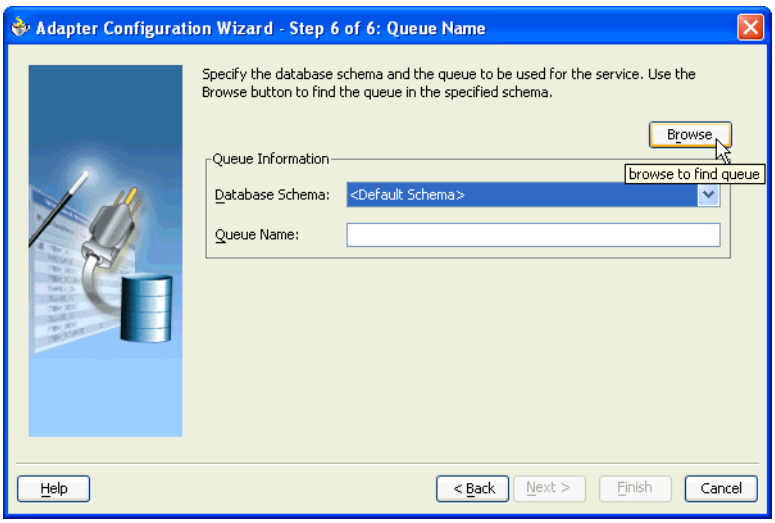
5. Select a database connection and click **Next**. This name must typically match with your `connector-factory location` setting in the `oc4j-ra.xml` file. Ensure that you restart Oracle BPEL Server after changing the `oc4j-ra.xml` file.



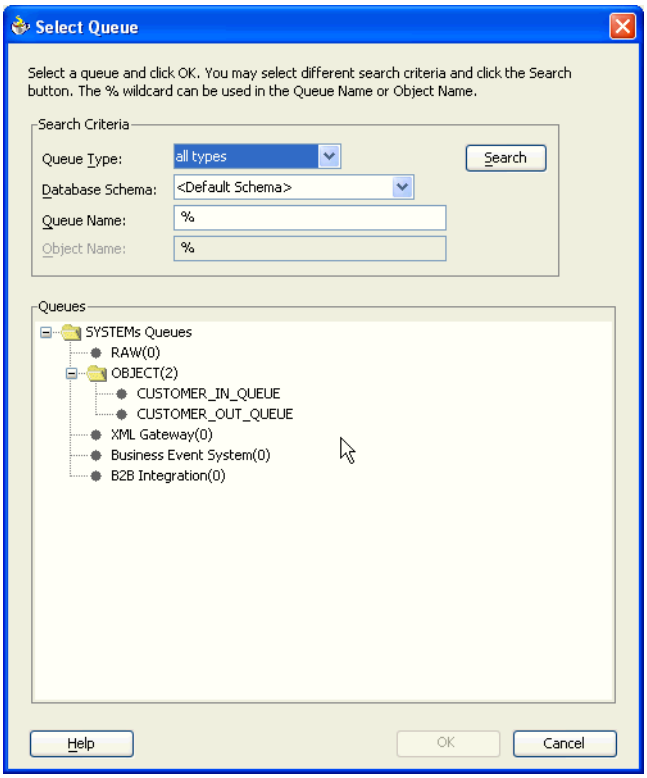
6. Select **Enqueue**, to send messages to a queue, or **Dequeue**, to receive messages from a queue. Click **Next**.



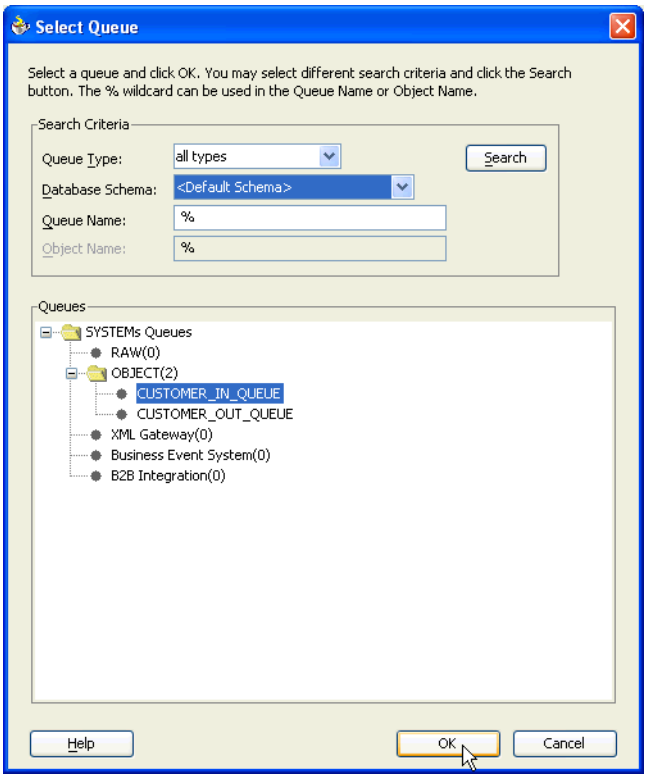
7. Select a database schema or click **Browse** to browse for the correct schema. The **<Default Schema>** selection is the schema owned by the connection user. For example, if you connect as **scott**/**tiger**, then the **<Default Schema>** is **scott**; you see all of the queues of **scott**. In general, you only see schemas and queues that the connection user has the privileges to see. For this example, **Browse** is selected.



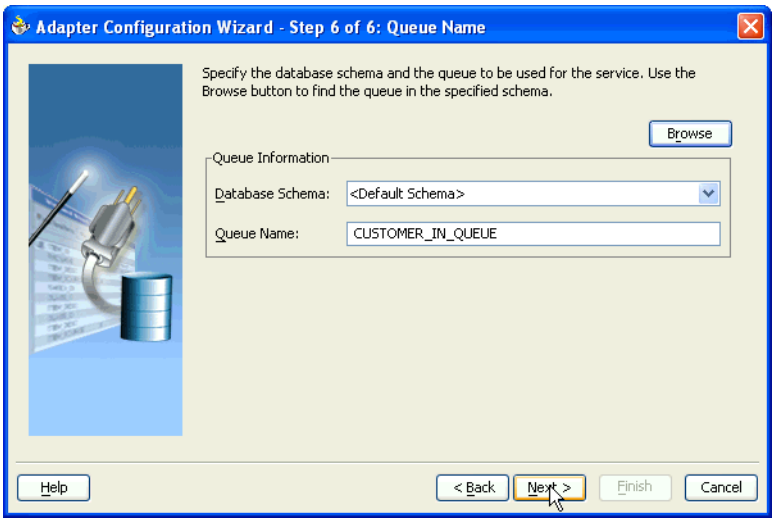
The Select Queue window appears. Note that **all types** is selected. This means that all available types display: **RAW**, **OBJECT**, and special OBJECT (**XML Gateway**, **Business Event System**, and **B2B Integration**) queues.



8. Select the correct queue and click **OK**.



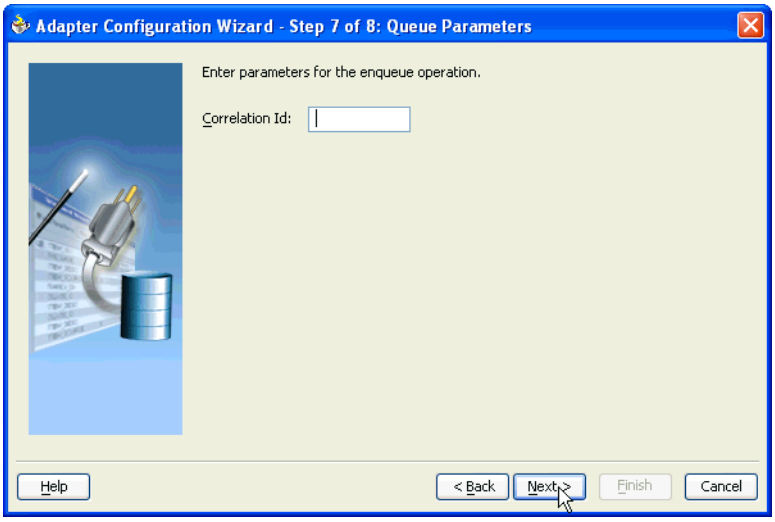
9. Verify that you have selected the database schema and click **Next**.



10. Enter an optional correlation ID from 1 to 30 characters in length and click **Next**. 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. 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, all messages in the queue are processed.

    If you enter a value for the Correlation ID in the outbound direction, 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.

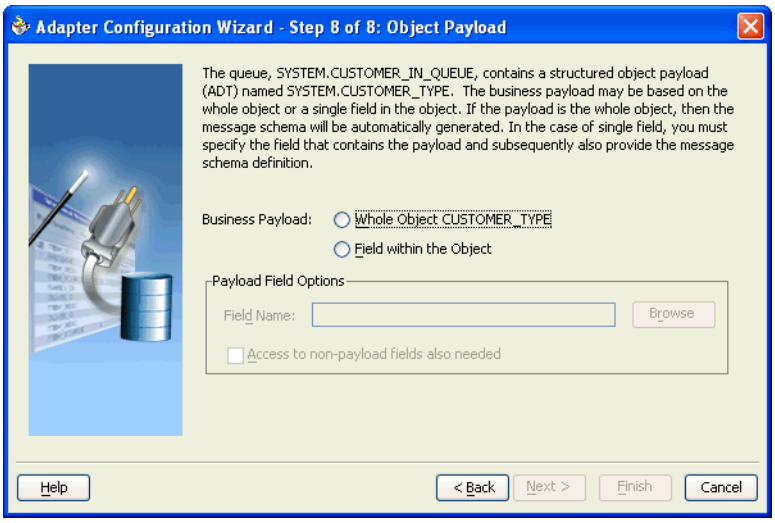


11. Select the business payload, either the entire object, or just one field within the object:

    - If you select **Whole Object CUSTOMER_TYPE**, click **Next** and go to Step 14. The message schema is automatically generated.
    - If you select **Field within the Object**, the business payload is contained in a single field in the object. Specify the correct file name, either by entering the

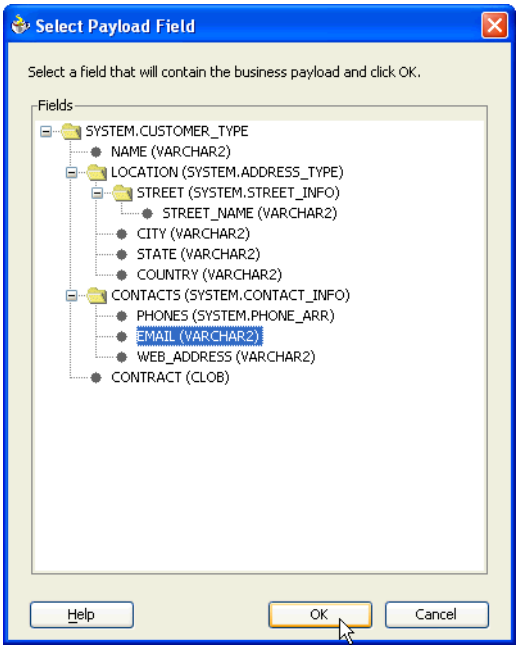
name or by browsing for it using the **Browse** button. The field containing the payload you select must be a CLOB, BLOB, or VARCHAR2.

The **Access to non-payload fields also needed** check box is available as an option. Select this check box if you need to specify additional information in a header field that is separate from the object payload. For example, your payload may be a JPG image. You may want to specify a person's name in the nonpayload field. This selection generates an additional header schema file (*object_name*`.xsd`, where *object_name* is the structured payload object used by the queue). Using this information is discussed in more detail in "Rule-Based Subscription for Multiconsumer Queues" on page 3-19.

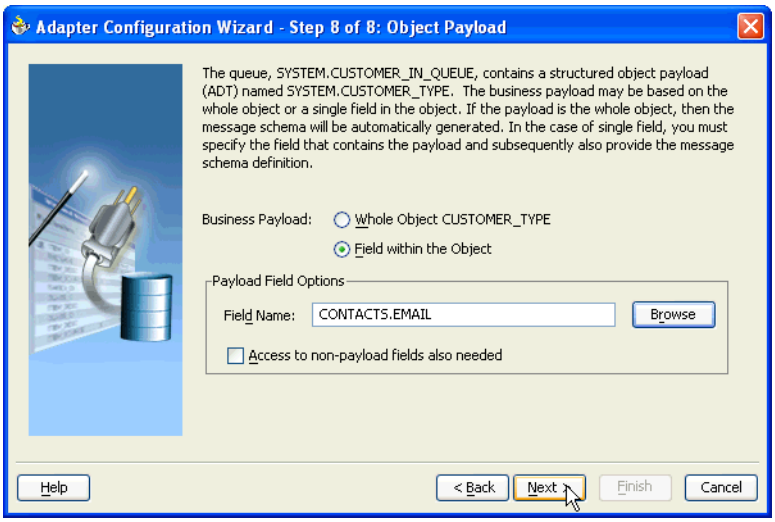


12. If you select the **Browse** button, the Select Payload Field window appears. Select the correct field name and click **OK**.

Figure 3–1 Select Payload Field Window



13. Click **Next**.

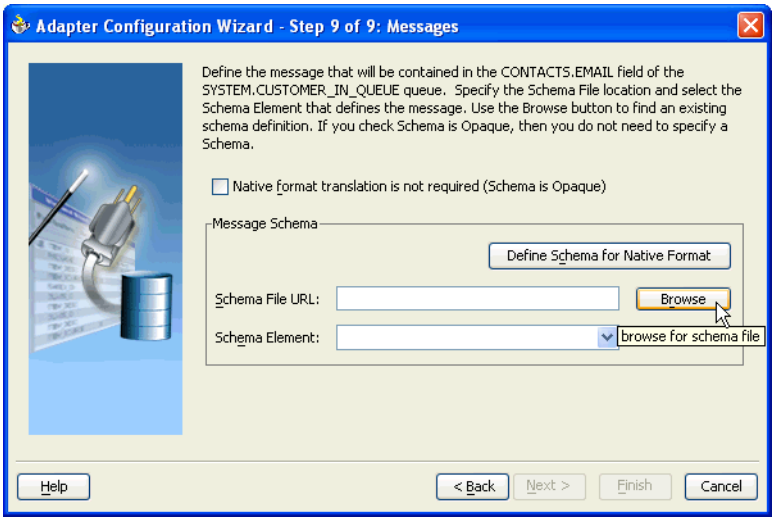


The **Messages** window appears. These settings define the correct schema for the message payload.

14. Perform one of the following tasks:

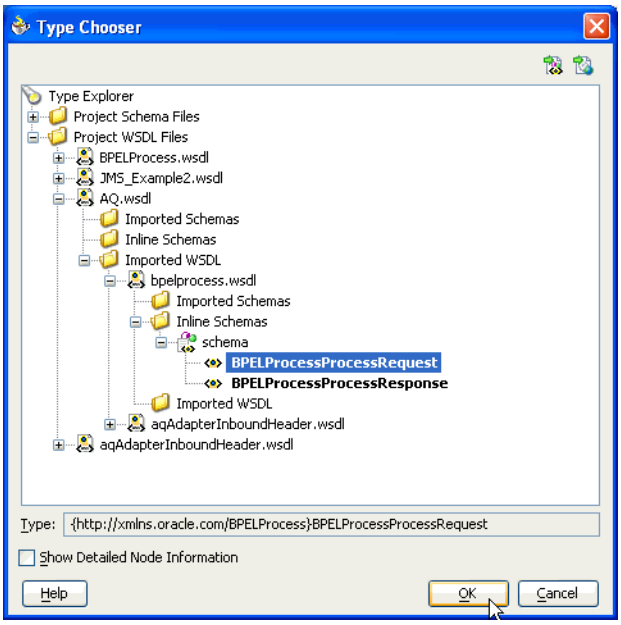
    - Check **Native format translation is not required (Schema is Opaque)**, which disables the rest of the fields.
    - Click **Define Schema for Native Format** to start the Native Format Builder Wizard, if the schema you want to use is not a native schema file. This wizard guides you through the creation of a native schema file from file formats such as comma-separated value (CSV), fixed-length, data type description (DTD), and COBOL Copybook.
    - 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. Click the **Browse** button.

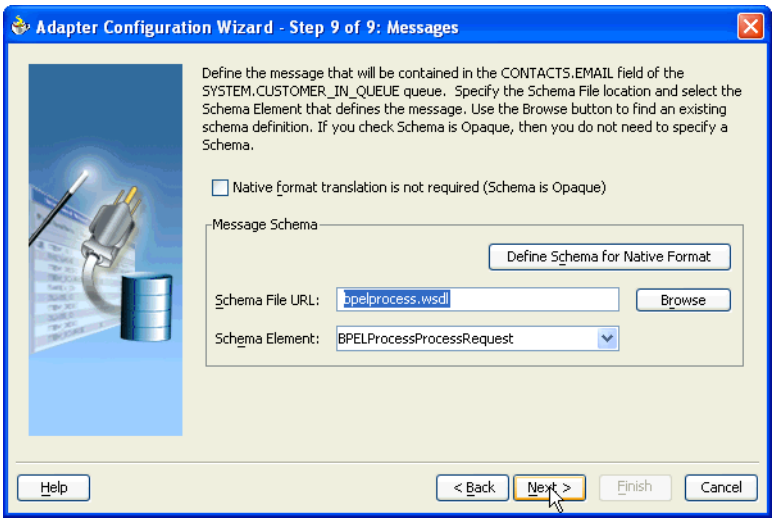


The **Type Chooser** window appears, with the Type Explorer navigation tree.

15. Browse the tree and select the appropriate schema type. Click **OK**.

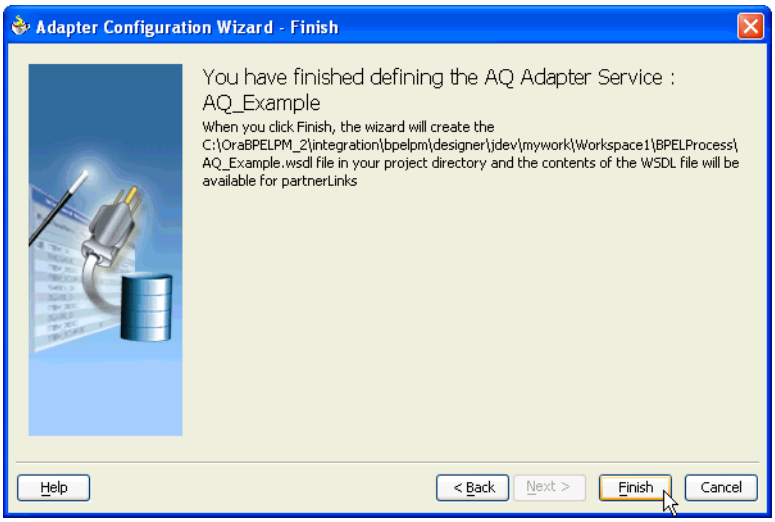


The Messages window reappears, with the **Schema File URL** field completed.



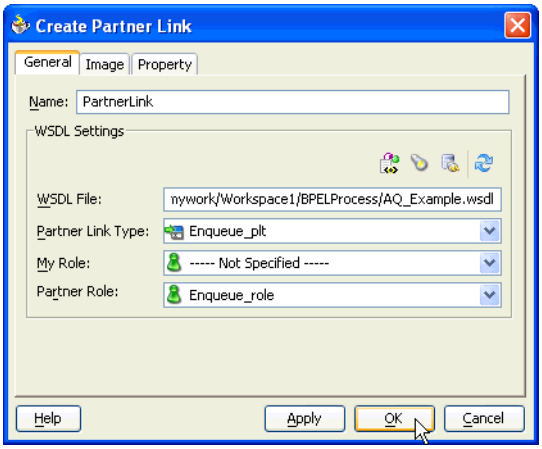
16. Click **Next**.

    The **Finish** window appears. This box shows the path and name of the adapter file that the wizard creates.



17. Click **Finish**.

    The Create Partner Link window appears with the fields completed.



18. Click **OK**.

Generated WSDL file

The adapter service generates a WSDL file to serve as the defined adapter interface. Here is the WSDL file generated by the walkthrough example.

This part of the code segment defines the name of the adapter, and the locations of various necessary schemas and other definition files.

```
<definitions
     name="AQ_Example"
     targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/AQ_Example/"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/AQ_Example/"
     xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
     xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
     xmlns:impl="http://xmlns.oracle.com/BPELProcess"
     xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/aq/outbound/"
    >
```

This code segment imports the necessary namespaces.

```
    <import namespace="http://xmlns.oracle.com/pcbpel/adapter/aq/outbound/"
location="aqAdapterOutboundHeader.wsdl"/>
    <import namespace="http://xmlns.oracle.com/BPELProcess"
location="bpelprocess.wsdl"/>
```

This code segment defines the message name and port type for the partner link.

```
    <message name="BPELProcessProcessRequest_msg">
        <part name="BPELProcessProcessRequest"
element="imp1:BPELProcessProcessRequest"/>
    </message>
    <portType name="Enqueue_ptt">
        <operation name="Enqueue">
            <input message="tns:BPELProcessProcessRequest_msg"/>
        </operation>
    </portType>
```

This code segment defines the necessary bindings for the enqueue operation and the target queue, and identifies the message header.

```
    <binding name="Enqueue_binding" type="tns:Enqueue_ptt">
    <jca:binding />
        <operation name="Enqueue">
      <jca:operation

InteractionSpec="oracle.tip.adapter.aq.outbound.AQEnqueueInteractionSpec"
          QueueName="CUSTOMER_IN_QUEUE"
          ObjectFieldName="CONTACTS.EMAIL"
          OpaqueSchema="false" >
      </jca:operation>
      <input>
        <jca:header message="hdr:Header" part="Header"/>
      </input>
        </operation>
    </binding>
    <service name="AQ_Example">
        <port name="Enqueue_pt" binding="tns:Enqueue_binding">
```

This last part defines the database connection, the connection factory (as defined in the `oc4j-ra.xml` file), and the name and role of the `partnerLinkType` and `portType`.

```
<!--Your runtime connection is declared in
J2EE_HOME/application-deployments/default/DbAdapter/oc4j-ra.xml
These mcf properties here are from your design time connection and
save you from having to edit that file and restart the application server
if eis/AQ/DBConnection2 is missing.

These mcf properties are safe to remove.-->
      <jca:address location="eis/AQ/DBConnection2"
UIConnectionName="DBConnection2"

ManagedConnectionFactory="oracle.tip.adapter.aq.AQManagedConnectionFactory"
          mcf.ConnectionString="jdbc:oracle:thin:@144.25.143.7:1521:shashipc"
          mcf.UserName="system"
          mcf.Password="47E570316F19A1CFFD2E2104BF5CA8FE" />
        </port>
    </service>
  <plt:partnerLinkType name="Enqueue_plt" >
    <plt:role name="Enqueue_role" >
      <plt:portType name="tns:Enqueue_ptt" />
    </plt:role>
```

```
  </plt:partnerLinkType>
</definitions>
```

Dequeuing and Enqueuing Object and ADT Payloads

The Adapter Configuration Wizard walkthrough shows how to enqueue one field in an object. To enqueue or dequeue the entire object as the payload, do the following:

- Select **Enqueue** or **Dequeue** in Step 6 on page 3-7.
- Select **Whole Object CUSTOMER_TYPE** in Step 11 on page 3-10, and skip to Step 16 on page 3-13.

The Adapter Configuration Wizard walkthrough provided an enqueue operation example. For a dequeue operation, the resulting WSDL file differs by including dequeue information instead of enqueue information, and by lacking an `ObjectFieldName` in the `jca:Operation` section of the code. See the following code sample for a dequeue operation and compare it to the WSDL file from the general walkthrough to see these differences:

```
<portType name="Dequeue_ptt">
    <operation name="Dequeue">
        <input message="tns:CUSTOMER_TYPE_msg"/>
    </operation>
</portType>
<binding name="Dequeue_binding" type="tns:Dequeue_ptt">
<pc:inbound_binding />
    <operation name="Dequeue">
  <jca:operation
      ActivationSpec="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec"
      QueueName="CUSTOMER_OUT_QUEUE
      OpaqueSchema="false" >
  </jca:operation>
  <input>
    <jca:header message="hdr:Header" part="Header"/>
  </input>
    </operation>
```

Dequeuing One Column of the Object/ADT Payload

The walkthrough is an example of enqueuing one field or column within an object payload. To create an adapter that dequeues the one field in an object, the steps are the same, except that in Step 6 on page 3-7, select **Dequeue**.

The following segment of the generated WSDL file specifies that one field, in this case `CONTACTS.EMAIL`, is dequeued.

```
<jca:operation
    ActivationSpec="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec"
    QueueName="CUSTOMER_IN_QUEUE"
    ObjectFieldName="CONTACTS.EMAIL"
    OpaqueSchema="true" >
</jca:operation>
```

Processing Large Numbers of Messages

If you want to process large numbers of messages with the AQ adapter, ensure that you set the `cacheWSIFOperation` property to `true` in the `bpel.xml` file:

```
<BPELSuitcase>
 <BPELProcess id="HelloWorld" src="HelloWorld.bpel">
```

```
<partnerLinkBindings>
   <partnerLinkBinding name="Dequeue">
      <property name="wsdlLocation">fileService.wsdl</property>
   </partnerLinkBinding>
   <partnerLinkBinding name="Enqueue">
      <property name="wsdlLocation">fileWriteService.wsdl</property>
      *<property name="cacheWSIFOperation">true</property>*
   </partnerLinkBinding>
</partnerLinkBindings>
```

Set this property through the **Property** tab of the Create Partner Link or Edit Partner Link window for the partner link.

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 6 on page 3-7.
- Enter the correlation ID in Step 10 on page 3-10.

The adapter dequeues messages enqueued with that same correlation ID only.

The resulting WSDL file contains the correlation ID:

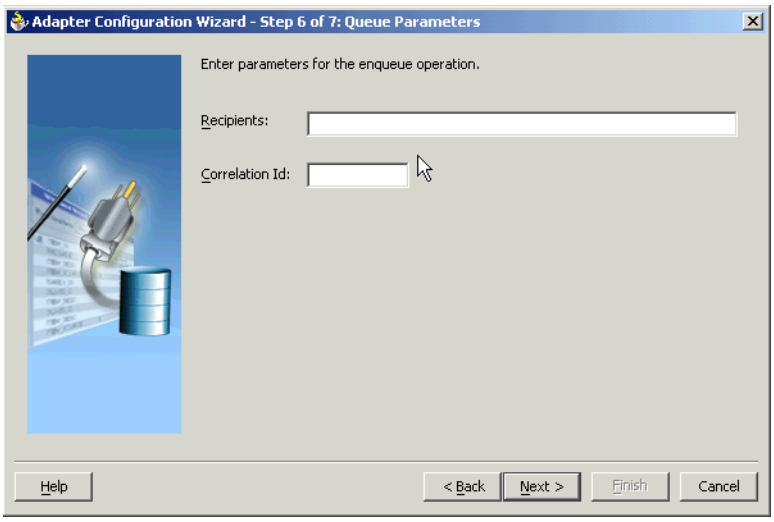
```
<jca:operation
   ActivationSpec="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec"
   QueueName="CUSTOMER_OUT_QUEUE"
   Correlation="147"
   OpaqueSchema="false" >
</jca:operation>
```

Enqueuing and Dequeuing from Multisubscriber Queues

Multisubscriber queues are accessible by multiple users, and sometimes those users are only concerned 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.

If you specify an adapter that enqueues to a multisubscriber queue in Step 8 on page 3-9, then the Queue Parameters window also includes a **Recipients** field with the **Correlation Id** field, as shown in Figure 3–2.

Figure 3–2 Queue Parameters Window with Correlation Id Field



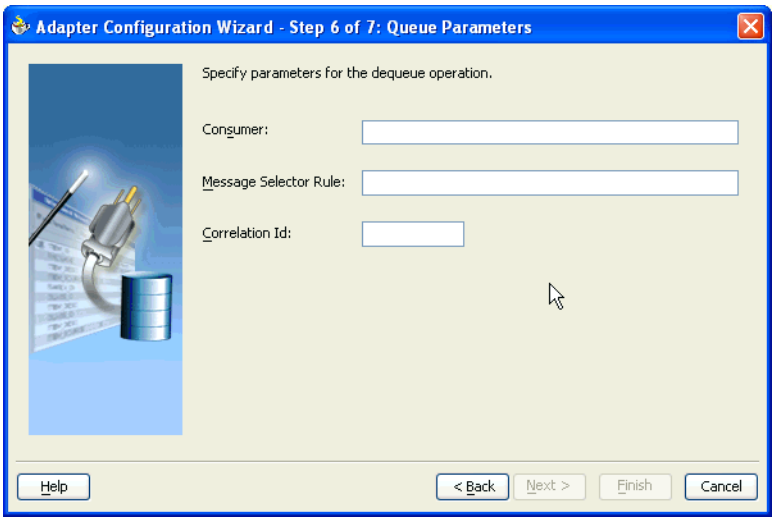
Enter the consumer name or names separated by commas that are the intended recipients for the messages enqueued by the adapter. The message remains in the queue until all recipients have dequeued the message. If the **Recipients** field is left empty, then all currently active consumers are recipients. This field can be overridden on a per message basis by setting the `RecipientList` field described in Table 3–1 on page 3-4 in the outbound header.

The following code is from a WSDL file generated by defining an AQ adapter that enqueues with a recipient list of `Bob`:

```
<jca:operation
   InteractionSpec="oracle.tip.adapter.aq.outbound.AQEnqueueInteractionSpec"
    QueueName="IP_IN_QUEUE"
    DatabaseSchema="SCOTT"
    ObjectFieldName="PAYLOAD"
    PayloadHeaderRequired="true"
    RecipientList="Bob"
    OpaqueSchema="true" >
</jca:operation>
```

When dequeuing from a multisubscriber queue, the Queue Parameters window appears, as shown in Figure 3–3.

Figure 3–3 Queue Parameters Window with Consumer Field



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 WSDL file generated by defining an AQ adapter with a consumer name:

```
<jca:operation
    ActivationSpec="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec"
    QueueName="IP_IN_QUEUE"
    DatabaseSchema="SCOTT"
    ObjectFieldName="PAYLOAD"
    PayloadHeaderRequired="true"
    Consumer="Bob"
    OpaqueSchema="true" >
</jca:operation>
<input>
  <jca:header message="tns:Header_msg" part="Header"/>
</input>
  </operation>
```

The **Message Selector Rule** field enables you to enter rules for accepting messages. This is discussed in more detail in the following section.

Correlation Id is a number assigned to a message to identify it to specific dequeuers, as mentioned in Step 10 on page 3-10. This differs from a subscription rule in that the queue stores messages with a correlation ID for later use when a subscriber using that correlation ID comes online.

Rule-Based Subscription for Multiconsumer Queues

When a dequeue is performed from a multisubscriber queue, it is sometimes necessary to screen the messages and accept only those that meet certain conditions. This condition may concern header information, as in selecting messages of only priority 1, or some aspect of the message payload, as in selecting only loan applications above $100,000.

The **Message Selector Rule** field appears in Step 10 on page 3-10 if the selected queue is multisubscriber. 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 only dequeues messages for which this Boolean expression is true.

In Step 11 on page 3-10, **Access to non-payload fields also needed** must be checked in order to access header information.

When this field is checked, the generated WSDL has additional code in the `type` section:

```
    <complexType name="HeaderCType" >
      <sequence>
        <!-- static header -->
        <element name="QueueHeader" type="hdr:HeaderType" />
        <!-- payload header -->
        <element name="PayloadHeader" type="obj1:SERVICE_TYPE" />
      </sequence>
    </complexType>
    <element name="Header" type="tns:HeaderCType" />
<element name="Header" type="tns:HeaderCType" />

and the message:
   <message name="Header_msg">
       <part name="Header" element="tns:Header"/>
   </message>
```

Note that `PayloadHeader` is the type for the whole ADT of the queue. The payload contains only the chosen payload field. If **Access to non-payload fields also needed** is checked, the `PayloadHeader` contains the whole ADT (including the payload field, which is also present in the header, but ignored by the adapter).

For working examples of rule-based subscription using header and payload information, go to

```
Oracle_Home\integration\orabpel\samples\tutorials\124.AQAdapter\
RuleBasedSubscription_Header

Oracle_Home\integration\orabpel\samples\tutorials\124.AQAdapter\
RuleBasedSubscription_Payload
```

Using AQ Headers in a BPEL Process

The entries included in AQ headers are described in "Dequeue and Enqueue Features" on page 3-3. These header entries are available for composing subscription rules.

Header information can also be translated into BPEL variables using the **Adapter** tab when defining BPEL activities that send or receive messages. This tab enables you to create header variables for use with the adapters.

You create header variables in **invoke**, **receive**, **reply**, and **pick - OnMessage** branch activities. Information passing through header variables is protocol specific.

Note that header information for messages coming from the AQ adapter is different from header information for corresponding messages flowing into the AQ adapter. You must use headers if you need to get or set any of these protocol-specific properties. The following examples describe when you must use headers:

- The AQ adapter enables you to get and set the priority of the message. For example, you want two different flows in your business process: one for high priority flows and one for low priority flows. This is possible if you use the inbound AQ header of the message flowing into the business process. In a similar

fashion, you can set the priority of an outbound message through the outbound AQ headers.

- In a file propagation scenario, files are being moved from one file system to another using the file adapter. In this case, it is imperative that you maintain file names across the two systems. Use file headers in both directions and set the file name in the outbound file header to use the file name in the inbound file header.

Header Variables in JDeveloper BPEL Designer

The following example describes how to create a special header variable in a **receive** activity in the outbound direction for the file adapter. You are *not* restricted to this direction or adapter type. You can also create this variable in either direction (inbound or outbound) with other adapter types that include headers (AQ, JMS, and FTP).

1. Click the **flashlight** icon to display the Variable Chooser window.
2. Select the second **Variables** folder, and click **Create** to display the Create Variable window.
3. Enter a unique and recognizable name in the **Name** field (for this example, **Variable_Header**).
4. Select **Message Type** and click the **flashlight** icon to display the Type Chooser window.
5. Expand **Message Types**, then **Project WSDL Files**, then ***service_name*.wsdl**, and then **Message Types**, where ***service_name*** is the name you specified for the service name when you ran the Adapter Configuration Wizard.

   A header message named **OutboundHeader_msg** (or a similar name that includes **Header_msg** as part of the name) appears.
6. If this name appears here, select it and go to Step 8.
7. If this name does not appear, perform the following additional steps.
   a. Expand **Imported WSDL**, then **fileAdapterOutboundHeader.wsdl** (or a similar name for the direction and adapter type you are using), and then **Message Types**.
   b. Select **OutboundHeader_msg** and go to Step 8.

   Note: The name that displays here for the AQ adapter may only be **Header**. Select this name. This indicates that the header is static, because you did not need to select the **Access to non-payload fields** check box when configuring the AQ adapter.
8. Click **OK** to close the Type Chooser window, Create Variable window, and Variable Chooser window.
9. Complete the setup of the **Receive** activity.
10. Open the source view of the BPEL process file to view the header variable you created:

```
bpelx:headerVariable="Variable_Header"/>
```

Configuring a Message Rejection Handler for Data Errors

Rejected messages (that is, messages with data errors) can be directed to a rejected messages queue, and can also be logged in a directory on the system for later review.

An example of a rejection handler can be found at

Oracle_Home\integration\orabpel\samples\tutorials\124.AQAdapter\AQMessageRejectionHandler

The `readme.txt` file in this directory describes a procedure for testing the message rejection handler.

The `Dequeue.wsdl` file in this directory includes the following code:

```
<pc:inbound_binding />
    <operation name="Dequeue">
  <jca:operation
      ActivationSpec="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec"
      QueueName="REJECTION_TEST_IN"
      DatabaseSchema="SCOTT"
      OpaqueSchema="false" >
  </jca:operation>
  <input>
    <jca:header message="hdr:Header" part="Header"/>
  </input>
    </operation>
</binding>
```

This part of the code directs the errored message to the `REJECTION_TEST_IN` queue.

Summary

This chapter discusses Oracle Streams AQ concepts and how to configure various types of AQ adapters. It also discusses different ways of filtering messages received from multisubscriber queues, and copying header information into BPEL variables for decision making.

4

Oracle Application Server Adapter for Databases

This chapter describes the Oracle Application Server Adapter for Databases (database adapter), which works in conjunction with Oracle BPEL Process Manager. Support for stored procedures and functions (for Oracle databases only) is also described. References to use cases for the database adapter and for stored procedures are provided.

This chapter contains the following topics:

- Introduction to the Database Adapter
- Database Adapter Concepts
- Use Cases for the Database Adapter
- The Adapter Configuration Wizard
- Advanced Configuration
- Third-Party Database Support
- Stored Procedure and Function Support
- Use Case for Creating and Configuring a Stored Procedure in JDeveloper BPEL Designer
- Summary

Introduction to the Database Adapter

The database adapter enables a BPEL process to communicate with Oracle databases or third-party databases through JDBC. The database adapter service is defined within a BPEL process partner link using the Adapter Configuration Wizard of Oracle BPEL Process Manager.

This section contains the following topics:

- Database Adapter Features
- Design Overview

Database Adapter Features

The database adapter connects to any relational database. For nonrelational databases and legacy systems, application and mainframe adapters are available. See *Oracle Application Server Adapter Concepts* for information about application and mainframe adapters.

To access an existing relational schema, you use the Adapter Configuration Wizard to do the following:

- Import a relational schema and map it as an XML schema (XSD)

  See "Relational-to-XML Mapping" on page 4-4 for more information.

- Abstract SQL operations such as `SELECT`, `INSERT`, and `UPDATE` as Web services

  See "SQL Operations as Web Services" on page 4-8 for more information.

While your BPEL process deals with XML and invokes Web services, database rows and values are queried, inserted, and updated. Unlike other solutions that give you a way to access data using a fixed schema, stored procedures, streams, or queues, with the database adapter, you access table data directly and transparently.

Features of the database adapter include:

- Compliance with open standards. The database adapter is an implementation of the JCA 1.5 connector. Like the other adapters that work with Oracle BPEL Process Manager, the database adapter is compatible with open standards such as BPEL, WSIF, and WSDL.
- Connectivity to any relational (SQL 92) database using JDBC, or ODBC using the Sun JdbcOdbcBridge
- Ability to map *any existing* relational schema to XML. The mapping is nonintrusive to the schema and no changes need to be made to it.
- Web services abstraction of SQL operations. The generated WSDL operations are `merge`, `insert`, `update`, `write`, `delete`, `select`, `queryByExample`, and inbound polling, which includes physical delete, logical delete, and sequencing-based polling strategies.
- Leveraging of OracleAS TopLink technology, an advanced object-to-relational persistence framework. You can access the underlying TopLink project, and use the OracleAS TopLink Mapping Workbench interface for advanced mapping and configuration, sequencing, batch and joined relationship reading, batch writing, parameter binding, statement caching, connection pooling, external transaction control (JTS and JTA), UnitOfWork for minimal updates, caching, optimistic locking, advanced query support, and query by example.

See the following for more information:

- The Oracle BPEL Process Manager forum at

  `http://forums.oracle.com/forums/forum.jsp?forum=212`

- The OracleAS TopLink forum at

  `http://forums.oracle.com/forums/forum.jsp?forum=48`

  This site contains over 2,000 topics, such as implementing native sequencing, optimistic locking, and JTA-managed connection pools with OracleAS TopLink

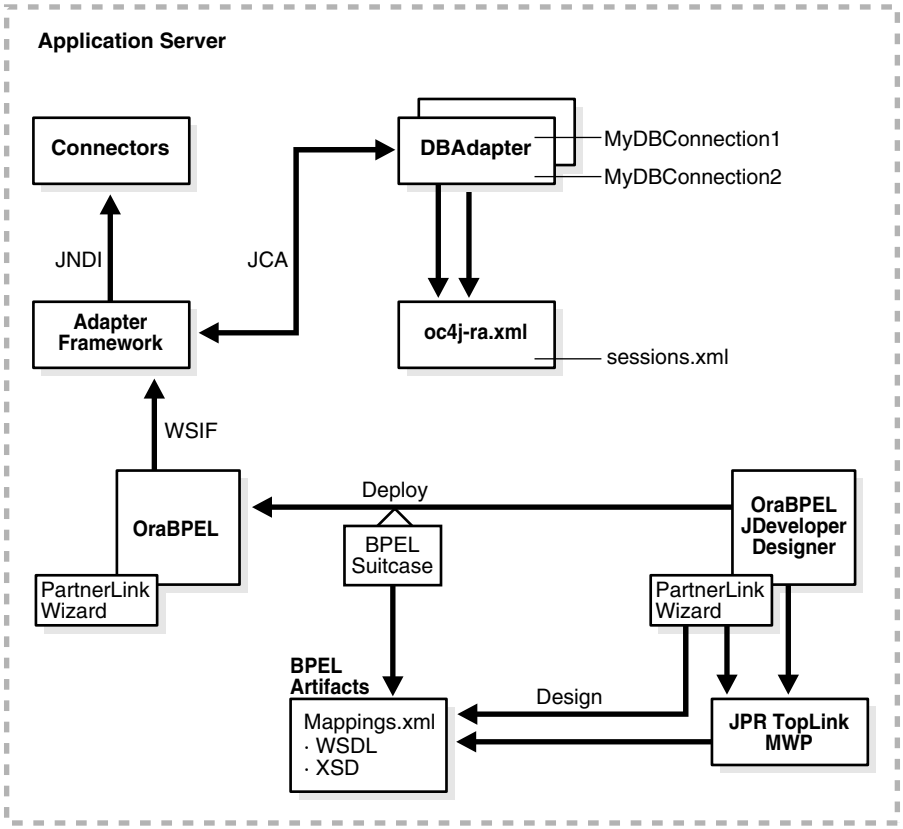
You can also access the forums from Oracle Technology Network at

`http://www.oracle.com/technology`

Design Overview

Figure 4–1 shows how the database adapter interacts with the various design-time and deployment artifacts.

Figure 4–1 How the Database Adapter Works



The database adapter is a separate JCA 1.5 connector. It is deployed to the application server during installation, and is configured using `oc4j-ra.xml`.

Each entry in `oc4j-ra.xml` has a Java Naming and Directory Interface (JNDI) name (location) and session and login properties, and represents a single database and database adapter instance. The connector is tied to the application server; therefore, it can be used independently, but any change to `oc4j-ra.xml` requires restarting the application server.

When a business process is executed in Oracle BPEL Process Manager, a Web service (WSDL) may be invoked (using WSIF) against the database. The `jca:address` tag in the WSDL is used to look up an adapter instance, and the `jca:operation` tag in the WSDL is used to set up an interaction (outbound) or activation (inbound) with the database adapter using a JCA interface. The `jca:operation` tag contains a link to OracleAS TopLink metadata for executing a query to push XML data to a relational schema, or vice versa.

The `toplink_mappings.xml` file and WSDL (with custom `jca:operation` and `jca:address` tags) are created during design time. In JDeveloper BPEL Designer, you create an endpoint, or partner link, for interacting with the database. Each partner link has its own WSDL. The WSDL defines all the operations (queries) that can be performed with that database.

To create the WSDL, you use the Adapter Configuration Wizard, where you import the relational schema, map it to an XML schema, and define one or more operations. This produces an XSD representing the schema and a `toplink_mappings.xml` file.

The Adapter Configuration Wizard creates an OracleAS TopLink Mapping Workbench project (a `.mwp` file) as part of the JDeveloper BPEL Designer project. Like the

JDeveloper BPEL Designer `.jpr` file, it allows a user to go back and visually change mappings or leverage OracleAS TopLink to set advanced properties. Saving the MWP project does not regenerate `toplink_mappings.xml`; that is done by running through the wizard again in edit mode. (No changes are needed; you simply run through it.)

During deployment, a copy of `toplink_mappings.xml` is included in the BPEL suitcase. It is later read by the database adapter and the metadata is cached.

The database adapter is used for relational-to-XML mapping; therefore, no Java class files are needed. The database adapter generates byte codes for the classes in memory based on information in the descriptors. You do not compile class files or deal with class path issues when using the database adapter. The MWP project in the JDeveloper BPEL Designer project may create Java files as a by-product of using the wizard, but they are needed at design time only.

Database Adapter Concepts

This section contains the following topics:

- Relational-to-XML Mapping
- SQL Operations as Web Services

Relational-to-XML Mapping

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 4–1 shows the structure of the `MOVIES` table. This table is used in the use cases described in this chapter. See "Use Cases for the Database Adapter" on page 4-16 for more information.

Table 4–1 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
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 as follows:

```
<xs:complexType name="Movies">
  <xs:sequence>
    <xs:element name="director" type="xs:string" minOccurs="0" nillable="true"/>
    <xs:element name="genre" type="xs:string" minOccurs="0" nillable="true"/>
    <xs:element name="rated" type="xs:string" minOccurs="0" nillable="true"/>
    <xs:element name="rating" type="xs:string" minOccurs="0" nillable="true"/>
    <xs:element name="releaseDate" type="xs:dateTime" minOccurs="0"
nillable="true"/>
    <xs:element name="runTime" type="xs:double" minOccurs="0" nillable="true"/>
    <xs:element name="starring" type="xs:string" minOccurs="0" nillable="true"/>
    <xs:element name="status" type="xs:string" minOccurs="0" nillable="true"/>
    <xs:element name="synopsis" type="xs:string" minOccurs="0" nillable="true"/>
    <xs:element name="title" type="xs:string"/>
    <xs:element name="totalGross" type="xs:double" minOccurs="0" nillable="true"/>
    <xs:element name="viewerRating" type="xs:string" minOccurs="0"
nillable="true"/>
  </xs:sequence>
</xs:complexType>
```

As the preceding code example shows, MOVIES is not just a single CLOB or XMLTYPE column containing the entire XML string. Instead, it is an XML complexType comprised of elements, each of which corresponds to a column in the MOVIES table. For flat tables, the relational-to-XML mapping is straightforward.

Table 4–2 and Table 4–3 show the structure of the EMP and DEPT tables, respectively. These tables are used in the MasterDetail use case. See "Use Cases for the Database Adapter" on page 4-16 for more information.

Table 4–2 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 4–3 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 equivalent has no similar notion of primary keys and foreign keys. Consequently, in the resulting XML, the same data is represented in a hierarchy, thereby preserving the relationships by capturing the detail record inline (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 allows the data to be sent as one packet of information. No cycles are allowed in the XML; therefore, an element cannot contain itself. This is handled automatically by the database adapter. However, you may 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 see the same `DEPT` record inline in both the `EMP` records.

Therefore, when you import tables and map them as XML, it is recommended that you avoid excessive duplication, although the database adapter does not print an element inside itself. The 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 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>
```

Note that one of the two preceding solutions is feasible only if getting back the foreign key suffices, as opposed to getting back the complete detail record in its entirety.

Relational Types to XML Schema Types

Table 4–4 shows how database datatypes are converted to XML primitive types when you import tables from a database.

Table 4–4 Mapping Database Datatypes 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
SHORT, SMALLINT	short
INT, SERIAL	int
INTEGER, BIGINT	integer
NUMBER, NUMERIC, DECIMAL, MONEY, SMALLMONEY, UNIQUEIDENTIFIER	decimal

Table 4–4 (Cont.) Mapping Database Datatypes to XML Primitive Types

Database Type	XML Type (Prefixed with :xs)
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 datatype 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. Timestamp support is basic, because only the xs:dateTime format is supported. The BFILE, USER DEFINED, OBJECT, STRUCT, VARRAY, and REF types are specifically not supported.

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 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 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 OracleAS TopLink Mapping Workbench. By pairing the database adapter with a transformation step, you can map any relational schema to any XML schema.

SQL Operations as Web Services

After mapping a relational schema as XML, you must also map basic SQL operations as Web services. Each operation discussed in the following sections has a corresponding tutorial and readme. It is recommended that you start with these and try to run one or more as you read this section. As the tutorials demonstrate, some operations translate directly to the SQL equivalent, while others are more complex.

See the following sections for details:

- "Use Cases for Outbound Invoke Operations" on page 4-9
- "Use Cases for Polling Strategies" on page 4-16

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 WRITE 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.

Merge Merge first reads the corresponding records in the database, calculates any changes, and then performs a minimal update. INSERT, UPDATE, and WRITE 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 particular row did not change or only one field changed, the DML calls will be 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 exemplar 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 needs to 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 varies.

```
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>
```

Use Cases for Outbound Invoke Operations Outbound invoke operations are demonstrated in the following tutorial files:

- Insert
- Update
- Delete
- Merge
- SelectAll
- SelectAllByTitle

- PureSQLSelect
- QueryByExample

For these files, go to

Oracle_Home\integration\orabpel\samples\tutorials\122.DBAdapter

Polling Strategies

The inbound receive allows 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 Oracle BPEL Process Manager. The stratagem 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 nonintrusive. Each strategy employs a different solution for the problem of what to do after reading a row or event so as not to pick it 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 strategies and factors to help you determine which strategy to employ for a particular situation:

- Physical Delete
- Logical Delete
- Sequencing Table: Last-Read Id
- Sequencing Table: Last Updated
- Control Tables

Physical Delete The physical delete polling strategy polls the database table for records and deletes them after processing. This strategy can be used 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 4–5 describes the requirements for using the delete polling strategy.

Table 4–5 Delete Polling Strategy Preconditions

Requirements Met	Conflicts with
Poll for inserts	No delete on source
Shallow delete[1]	No updates on source
Cascading delete[1]	Poll for updates
Minimal SQL	Poll for deletes
Zero data replication	Poll for child updates

Table 4–5 (Cont.) Delete Polling Strategy Preconditions

Requirements Met	Conflicts with
Default	--
Allows raw SQL	--
Concurrent polling[2]	--

[1] Delete can be configured to delete the top-level row, to cascade all, or to cascade on a case-by-case basis.
[2] 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 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`. 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 `toplink_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.

A receive operation appears in an inbound WSDL as:

```
<operation name="receive">
   <jca:operation
      ActivationSpec="oracle.tip.adapter.db.DBActivationSpec"
         …
      PollingStrategyName="DeletePollingStrategy"
      DeleteDetailRows="true"
```

Logical Delete The logical delete polling strategy involves updating a special field on each row processed, and updating the `WHERE` clause at run time 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 database adapter.

Preconditions: You must have the logical delete privilege or a one-time alter schema (add column) privilege on the source table. Table 4–6 describes the requirements for using the logical delete polling strategy.

Table 4–6 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	--

Table 4–6 (Cont.) Logical Delete Polling Strategy Preconditions

Requirements Met	Conflicts With
Minimal configuration	--
Allows raw SQL	--
Poll for updates[1]	--
Poll for child updates[2]	--
Concurrent polling[3]	--

[1] By adding a trigger.
[2] By adding a trigger.
[3] 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:

```
<operation name="receive">
   <jca:operation
      ActivationSpec="oracle.tip.adapter.db.DBActivationSpec"
         …
      PollingStrategyName="LogicalDeletePollingStrategy"
      MarkReadField="STATUS"
      MarkReadValue="PROCESSED"
```

Given the configuration for logical delete, the 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 should 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 needs to 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-1"
      MarkReadValue="PROCESSED"
```

The polling query instead looks like the following:

```
Update EMPLOYE set STATUS = 'RESERVED-1' where (CRITERIA) AND (STATUS =
'UNPROCESSED');

Select … from EMPLOYEE where (CRITERIA) AND (STATUS = 'RESERVED-1');
```

The after-read `UPDATE` is faster because it can update all:

```
Update EMPLOYEE set STATUS = 'PROCESSED' where (CRITERIA) AND (STATUS =
'RESERVED-1');
```

Sequencing Table: Last-Read Id 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 has already 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 fields on the processed row ever need to be modified by the database adapter.

Native sequencing with a preallocation 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 a sequencing strategy such as the `sequence` field can be used to order the rows in a sequence for processing. When the rows are ordered in a line, the 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 4–7 describes the requirements for using the sequencing polling strategy.

Table 4–7 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	Poll for child updates
One extra SQL select	--
Zero data replication	--
Moderate configuration	--

Configuration: A separate helper table must be defined. On the source table, you must specify which column is ever increasing.

```
<operation name="receive">
<jca:operation
ActivationSpec="oracle.tip.adapter.db.DBActivationSpec"
…
PollingStrategyName="SequencingPollingStrategy"
SequencingFieldName="MODIFIED_DATE"
```

```
SequencingFieldType="java.sql.Date"
SequencingTableNameFieldValue="EMPLOYEE"
SequencingTableName="SEQUENCING_HELPER"
SequencingTableNameFieldName="TABLE_NAME"
SequencingTableValueFieldName="LAST_READ_DATE"
```

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 above `ActivationSpec`, 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, as long as an increment by 1 is used. You can also use the sequence number of a materialized view log.

Sequencing Table: Last Updated 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 has already 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. No fields on the processed row ever need to be modified by the database adapter.

This strategy is also called a nondestructive delete because no updates are made to the source rows, and a sequencing strategy such as the `last_updated` field can be used to order the rows in a sequence for processing. When the rows are ordered in a line, the database adapter knows which rows are processed and which are not with a single unit of information.

See "Sequencing Table: Last-Read Id" on page 4-13 for information about preconditions and configuration.

Control Tables The control table 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. Information that is not in the rows themselves can be used 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 good control tables.

Preconditions: You must have create/alter triggers privilege on the source table. Table 4–8 describes the requirements for using the control table polling strategy.

Table 4–8 Control Table Polling Strategy Preconditions

Requirements Met	Conflicts With
Poll for inserts	Advanced configuration: the native XML from the database will have 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 timestamp, 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 OracleAS 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 need an explicit `WHERE` clause for each polling query.

Some sample triggers follow 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;
```

Use Cases for Polling Strategies Polling strategies are demonstrated in the following tutorials:

- `PollingLogicalDeleteStrategy`
- `PollingLastUpdatedStrategy`
- `PollingLastReadIdStrategy`
- `PollingControlTableStrategy`
- `MasterDetail` (for physical delete polling strategy

For these files, go to

Oracle_Home\integration\orabpel\samples\tutorials\122.DBAdapter

Use Cases for the Database Adapter

Using the database adapter is demonstrated in the `122.DBAdapter` tutorial. Go to

Oracle_Home\integration\orabpel\samples\tutorials\122.DBAdapter

Table 4–9 shows the database adapter samples that are provided with Oracle BPEL Process Manager.

Table 4–9 Database Adapter Use Cases

Tutorial Name	Description
`Delete`	Illustrates the outbound `delete` operation of the database adapter. An XML record is passed to the operation and the row in the database with the same primary key is deleted.
`File2StoredProcedure`	Describes a simple scenario in which the file adapter is used to provide instance XML to a stored procedure, `ADDEMPLOYEES`, which is then executed. The instance XML provides a value for the parameter of the stored procedure. The `ADDEMPLOYEES` procedure must be installed in an Oracle database (not Oracle Lite).
`File2Table`	Illustrates the use of an input a native (CSV) data file defined in a custom format. The input file is a purchase order, which the file adapter processes and publishes as an XML message to the FIle2Table BPEL process. The message is transformed to another purchase order format and routed to an `invoke` activity.
`Insert`	Illustrates the outbound `insert` operation of the database adapter. An XML record is passed to the operation and inserted into the database as relational data. (In JDeveloper BPEL Designer, `Merge` (`Insert` or `Update`) is provided.)
`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 in an Oracle database (not Oracle Lite).
`MasterDetail`	Illustrates how to migrate data from one set of tables to another. The sample uses the database adapter to read data from one set of tables, process the data, and write it in to another set of database tables using the adapter.
`Merge`	Illustrates the outbound `merge` operation of the database adapter. An XML record is passed to the operation and a corresponding row in the database is either inserted or updated.
`PollingControlTableStrategy`	Illustrates an inbound polling operation to poll XML instances from the `MOVIES table`. When a new row is inserted into the `MOVIES` table, the polling operation raises it to Oracle BPEL Process Manager. This strategy uses a control table to store the primary key of every row that has not yet been 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.
`PollingLastReadIdStrategy`	Illustrates an inbound polling operation to poll XML instances from the `MOVIES` table. Whenever a new row is inserted into the `MOVIES` table, the polling operation raises it to Oracle BPEL Process Manager. This strategy uses a helper table to remember a sequence value.
`PollingLastUpdatedStrategy`	Illustrates an inbound polling operation to poll XML instances from the `MOVIES` table. Whenever a new row is inserted into the `MOVIES` table, the polling operation raises it to Oracle BPEL Process Manager. This strategy involves using a helper table to remember a `last_updated` value.
`PollingLogicalDeleteStrategy`	Illustrates an inbound polling operation to poll XML instances from the `MOVIES` table. Whenever a new row is inserted into the `MOVIES` table, the polling operation raises it to Oracle BPEL Process Manager. This strategy involves updating a special field on each row processed, and updating the `WHERE` clause at run time to filter out processed rows.
`PureSQLPolling`	Illustrates how to poll a table based on a date field.
`PureSQLSelect`	Illustrates how to bypass the JDeveloper BPEL Designer `WHERE`-clause builder to specify arbitrarily complex SQL strings for `SELECT` operations.

Table 4–9 (Cont.) Database Adapter Use Cases

Tutorial Name	Description
`QueryByExample`	illustrates the outbound `queryByExample` operation of the database adapter. A `SELECT` SQL query is built dynamically based on fields set in an example XML record, and any matching records are returned.
`ResultSetConverter`	Illustrates a workaround for using REF CURSORs. 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.
`SelectAll`	Illustrates the outbound `SelectAll` operation of the database adapter. With no `WHERE` clause, all rows in the `MOVIES` table are returned as XML.
`SelectAllByTitle`	Illustrates the outbound `SelectAllByTitle` operation of the database adapter. The row in the `MOVIES` table with the selected title is returned as XML.
`Update`	illustrates the outbound `Update` operation of the database adapter. An XML record is passed to the operation and the row in the database with the same primary key is updated. (In JDeveloper BPEL Designer, `Merge` (`Insert` or `Update`) is provided.)

See Table 4–1 on page 4-4 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.

The Adapter Configuration Wizard

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 to both create and edit adapter services.

This section contains the following topics:

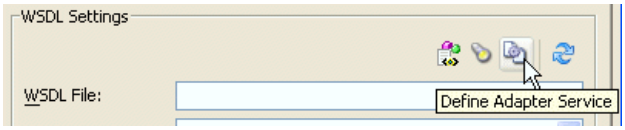
- Starting the Adapter Configuration Wizard
- Connecting to a Database
- Selecting the Operation Type
- Selecting and Importing Tables
- Defining Primary Keys
- Creating Relationships
- Creating the Object Model
- Defining a WHERE Clause
- Choosing an After-Read Strategy
- Internal Processes at Design Time

Starting the Adapter Configuration Wizard

After you create a BPEL project in JDeveloper BPEL Designer, you can start defining a database adapter. If you lose focus on the window, use `alt-tab` to get it back.

To launch the Adapter Configuration Wizard:

1. Ensure that **Process Activities** is selected in the drop-down list of the **Component Palette** section.
2. Drag and drop a **PartnerLink** activity onto the right side of the designer window.
3. Enter a name in the Create Partner Link window.
4. Click the **Define Adapter Service** icon to start the Adapter Configuration Wizard.



5. Click **Next** on the Welcome window.
6. Select **Database Adapter** for the **Adapter Service Type** and click **Next**.

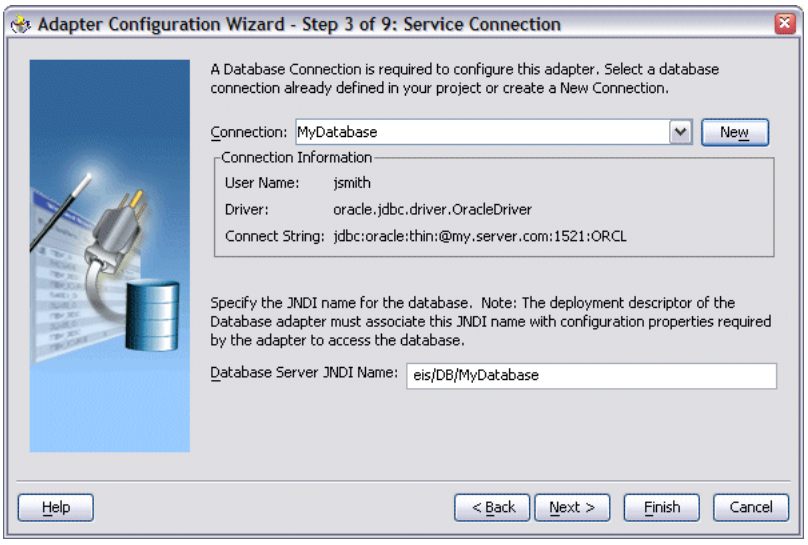
See "Connecting to a Database" on page 4-19 to continue using the wizard.

Connecting to a Database

Figure 4–2 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.

You can provide a Java Naming and Directory Interface (JNDI) name to identify the database connection, or use the default name that is provided. The JNDI name acts as a placeholder for the connection used when your service is deployed to Oracle BPEL Server. This enables you to use different databases for development and production. The Adapter Configuration Wizard captures the design-time connection in the generated WSDL as well, to serve as a fallback in case the run-time lookup fails.

Figure 4–2 Adapter Configuration Wizard: Service Connection



Note the following:

- In production environments, it is recommended that you add the JNDI entry to the adapter deployment descriptor (`oc4j-ra.xml`). This way, the database adapter is more performant by working in a managed mode. In a nonmanaged mode, the database adapter uses the design-time connection information.

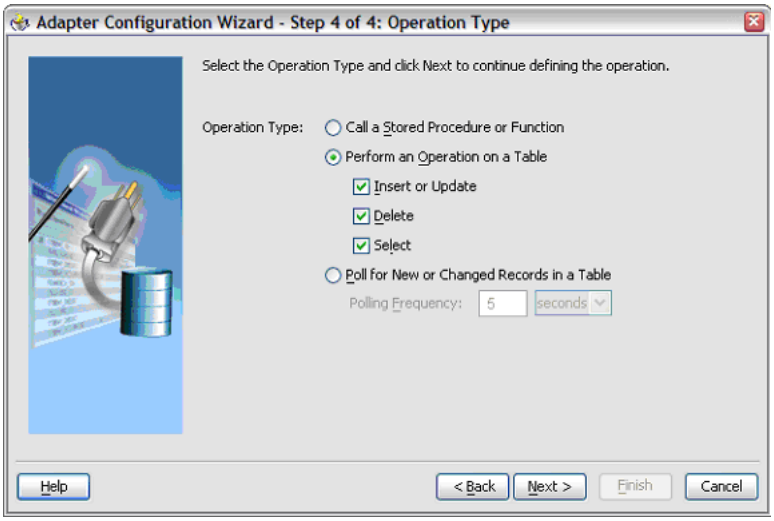
- When you click **Next**, a connection to the database is attempted. If a connection cannot be made, you are not able to proceed to the next window, even if you are editing an existing partner link.

See "Selecting the Operation Type" on page 4-20 to continue using the wizard.

Selecting the Operation Type

Figure 4–3 shows where you indicate the type of operation you want to configure for this service.

Figure 4–3 Adapter Configuration Wizard: Operation Type



The follow operation types are available:

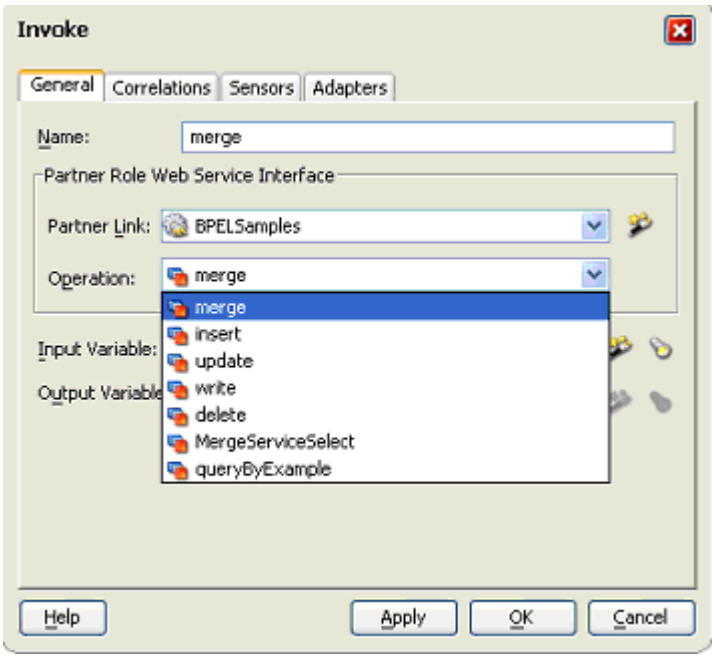
- **Call a Stored Procedure or Function**

  Select this option if you want the service to execute a stored procedure or function. See "Stored Procedure and Function Support" on page 4-54 for more information.

- **Perform an Operation on a Table**

  Select this option for outbound operations. You can select **Insert or Update**, **Delete**, **Select**, or any combination of the three. These operations loosely translate to SQL `INSERT`, `UPDATE`, `DELETE`, and `SELECT` operations. See "DML Operations" on page 4-8 for more information.

  If you select all three, then after you run the wizard, you see the following operations in the **Operation** list of the Invoke window: **merge, insert, update, write, delete,** ***serviceName*****Select,** and **queryByExample**.



Note the following:

- The operations **merge**, **insert**, **update**, and **write** are created from selecting **Insert or Update**.
- The preceding Invoke window shows the `MergeService` service name as part of the **Select** operation, that is, **MergeServiceSelect**.
- The **queryByExample** operation appears in every WSDL.
- If the **Operation** list is initially blank, reselect the partner link and click the **Operation** list again.

- **Poll for New or Changed Records in a Table**

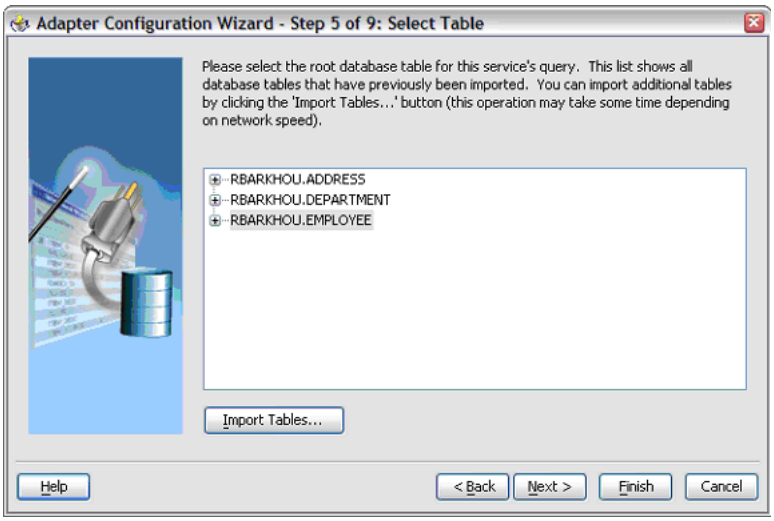
  Select this option for an inbound operation (that is, an operation that is associated with a **Receive** activity). This operation type polls a specified table and returns for processing any new rows that are added. You can also specify the polling frequency. See "Polling Strategies" on page 4-10 for more information.

See "Selecting and Importing Tables" on page 4-21 to continue using the wizard.

Selecting and Importing Tables

Figure 4–4 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 4–4 Adapter Configuration Wizard: Select Table



This window shows all the tables that have been previously imported in the JDeveloper BPEL Designer project (including tables that were imported for other partner links). This enables you to reuse configured table definitions across multiple partner links in a given BPEL project. These are the generated TopLink descriptors.

If the root database table you want to use for this operation has not been previously imported, you can click **Import Tables....** If you want to reimport a table (if the table structure has changed on the database, for example), import it again. You can then reimport the table and overwrite the previously configured table definition.

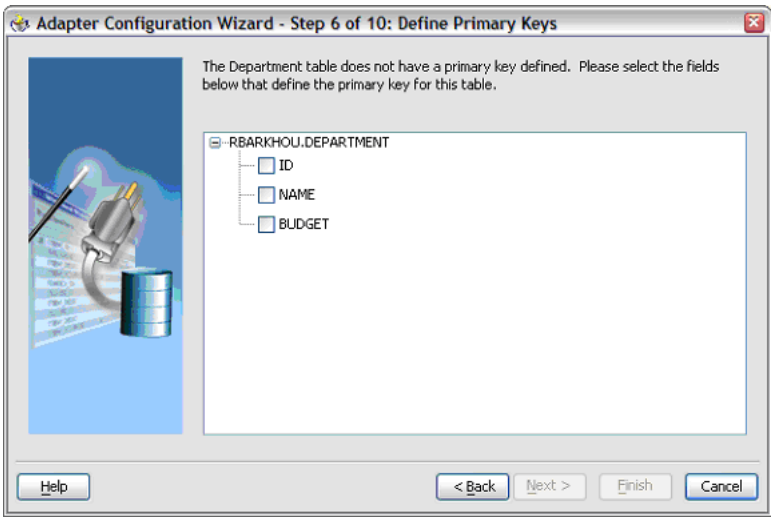
Note: If you reimport a table, you lose any custom relationships you may have defined on that table, as well as any custom `WHERE` clauses (if the table being imported was the root table).

See "Defining Primary Keys" on page 4-22 to continue using the 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 4–5. You must specify a primary key for all imported tables. You can select multiple fields if you need to specify a multipart primary key.

Figure 4–5 Adapter Configuration Wizard: Define Primary Keys



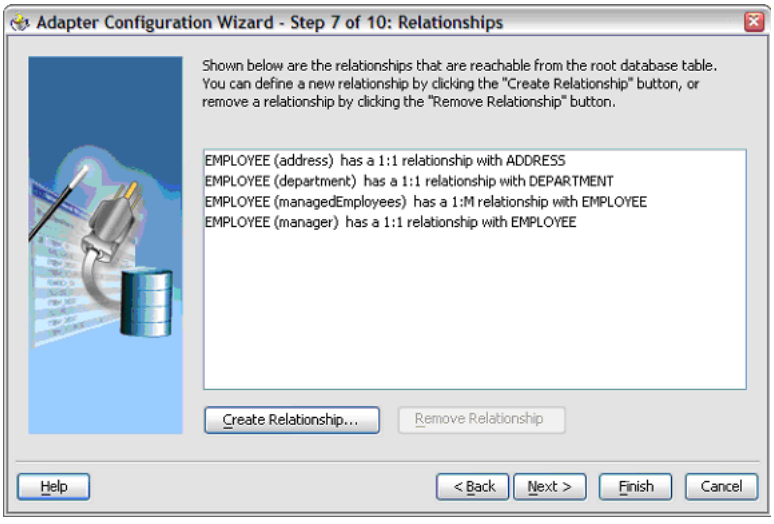
The primary key that you specify here is recorded on the offline database table and is not persisted back to the database schema; the database schema is left untouched.

See "Creating Relationships" on page 4-23 to continue using the wizard.

Creating Relationships

Figure 4–6 shows the relationships defined on the root database table and any other related tables. You can click **Create Relationships…** to create a new relationship between two tables, or **Remove Relationship** to remove it.

Figure 4–6 Adapter Configuration Wizard: Relationships



Note the following regarding creating relationships:

- If foreign key constraints between tables already 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, as well as a one-to-many (1:M) from the target table to the source table.

- As Figure 4–6 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
```

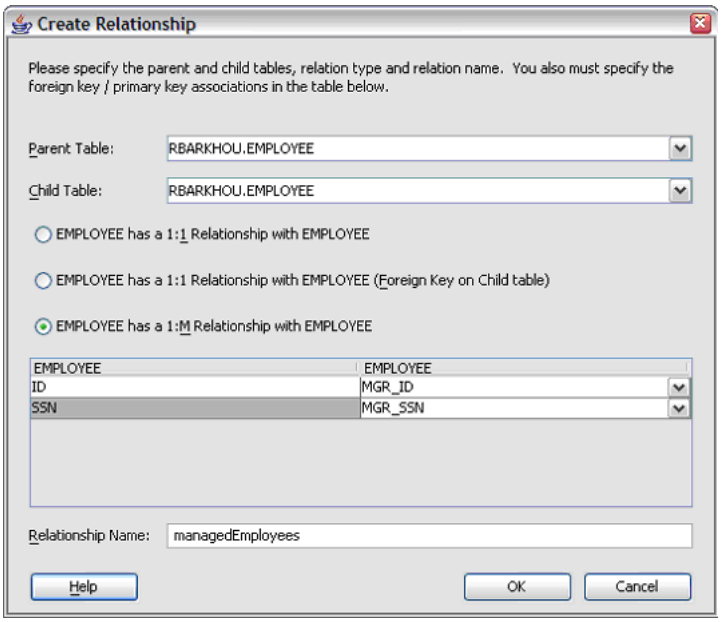
  If you remove relationship 2, then you see only:

```
A --1:1--> B
```

  If you remove relationship 1, you no longer see any relationships.

Figure 4–7 shows where you can create a new relationship.

Figure 4–7 Creating Relationships



To create a new 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 wizard, a TopLink direct-to-field mapping corresponding to each field in the database is created. Consider the schemas shown in Figure 4–8 and Figure 4–9:

Figure 4–8 EMPLOYEE Schema

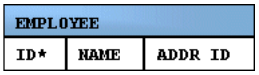


Figure 4–9 ADDRESS Schema

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 looks like this:

`Employee:`

- `id`
- `name`
- `homeAddress` (one-to-one mapping to the `ADDRESS` table; this attribute now represents the entire `Address` object.)

When a relationship is removed, the direct mappings for the foreign keys are restored.

Different Types of One-to-One Mappings

The following ways of specifying one-to-one relationships are supported:

- The foreign keys exist on the parent table, as shown in Figure 4–10 and Figure 4–11.
- The foreign keys exist on the child table, as shown in Figure 4–12 and Figure 4–13.

Figure 4–10 Foreign Keys on the Parent Table EMPLOYEE

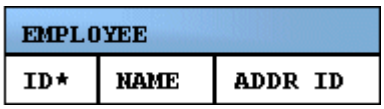


Figure 4–11 Foreign Keys on the Parent Table ADDRESS

ADDRESS		
ID*	ZIP	STREET

Figure 4–12 Foreign Keys on the Child Table EMPLOYEE

EMPLOYEE	
ID*	NAME

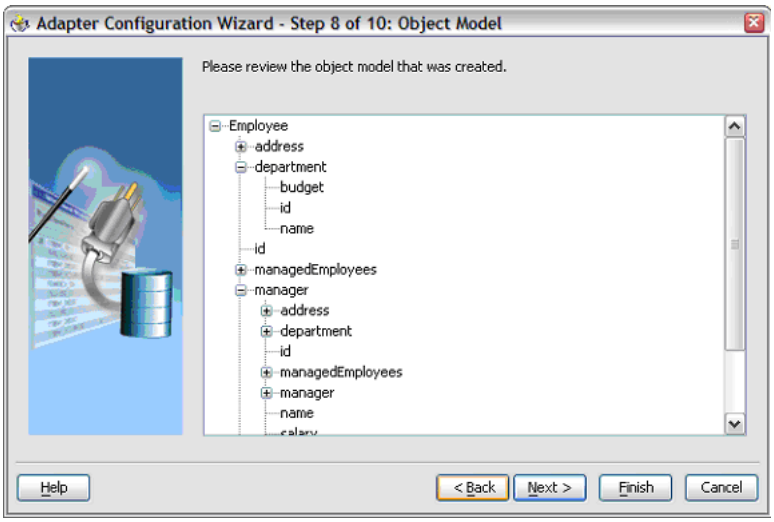
Figure 4–13 Foreign Keys on the Child Table ADDRESS

ADDRESS			
ID*	ZIP	STREET	EMP_ID

Creating the Object Model

Figure 4–14 shows the object model that is created from the imported table definitions, including any relationships that you may have defined.

Figure 4–14 Adapter Configuration Wizard: Object Model



If your object model 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. This is the descriptor object model, not the XSD.

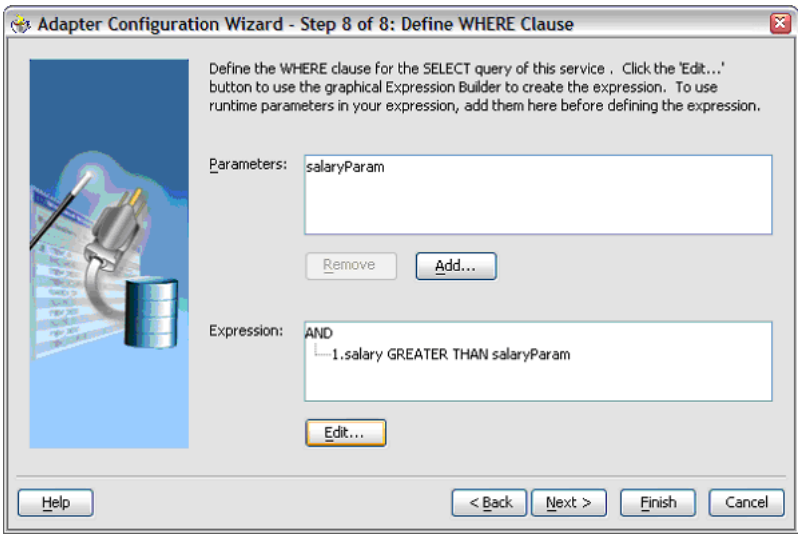
See "Defining a WHERE Clause" on page 4-26 to continue using the wizard.

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.

Figure 4–15 shows where you define a WHERE clause for an outbound service. For inbound services, you do not see the **Parameters** section.

Figure 4–15 Adapter Configuration Wizard: Define WHERE Clause



> **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 4–16.

2. `EMP.ADDR_ID = ADDR.ID`

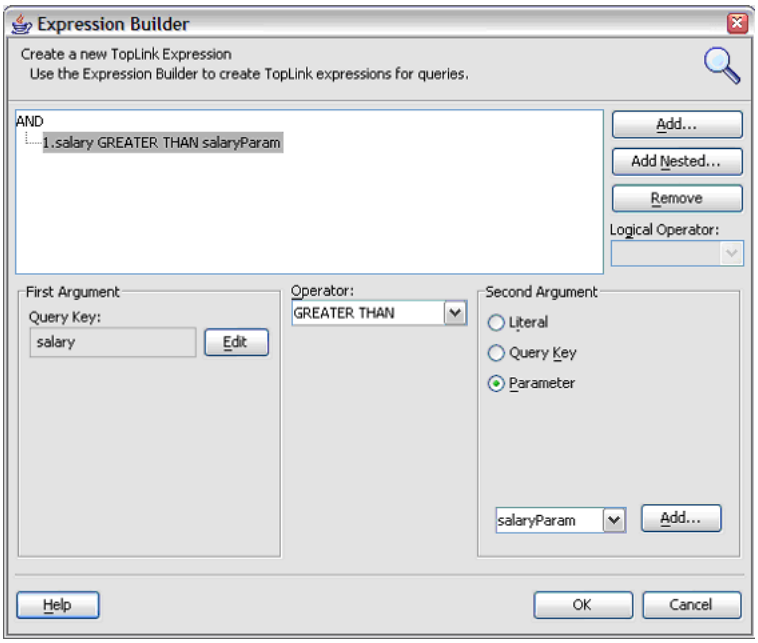
   In this case, the RHS is another database field. This RHS is the **Query Key** option shown in Figure 4–16.

3. `EMP.ID = ?`

   In this case, the RHS value must be specified at run time. This is the **Parameter** option shown in Figure 4–16.

You create the parameters that you need 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 4–16.

Figure 4–16 Expression Builder



See the following for more information:

- The OracleAS TopLink page on OTN at

  `http://www.oracle.com/technology/products/ias/toplink/index.html`

- OracleAS TopLink documentation at

  `http://download.oracle.com/docs/cd/B14099_04/web.htm#toplink`

  This site contains documentation on configuring expressions using the XPath Expression Builder.

See "Choosing an After-Read Strategy" on page 4-28 to continue using the wizard.

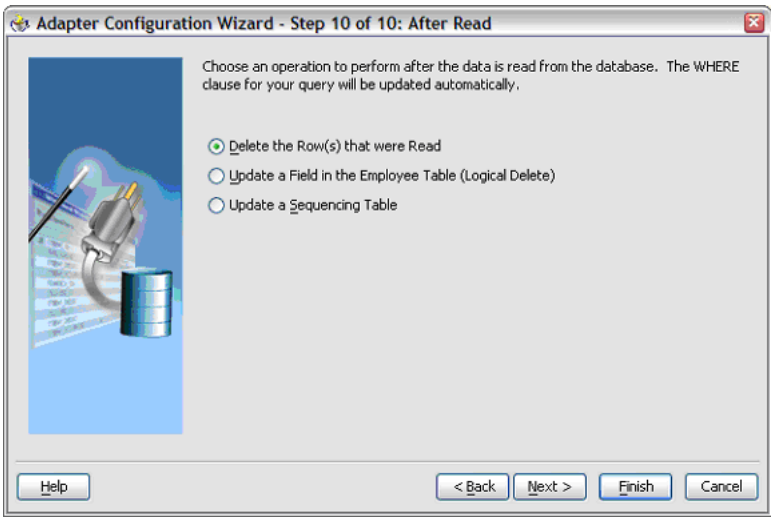
Choosing an After-Read Strategy

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

Figure 4–17 shows these options.

Figure 4–17 Adapter Configuration Wizard: After-Read Strategies



See "Polling Strategies" on page 4-10 for more information.

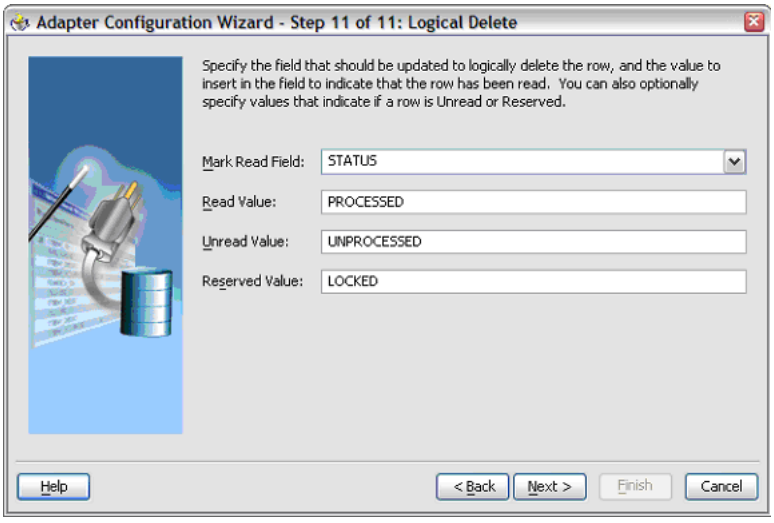
Delete the Rows that Were Read

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 4–18.

Figure 4–18 Adapter Configuration Wizard: Logical Delete



Using this approach, your database table looks something like Figure 4–19.

Figure 4–19 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

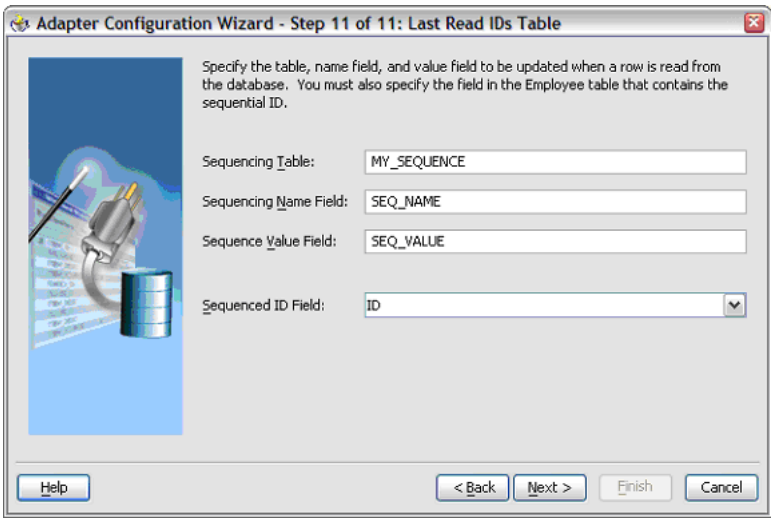
Note the following:

- Rows 150 and 153 have been previously read and processed.
- At the next polling event, row 152 will be read and processed because it contains `UNPROCESSED` in the Status column. Because an explicit `Unread Value` was provided, row 151 will not be read.
- Row 154 has been flagged as `LOCKED` and will not be read. You can use this reserved value if your table is in use 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 4–20 shows the information you provide. The `WHERE` clause of your query is updated automatically after you complete the configuration.

Figure 4–20 Adapter Configuration Wizard: Last Read IDs Table



Using these settings, your sequence table looks something like Figure 4–21.

Figure 4–21 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 will search 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`.

Internal Processes at Design Time

This section describes happens internally at design time when you use the Adapter Configuration Wizard to configure the database adapter.

Importing Tables

When you import a table, the offline table support of JDeveloper BPEL Designer creates an offline snapshot of the database table. You can modify this offline version of the table (for example, you can add a foreign key constraint) without affecting the real database table. This creates a TopLink descriptor and associated Java source file for the table, and all the attributes in the descriptor are automapped to their corresponding database columns. The TopLink descriptor maps the Java class to the offline database table.

Most typical data columns are mapped as direct-to-field mappings, meaning that the value in the database column is directly mapped to the attribute. For example, a `SALARY` column in the database is mapped to a `salary` attribute in the object model, and that attribute contains the value of that column.

If foreign key constraints are already present in the imported tables, then relationship mappings are autogenerated between the tables. To cover as many scenarios as possible, two mappings are generated for every foreign key constraint encountered: a one-to-one mapping from the source table to the target table, and a one-to-many mapping in the opposite direction. After this is done, you are left with an OracleAS TopLink Mapping Workbench project in your BPEL project.

Note: The Java classes that are created as part of the descriptor generation process are never actually deployed with your process or used at run time. They are present in the design time because OracleAS TopLink Mapping Workbench is expecting each descriptor to be associated with a Java class. When your process is deployed, the mapping metadata is stored in `toplink_mappings.xml`.

When you have finished importing tables, you must select a root database table. In doing so, you are actually selecting which TopLink descriptor stores the autogenerated query.

Creating Relationships

When you create or remove a relationship, you are actually modifying the TopLink descriptor mappings. Creating a new relationship does the following:

- Creates a foreign key constraint in the offline database table
- Adds a one-to-one or one-to-many mapping to the descriptor
- Removes the direct-to-field mappings to the foreign key fields

Removing a relationship mapping does the following:

- Removes the one-to-one or one-to-many mapping from the descriptor
- Removes the foreign key constraint from the offline database table
- Adds direct-to-field mappings for each foreign key field involved in the relationship

Generating Design-Time Artifacts

The following files are generated:

- *service_name*`.wsdl`—contains the database adapter service definition
- *RootTable*`.xsd`—the XML type definition of the root object
- `toplink_mappings.xml`—contains the TopLink mapping metadata for your BPEL project. It is the only Toplink artifact that is deployed to the server.

Advanced Configuration

The Adapter Configuration Wizard generates everything you need to use the database adapter as part of a BPEL process. The following sections describe what happens in the background when you use the wizard, as well as performance considerations.

This section contains the following topics:

- The OracleAS TopLink Mapping Workbench Project
- Relational-to-XML Mappings (toplink_mappings.xml)
- The Service Definition (WSDL)
- XML Schema Definition (XSD)
- Deployment
- Performance

The OracleAS TopLink Mapping Workbench Project

The wizard works by creating an OracleAS TopLink Mapping Workbench project as part of your BPEL process project. This TopLink project contains metadata for mapping a database schema to objects/XML.

The TopLink mappings are stored in two formats. The `toplink_mappings.mwp` file is your design time project, which you can edit visually in JDeveloper BPEL Designer. In contrast, the `toplink_mappings.xml` file is an XML representation of your project for use at run time. It is not as easy as editing the `.bpel` file, where there is only one file, but you can toggle between **Diagram View** and **Source**.

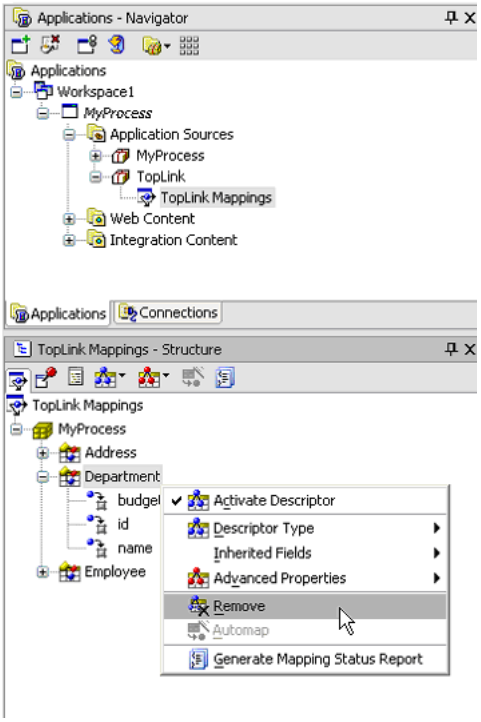
Note the following:

- Rather than edit the `toplink_mappings.xml` file directly, it is recommended that you edit the `toplink_mappings.mwp` visually, and regenerate all the BPEL artifacts to reflect the changes. You can do this by double-clicking the partner link to open the Adapter Configuration Wizard in edit mode, and then clicking through the wizard until you can click **Finish**. Changing the MWP version does not update the XML version until you click through the wizard in edit mode..
- When running the wizard, any changes that affect the TopLink project (importing tables, creating or removing mappings, specifying an expression, and so on) are applied immediately, and are *not undone* if you cancel the wizard.

Deleting a Descriptor

You cannot remove TopLink descriptors from your project from within the wizard because removing descriptors can potentially affect other partner links that are sharing that descriptor. To explicitly remove a descriptor, do the following:

- Click the **TopLink Mappings** node under **Application Sources** under your project in the **Applications - Navigator**.
- Select the descriptor from the tree in the **TopLink Mappings - Structure** pane.
- Right-click and select **Remove**.

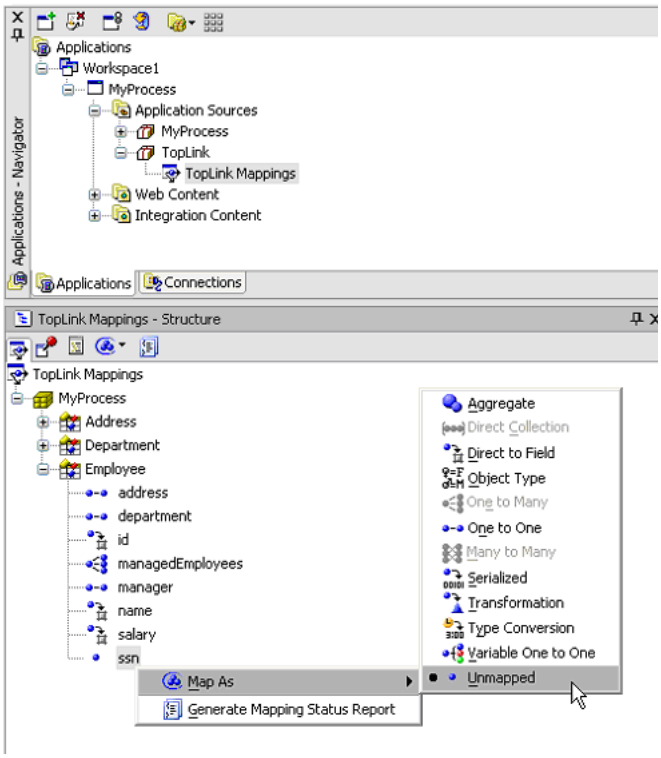


Returning Partial Objects When Querying

Currently, the Adapter Configuration Wizard does not have built-in support for partial object reading, that is, returning only specific fields from a table. To achieve this functionality, you can manually unmap any attributes that you do not want to include in the result set. Relationship mappings can be unmapped by removing them in the **Relationships** window, but direct mappings must be explicitly unmapped on the TopLink descriptor:

1. Click the **TopLink Mappings** node under **Application Sources** under your project in the **Applications - Navigator**.

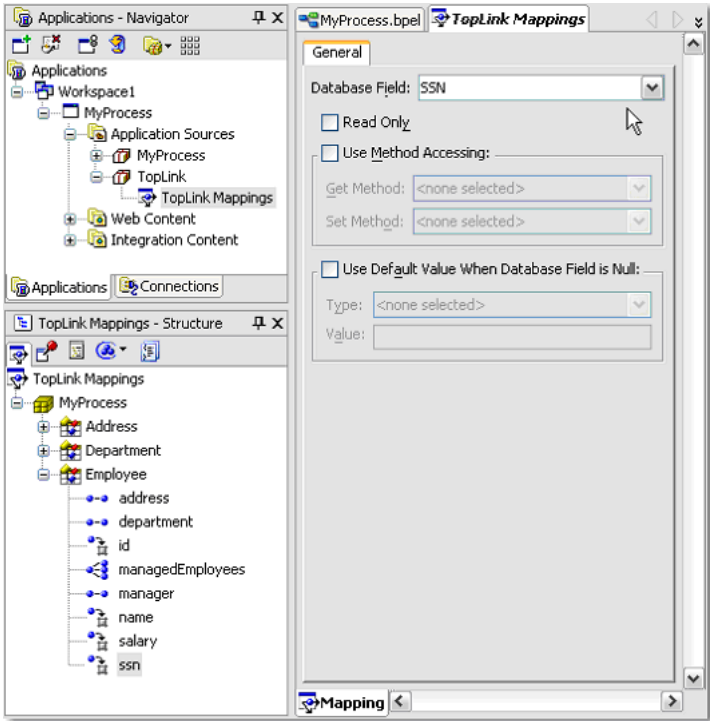
2. Select the descriptor containing the attribute you want to unmap from the tree in the **TopLink Mappings - Structure** pane.
3. Right-click the attribute you want to unmap and select **Map As > Unmapped**.



To remap the attribute, you can do the following:

1. Click the **TopLink Mappings** node under **Application Sources** under your project in the **Applications - Navigator**.
2. Select the descriptor containing the attribute you want to remap from the tree in the **TopLink Mappings - Structure** pane.
3. Right-click the attribute you want to remap and select **Map As > Direct to Field**.

   The TopLink Mappings Editor automatically opens in the JDeveloper BPEL Designer window.
4. From **Database Field**, select the column to which the attribute should be mapped.



Renaming a Mapping

Open the corresponding Java source file and change the name. Then go to the structure/Mappings pane, and the newly named attribute will appear unmapped. Right-click it and select **Map As** to remap it. Then save and regenerate BPEL artifacts.

Keep in mind there are four views, the project view, the table/descriptor view, and the individual attribute/column view you can access from the TopLink Mappings structure window. The Java source view is not exactly a TopLink view, but can be treated as such (when renaming a mapping).

Configuring Offline Database Tables

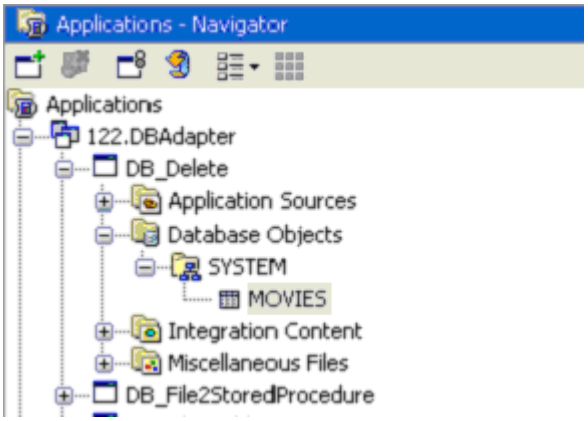
Offline database tables are internal to the OracleAS TopLink Mapping Workbench project. When you run the wizard, a TopLink project is created. When you import tables, they are saved as offline table definitions.

You can use the offline database tables to control the micromapping from database datatype to XML datatype. If you are using a third-party database, you may need to edit these objects as a workaround. For instance, a `serial` field on a third-party database may need to be mapped as `Integer` so that it is recognized by the wizard and mapped to `xs:integer`.

Run the wizard once. Then add the following to your JDeveloper BPEL Designer project: `database/`*`schemaName`*`/`*`schemaName`*`.schema`

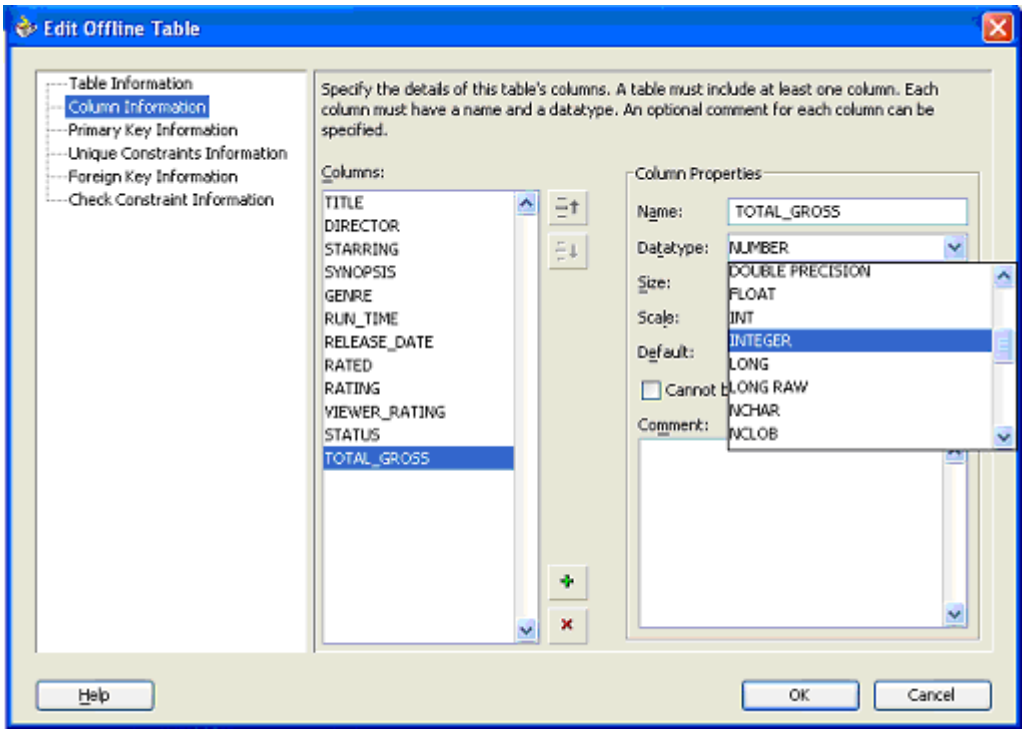
Click the table name (see Figure 4–22) after it is added to your project and change the types of any of the columns. When you run the wizard again (in edit mode) and click **Finish**, the `toplink_mappings.xml` and XSD file are remapped based on the new database datatypes.

Figure 4–22 Configuring Offline Tables



Edit the offline table in your JDeveloper BPEL Designer project (see Figure 4–23), not the table reachable from the `ConnectionManager`. If you try the latter, the column types will not be editable, because you are editing the table itself, not an offline representation of it.

Figure 4–23 Editing Offline Tables



Relational-to-XML Mappings (toplink_mappings.xml)

The database adapter is implemented using OracleAS TopLink. For every business process, there is an underlying TopLink project, which contains metadata from mapping a database schema to objects/XML.

In OracleAS TopLink terminology, `toplink_mappings.xml` is an XML deployment file. It is generated from a `.mwp` project file for use at run time. It is recommended that

you edit the project in OracleAS TopLink Mapping Workbench and periodically refresh `toplink_mappings.xml`.

The `toplink_mappings.xml` file is the run-time version of the OracleAS TopLink Mapping Workbench project. If you edit this file directly, keep in mind that changes are not reflected in the design-time `toplink_mappings.mwp`. Therefore, any changes are lost when you edit a partner link.

The `toplink_mappings.xml` file consists of a set of descriptors and mappings. Descriptors roughly represent a single table in the database schema, and mappings represent either a single column in the table (direct to field), or a one-to-one or one-to-many relationship to another table (foreign reference).

When modifying the `toplink_mappings.xml` file, the recommended approach is to use OracleAS TopLink Mapping Workbench. The following is an example of a mapping and a descriptor from a `toplink_mappings.xml` file.

```
<mappings>
   <database-mapping>
      <attribute-name>fname</attribute-name>
      <read-only>false</read-only>
      <field-name>ACTOR.FNAME</field-name>
      <attribute-classification>java.lang.String</attribute-classification>
      <type>oracle.toplink.mappings.DirectToFieldMapping</type>
   </database-mapping>
```

and:

```
<descriptor>
   <java-class>BusinessProcess.Actor</java-class>
   <tables>
      <table>ACTOR</table>
   </tables>
   <primary-key-fields>
      <field>ACTOR.ID</field>
      <field>ACTOR.PROGRAM_ID</field>
      <field>ACTOR.PROGRAM_TYPE</field>
   </primary-key-fields>
```

However, the recommended approach is to work from the OracleAS TopLink Mapping Workbench.

Useful attributes on foreign reference mappings (one-to-one, one-to-many) include:

- `<privately-owned>false/true`

  If a relationship is privately owned, that means that any target rows are deleted whenever any source rows are deleted.

  This is important for one-to-many relationships because, if you remove Dept without first deleting its Emp rows, you get a 'child records found' constraint exception.

  If you set `privately-owned` to true, the database adapter automatically deletes child records before deleting source rows. In XML everything is assumed to be privately owned; therefore, this tag is set to true by default.

- `<uses-batch-reading>false/true` and `<uses-joining>false/true`

  There are two key optimizations in relation to reading rows with detail rows from the database.

The following shows the series of selects that OracleAS TopLink uses to read two department objects (1 and 2), and their employees:

Unoptimized:

```
SELECT DEPT_COLUMNS FROM DEPT WHERE (subQuery)
SELECT EMP_COLUMNS FROM EMP WHERE (DEPTID = 1)
SELECT EMP_COLUMNS FROM EMP WHERE (DEPTID = 2)
```

Batch Reading:

```
SELECT DEPT_COLUMNS FROM DEPT WHERE (subQuery)
SELECT EMP_COLUMNS FROM EMP e, DEPT d WHERE ((subQuery) AND (e.DEPTID =
d.DEPTID))
```

Joined Reading:

```
SELECT DEPT_COLUMNS, EMP_COLUMNS FROM DEPT d, EMP e WHERE ((subQuery) AND
(e.DEPTID = d.DEPTID))
```

Joined reading appears to be the more advanced, but only works for one-to-one mappings currently, and the detail record cannot be null because the join is not an outer join.

Therefore, by default, batch reading is enabled, but not joined reading. This can easily be reversed to improve performance.

If you specify raw SQL for a query, that query cannot be a batched or joined read. To use batched or joined reading, you must not use raw SQL.

You can set other properties in `toplink_mappings.xml`.

The Service Definition (WSDL)

The WSDL generated by the Adapter Configuration Wizard defines the adapter service. This WSDL specifies the various operations exposed by the service. Table 4–10 specifies the operations that are generated based on your selection in the wizard.

Table 4–10 WSDL Operations Generated by the Adapter Configuration Wizard

Adapter Configuration Wizard Selection	Generated WSDL Operation
Insert or Update	`insert, update, merge, write, queryByExample`
Delete	`delete, queryByExample`
Select	*`serviceName`*`Select, queryByExample`
Poll for New or Changed Records in a Table	`receive`

Of the preceding operations, `receive` is associated with a BPEL `receive` activity, whereas the rest of the preceding operations are associated with a BPEL `invoke` activity.

See "SQL Operations as Web Services" on page 4-8 for more information on the preceding operations.

This section discusses the database adapter-specific parameters in the generated WSDL. This is intended for advanced users who want information about all the parameters in the generated WSDL.

A given database adapter service is meant for either continuous polling of a data source (translates to a JCA Activation) or for performing a one-time DML operation (translates to a JCA Interaction). In the continuous polling case, the WSDL contains only one receive operation with a corresponding activation spec defined in the binding section. In the one-time DML operation case, the WSDL contains multiple operations, all of which have a corresponding interaction spec defined in the binding section.

Table 4–11 specifies the JCA Activation/Interaction spec associated with each of the preceding operations:

Table 4–11 Operation and JCA Activation/Interaction Spec

WSDL Operation	JCA Activation/Interaction Spec
`insert, update, merge, write, delete`	`oracle.tip.adapter.db.DBWriteInteractionSpec`
`select, queryByExample`	`oracle.tip.adapter.db.DBReadInteractionSpec`
`receive`	`oracle.tip.adapter.db.DBActivationSpec`

DBWriteInteractionSpec

The following code example shows the binding section corresponding to the `movie` service to write to the `Movies` table:

```
<binding name="movie_binding" type="tns:movie_ptt">
    <jca:binding />
    <operation name="merge">
        <jca:operation
            InteractionSpec="oracle.tip.adapter.db.DBWriteInteractionSpec"
            DescriptorName="BPELProcess1.Movies"
            DmlType="merge"
            MappingsMetaDataURL="toplink_mappings.xml" />
        <input/>
    </operation>
    <operation name="insert">
        <jca:operation
            InteractionSpec="oracle.tip.adapter.db.DBWriteInteractionSpec"
            DescriptorName="BPELProcess1.Movies"
            DmlType="insert"
            MappingsMetaDataURL="toplink_mappings.xml" />
        <input/>
    </operation>
    <operation name="update">
        <jca:operation
            InteractionSpec="oracle.tip.adapter.db.DBWriteInteractionSpec"
            DescriptorName="BPELProcess1.Movies"
            DmlType="update"
            MappingsMetaDataURL="toplink_mappings.xml" />
        <input/>
    </operation>
    <operation name="write">
        <jca:operation
            InteractionSpec="oracle.tip.adapter.db.DBWriteInteractionSpec"
            DescriptorName="BPELProcess1.Movies"
            DmlType="write"
            MappingsMetaDataURL="toplink_mappings.xml" />
        <input/>
    </operation>
    <operation name="delete">
        <jca:operation
```

```
                InteractionSpec="oracle.tip.adapter.db.DBWriteInteractionSpec"
                DescriptorName="BPELProcess1.Movies"
                DmlType="delete"
                MappingsMetaDataURL="toplink_mappings.xml" />
            <input/>
    </binding>
```

Table 4–12 describes the `DBWriteInteractionSpec` parameters:

Table 4–12 DBWriteInteractionSpec Parameters

Parameter	Description	Mechanism to Update
`DescriptorName`	Indirect reference to the root database table that is being written to	Wizard updates automatically. Do *not* modify this manually.
`DmlType`	The DML type of the operation (`insert`, `update`, `merge`, `write`)	Wizard updates automatically. Do *not* modify this manually.
`MappingsMetaDataURL`	Reference to file containing relational-to-XML mappings (`toplink_mappings.xml`)	Wizard updates automatically. Do *not* modify this manually.

DBReadInteractionSpec

The following code example corresponds to the `movie` service to query the `Movies` table:

```
<binding name="movie_binding" type="tns:movie_ptt">
       <jca:binding />
       <operation name="movieSelect">
           <jca:operation
               InteractionSpec="oracle.tip.adapter.db.DBReadInteractionSpec"
               DescriptorName="BPELProcess1.Movies"
               QueryName="movieSelect"
               MappingsMetaDataURL="toplink_mappings.xml" />
           <input/>
       </operation>
       <operation name="queryByExample">
           <jca:operation
               InteractionSpec="oracle.tip.adapter.db.DBReadInteractionSpec"
               DescriptorName="BPELProcess1.Movies"
               IsQueryByExample="true"
               MappingsMetaDataURL="toplink_mappings.xml" />
           <input/>
       </operation>
   </binding>
```

Table 4–13 describes the `DBReadInteractionSpec` parameters:

Table 4–13 DBReadInteractionSpec Parameters

Parameter	Description	Mechanism to Update
`DescriptorName`	Indirect reference to the root database table that is being queried	Wizard updates automatically. Do *not* modify this manually.

Table 4–13 (Cont.) DBReadInteractionSpec Parameters

Parameter	Description	Mechanism to Update
QueryName	Reference to the SELECT query inside the relational-to-XML mappings file	Wizard updates automatically. Do *not* modify this manually.
IsQueryByExample	Indicates if this query is a queryByExample or not	Wizard updates automatically. Do *not* modify this manually. This parameter is needed for queryByExample only.
MappingsMetaDataURL	Reference to file containing relational-to-XML mappings (toplink_mappings.xml)	Wizard updates automatically. Do *not* modify this manually.

DBActivationSpec

The following code example shows the binding section corresponding to the MovieFetch service to poll the Movies table using DeletePollingStrategy:

```
<binding name="MovieFetch_binding" type="tns:MovieFetch_ptt">
    <pc:inbound_binding/>
    <operation name="receive">
        <jca:operation
            ActivationSpec="oracle.tip.adapter.db.DBActivationSpec"
            DescriptorName="BPELProcess1.Movies"
            QueryName="MovieFetch"
            PollingStrategyName="DeletePollingStrategy"
            MaxRaiseSize="1"
            MaxTransactionSize="unlimited"
            PollingInterval="5"
            MappingsMetaDataURL="toplink_mappings.xml" />
    <input/>
    </operation>
</binding>
```

Table 4–14 describes the DBActivationSpec parameters:

Table 4–14 DBActivationSpec Parameters

Parameter	Description	Mechanism to Update
DescriptorName	Indirect reference to the root database table that is being queried	Wizard updates automatically. Do *not* modify this manually.
QueryName	Reference to the SELECT query inside the relational-to-XML mappings file	Wizard updates automatically. Do *not* modify this manually.
PollingStrategyName	Indicates the polling strategy to be used	Wizard updates automatically. Do *not* modify this manually.
PollingInterval	Indicates how often to poll the root database table for new events (in seconds)	Wizard updates automatically. Do *not* modify this manually.
MaxRaiseSize	Indicates the maximum number of XML records that can be raised at a time to the BPEL engine	Modify manually in the generated WSDL.
MaxTransactionSize	Indicates the maximum number of rows to process as part of one database transaction	Modify manually in the generated WSDL.
MappingsMetaDataURL	Reference to file containing relational-to-XML mappings (toplink_mappings.xml)	Wizard updates automatically. Do *not* modify this manually.

The following code example is the binding section corresponding to the MovieFetch service to poll the Movies table using LogicalDeletePollingStrategy:

```
<binding name="PollingLogicalDeleteService_binding"
         type="tns:PollingLogicalDeleteService_ptt">
    <pc:inbound_binding/>
    <operation name="receive">
        <jca:operation
            ActivationSpec="oracle.tip.adapter.db.DBActivationSpec"
            DescriptorName="PollingLogicalDeleteStrategy.Movies"
            QueryName="PollingLogicalDeleteService"
            PollingStrategyName="LogicalDeletePollingStrategy"
            MarkReadFieldName="DELETED"
            MarkReadValue="TRUE"
            MarkReservedValue="MINE"
            MarkUnreadValue="FALSE"
            MaxRaiseSize="1"
            MaxTransactionSize="unlimited"
            PollingInterval="10"
            MappingsMetaDataURL="toplink_mappings.xml" />
    <input/>
    </operation>
</binding>
```

Table 4–15 describes all of the additional `DBActivationSpec` parameters for `LogicalDeletePollingStrategy`:

Table 4–15 DBActivationSpec Parameters for LogicalDeletePollingStrategy

Parameter	Description	Mechanism to Update
`MarkReadFieldName`	Specifies the database column to use to mark the row as read	Wizard updates automatically. Do not modify this manually.
`MarkReadValue`	Specifies the value to which the database column is set to mark the row as read	Wizard updates automatically. Do *not* modify this manually.
`MarkReservedValue`	Specifies the value to which the database column is set to mark the row as reserved. This parameter is optional. You can use it when multiple adapter instances are providing the same database adapter service.	Wizard updates automatically. Do *not* modify this manually.
`MarkUnreadValue`	Specifies the value to which the database column is set to mark the row as unread. This parameter is optional. Use it when you want to indicate specific rows that the database adapter must process.	Wizard updates automatically. Do *not* modify this manually.

The following code example shows the binding section corresponding to the `MovieFetch` service to poll the `Movies` table using `SequencingPollingStrategy`:

```
<binding name="PollingLastReadIdStrategyService_binding"
         type="tns:PollingLastReadIdStrategyService_ptt">
    <pc:inbound_binding/>
    <operation name="receive">
        <jca:operation
            ActivationSpec="oracle.tip.adapter.db.DBActivationSpec"
            DescriptorName="PollingLastReadIdStrategy.Movies"
            QueryName="PollingLastReadIdStrategyService"
            PollingStrategyName="SequencingPollingStrategy"
            SequencingFieldName="SEQUENCENO"
            SequencingTableNameFieldValue="MOVIES"
            SequencingTableName="PC_SEQUENCING"
```

```
                SequencingTableNameFieldName="TABLE_NAME"
                SequencingTableValueFieldName="LAST_READ_ID"
                MaxRaiseSize="1"
                MaxTransactionSize="unlimited"
                PollingInterval="10"
                MappingsMetaDataURL="toplink_mappings.xml" />
        <input/>
        </operation>
    </binding>
```

Table 4–16 describes all of the additional `DBActivationSpec` parameters for `SequencingPollingStrategy`:

Table 4–16 DBActivationSpec Parameters for SequencingPollingStrategy

Parameter	Description	Mechanism to update
`SequencingFieldName`	Specifies the database column that is monotonically increasing	Wizard updates automatically. Do *not* modify this manually.
`SequencingFieldType`	Specifies the type of the database column that is monotonically increasing. This parameter is optional. Use it if the type is not `NUMBER`.	Wizard updates automatically. Do *not* modify this manually.
`SequencingTableNameFieldValue`	Specifies the root database table for this polling query	Wizard updates automatically. Do *not* modify this manually.
`SequencingTableName`	Name of the database table that is serving as the helper table	Wizard updates automatically. Do *not* modify this manually.
`SequencingTableNameFieldName`	Specifies the database column in the helper table that is used to store the root database table name	Wizard updates automatically. Do *not* modify this manually.
`SequencingTableValueFieldName`	Specifies the database column in the helper table that is used to store the sequence number of the last processed row in the root database table name	Wizard updates automatically. Do *not* modify this manually.

See "Deployment" on page 4-44 for details about the `service` section of the WSDL.

XML Schema Definition (XSD)

From a database schema, the wizard generates an XML schema representation of that object. This schema is used by the BPEL process.

For example, from the table named `Movies`, the following is generated:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<xs:schema
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/db/top/SelectAllByTitle"
xmlns="http://xmlns.oracle.com/pcbpel/adapter/db/top/SelectAllByTitle"
elementFormDefault="unqualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="MoviesCollection" type="MoviesCollection"/>
   <xs:element name="Movies" type="Movies"/>
   <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="director" type="xs:string" minOccurs="0"
                     nillable="true"/>
         <xs:element name="genre" type="xs:string" minOccurs="0" nillable="true"/>
         <xs:element name="rated" type="xs:string" minOccurs="0" nillable="true"/>
         <xs:element name="rating" type="xs:string" minOccurs="0"
                     nillable="true"/>
         <xs:element name="releaseDate" type="xs:dateTime" minOccurs="0"
                     nillable="true"/>
         <xs:element name="runTime" type="xs:double" minOccurs="0"
                     nillable="true"/>
         <xs:element name="starring" type="xs:string" minOccurs="0"
                     nillable="true"/>
         <xs:element name="status" type="xs:string" minOccurs="0"
                     nillable="true"/>
         <xs:element name="synopsis" type="xs:string" minOccurs="0"
                     nillable="true"/>
         <xs:element name="title" type="xs:string"/>
         <xs:element name="totalGross" type="xs:double" minOccurs="0"
                     nillable="true"/>
         <xs:element name="viewerRating" type="xs:string" minOccurs="0"
                     nillable="true"/>
      </xs:sequence>
   </xs:complexType>
   <xs:element name="findAllInputParameters" type="findAll"/>
   <xs:complexType name="findAll">
      <xs:sequence/>
   </xs:complexType>
   <xs:element name="SelectAllByTitleServiceSelect_titleInputParameters"
type="SelectAllByTitleServiceSelect_title"/>
   <xs:complexType name="SelectAllByTitleServiceSelect_title">
      <xs:sequence>
         <xs:element name="title" type="xs:string" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

This is a generated file. Changes to this file do not affect the behavior of the adapter. It is a declaration of the XML file that the database adapter produces and consumes.

You may need to modify the XSD file if you update the underlying `toplink_mappings.xml`. In that case, regenerate both files by rerunning the Adapter Configuration Wizard in edit mode.

The generated XSD flags all elements as optional with `minOccurs=0`, except for the primary key attributes, which are mandatory.

> **Note:** Do not manually modify the XSD file to configure the database adapter.

Deployment

The database adapter service that has been configured thus far is deployable as part of a business process. It is currently not deployable as a standalone. To deploy the adapter service, deploy the corresponding business process from Oracle BPEL Process Manager. Before you deploy, understand which run-time connection the adapter service will use and ensure that it is valid.

The adapter service WSDL refers to the run-time connection configured in the deployment descriptor of the database adapter. (In Oracle Application Server, it is `oc4j-ra.xml`). The relevant code example for the service WSDL follows:

```
<!-- Your runtime connection is declared in
J2EE_HOME/application-deployments/default/DbAdapter/oc4j-ra.xml.
These 'mcf' properties here are from your design time connection and save you from having to edit
that file and restart the application server if eis/DB/scott is missing.
These 'mcf' properties are safe to remove.
-->
<service name="get">
  <port name="get_pt" binding="tns:get_binding">
    <jca:address location="eis/DB/scott"
      UIConnectionName="scott"
      ManagedConnectionFactory="oracle.tip.adapter.db.DBManagedConnectionFactory"
      mcf.DriverClassName="oracle.jdbc.driver.OracleDriver"
      mcf.PlatformClassName="oracle.toplink.oraclespecific.Oracle9Platform"
      mcf.ConnectionString="jdbc:oracle:thin:@mypc.home.com:1521:orcl"
      mcf.UserName="scott"
      mcf.Password="7347B141D0FBCEA077C118A5138D02BE"
    />
  </port>
</service>
```

Note the following about the preceding deployment-related code example:

- The attribute `location` points to the run-time connection that is used at run time after the service is deployed. The location attribute links up to the database server JNDI name, which you may remember from the wizard.
- The run-time connection information corresponding to the given `location` value is specified in the adapter deployment descriptor (`oc4j-ra.xml`) file.
  - If the `location` specified in the WSDL exists in the `oc4j-ra.xml` file, then the database adapter uses the run-time connection information under the corresponding `location` entry in the `oc4j-ra.xml` file.
  - If the `location` specified in the WSDL does *not* exist in the `oc4j-ra.xml` file, then you have the following options:
    * Add a new entry to `oc4j-ra.xml` and specify the connection information. Restart the Oracle BPEL Server and deploy the business process *(recommended)*.
    * The modeled service functions normally if the run-time connection is the same as the design-time connection. All connection information is captured in the WSDL, as shown in the preceding code example in the `mcf.*` parameters. However, the database adapter runs in nonmanaged mode if you use this option, which is provided for development and testing. For production, use the recommended option.

The entry in the `oc4j-ra.xml` file is not generated by the wizard. You must manually update this file and restart the BPEL PM server for this to take effect. This file is created by the application server the first time Oracle BPEL Server comes up. Therefore, in a standalone installation, you do not see this file unless you start Oracle BPEL Server at least once.

Note: You must restart Oracle BPEL Server for updates in `oc4j-ra.xml` to take effect.

Location of the oc4j-ra.xml File

For the Oracle BPEL Process Manager for Developers installation, `oc4j-ra.xml` is at

```
Oracle_Home\integration\orabpel\system\appserver\oc4j\j2ee\home\
application-deployments\default\DbAdapter\oc4j-ra.xml
```

For the Oracle BPEL Process Manager for OracleAS Middle Tier installation, `oc4j-ra.xml` is at

```
Oracle_Home\j2ee\OC4J_BPEL\application-deployments\default\DBAdapter\oc4j-ra.xml
```

A sample entry from the `oc4j-ra.xml` file follows:

```
<connector-factory location="eis/DB/DBConnection1" connector-name="Database
Adapter">
<config-property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<config-property name="connectionString"
                 value="jdbc:oracle:thin:@localhost:1521:orcl"/>
<config-property name="userName" value="scott"/>
<config-property name="password" value="tiger"/>
<config-property name="minConnections" value="5"/>
<config-property name="maxConnections" value="5"/>
<config-property name="minReadConnections" value="1"/>
<config-property name="maxReadConnections" value="1"/>
<config-property name="usesExternalConnectionPooling" value="false"/>
<config-property name="dataSourceName" value=""/>
<config-property name="usesExternalTransactionController" value="false"/>
<config-property name="platformClassName"
                 value="oracle.toplink.internal.databaseaccess.Oracle9Platform"/>
<config-property name="usesNativeSequencing" value="true"/>
<config-property name="sequencePreallocationSize" value="50"/>
<config-property name="tableQualifier" value=""/>
</connector-factory>
```

The properties specified in the connection factory use OracleAS TopLink by translating to roughly 40 properties of an underlying TopLink login object. This is the equivalent of the OracleAS TopLink `sessions.xml` file.

Table 4–17, Table 4–18, and Table 4–19 show the configuration properties available to configure the run-time connection.

Table 4–17 Run-Time Connection Configuration Properties

Parameter	Description	Mechanism to Update
`driverClassName`	Name of the Java class for the JDBC driver being used	See "Third-Party Database Support" on page 4-52 for more information.
`connectionString`	JDBC Connection String	See "Third-Party Database Support" on page 4-52 for more information.
`userName`	Database login name	-
`password`	Database login password	-

Table 4–18 OracleAS TopLink Connection Pooling for the Database Adapter

Parameter	Description	Mechanism to Update
`minConnections`	Minimum number of connections in the pool	This is to use the adapter's local Toplink connection pool.
`maxConnections`	Maximum number of connections in the pool	This is to use the adapter's local Toplink connection pool.
`minReadConnections`	Minimum number of read-only connections in the pool	This is to use the adapter's local Toplink connection pool.
`maxReadConnections`	Maximum number of read-only connections in the pool	This is to use the adapter's local Toplink connection pool.

Table 4–19 Application Server Connection Pooling

Parameter	Description	Mechanism to Update
`dataSourceName`	Points to the data source configured in the application server	Use the application server connection pooling, for example: `java:comp/env/jdbc/myDataSourceName`
`usesExternalConnectionPooling`	Use application server's connection pooling	Use the application server connection pooling.
`usesExternalTransactionController`	Indicates if the application server's connection pool is managed or not	Use the application server connection pooling.
`platformClassName`	Indicates the database platform (see Table 4–20)	-
`usesNativeSequencing`	Indicates whether to use native database sequencing or not	-
`sequencePreallocationSize`	This corresponds to the `INCREMENT BY` value on the native database sequence	-
`tableQualifier`	Schema name to be used if the table names are not qualified in `toplink_mappings.xml`	Obsolete, because the wizard qualifies table names as it generates them.

If the application server connection pool is used, then most connection properties in this class are managed elsewhere.

Table 4–20 shows the advanced properties, which are database platform variables. Set the `DatabasePlatform` name to one of the following variables.

Table 4–20 Application Server Connection Pooling

Database	PlatformClassName
Oracle9+ (including 10*g*)	`oracle.toplink.oraclespecific.Oracle9Platform`
Oracle8	`oracle.toplink.internal.databaseaccess.OraclePlatform`
Oracle7	`oracle.toplink.internal.databaseaccess.DB2Platform`
DB2	`oracle.toplink.internal.databaseaccess.DB2Platform`
AS400	`oracle.toplink.internal.databaseaccess.DB2Platform`
Informix	`oracle.toplink.internal.databaseaccess.InformixPlatform`

Table 4–20 (Cont.) Application Server Connection Pooling

Database	PlatformClassName
Sybase	`oracle.toplink.internal.databaseaccess.SybasePlatform`
SQLServer	`oracle.toplink.internal.databaseaccess.SQLServerPlatform`
Any other database	`oracle.toplink.internal.databaseaccess.DatabasePlatform`

Advanced Properties

The following properties are configurable by using the managed connection factory entry in the `oc4j-ra.xml` file:

```
String connectionString
String userName
String password
String encryptionClassName
Integer minConnections
Integer maxConnections
Boolean useReadConnectionPool
Integer minReadConnections
Integer maxReadConnections
String dataSourceName
String driverClassName
Integer cursorCode
String databaseName
String driverURLHeader
Integer maxBatchWritingSize
String platformClassName
String sequenceCounterFieldName
String sequenceNameFieldName
Integer sequencePreallocationSize
String sequenceTableName
String serverName
Boolean shouldBindAllParameters
Boolean shouldCacheAllStatements
Boolean shouldIgnoreCaseOnFieldComparisons
Boolean shouldForceFieldNamesToUpperCase
Boolean shouldOptimizeDataConversion
Boolean shouldTrimStrings
Integer statementCacheSize
Integer stringBindingSize
String tableQualifier
Integer transactionIsolation
Boolean usesBatchWriting
Boolean usesByteArrayBinding
Boolean usesDirectDriverConnect
Boolean usesExternalConnectionPooling
Boolean usesExternalTransactionController
Boolean usesJDBCBatchWriting
Boolean usesNativeSequencing
Boolean usesNativeSQL
Boolean usesStreamsForBinding
Boolean usesStringBinding
```

The following properties appear in the `oracle.toplink.sessions.DatabaseLogin` object.

See OracleAS TopLink API reference information on `DBConnectionFactory` Javadoc and `DatabaseLogin` Javadoc at

http://download-east.oracle.com/docs/cd/B10464_02/web.904/b10491/index.html

To configure any of the preceding properties:

1. Add the following to the `ra.xml` file:

```
<config-property>
    <config-property-name>usesJDBCBatchWriting</config-property-name>
    <config-property-type>java.lang.Boolean</config-property-type>
    <config-property-value>true</config-property-value>
  </config-property>
```

   For Oracle BPEL Process Manager for Developers, `ra.xml` is at

   Oracle_Home\integration\orabpel\system\appserver\oc4j\j2ee\home\connectors\DbAdapter\DbAdapter\META-INF\ra.xml

   For Oracle BPEL Process Manager for OracleAS Middle Tier, `ra.xml` is at

   Oracle_Home\j2ee\OC4J_BPEL\connectors\DbAdapter\DbAdapter\META-INF\ra.xml

2. Add the following to the `oc4j-ra.xml` file:

```
<config-property name="usesJDBCBatchWriting" value="true"/>
```

3. Restart Oracle BPEL Server for the changes to take effect.

You can also update the factory default `oc4j-ra.xml` and `ra.xml` files before you deploy the database adapter. This way, you need only deploy once and do not need to restart the application server.

Performance

The database adapter is preconfigured with many performance optimizations. You can, however, make some changes to reduce the number of round trips to the database, as described in the following sections.

Outbound Write: Should You Use Merge, Write, or Insert?

If you run through the Adapter Configuration Wizard and select **Insert or Update**, you get a WSDL with the following operations: `merge` (default), `insert`, `update`, and `write`. The latter three call TopLink queries of the same name, avoiding advanced functionality that you may not need for straightforward scenarios. You can make the change by double-clicking an `invoke` activity and selecting a different operation. The `merge` is the most expensive, followed by the `write` and then the `insert`.

The `merge` first does a read of every element and calculates what has changed. If a row has not changed, it is not updated. The extra reads (for existence) and the complex change calculation add considerable overhead. For simple cases, this can be safely avoided; that is, if you changed only two columns, it does not matter if you update all five anyway. For complex cases, however, the opposite is true. If you have a master record with 100 details, but you changed only two columns on the master and two on one detail, the `merge` updates those four columns on two rows. A `write` does a write of every column in all 101 rows. Also, the `merge` may appear slower, but can actually relieve pressure on the database by minimizing the number of writes.

The `insert` operation is the most performant because it uses no existence check and has no extra overhead. You have no reads, only writes. If you know that you will do an insert most of the time, try an `insert`, and catch a Unique Key Constraint SQL exception inside your BPEL process, which can then perform a `merge` or `update` instead. For simple schemas this makes sense, but a `merge` is good if you have a mix

of new and existing objects (for instance, a master row containing several new details). The `update` is similar to the `insert`.

To monitor performance, you can enable debug logging and then watch the SQL for various inputs.

The OracleAS TopLink Cache: When Should You Use It?

Caching is an important performance feature of OracleAS TopLink. However, issues with stale data can be difficult to manage. By default, the database adapter uses a `WeakIdentityMap`, meaning a cache is used only to resolve cyclical references, and entries are quickly reclaimed by the Java virtual machine. If you have no cycles (and you ideally should not for XML), you can switch to a `NoIdentityMap`. The TopLink default is a `SoftCacheWeakIdentityMap`. This means that the most frequently used rows in the database are more likely to appear already in the cache.

For a knowledge article on caching, go to

`http://www.oracle.com/technology/tech/java/newsletter/november04.html`

Existence Checking

One method of performance optimization for `merge` is to eliminate check database existence checking. The existence check is marginally better if the row is new, because only the primary key is returned, not the entire row. But, due to the nature of `merge`, if the existence check passes, the entire row must be read anyway to calculate what changed. Therefore, for every row to be updated, you see one extra round trip to the database during `merge`.

It is always safe to use check cache on the root descriptor/table and any child tables if A is master and B is a privately owned child. If A does not exist, B cannot exist. And if A exists, all its Bs are loaded as part of reading A; therefore, check cache works.

Inbound (Polling): maxRaiseSize

On read (inbound) you can set `maxRaiseSize = 0` (unbounded), meaning that if you read 1000 rows, you will create one XML with 1000 elements, which is passed through a single Oracle BPEL Process Manager instance. A `merge` on the outbound side can then take all 1000 in one group and write them all at once with batch writing.

Inbound (Polling): Choosing a Polling Strategy

Your choice of polling strategy matters too. Avoid the delete polling strategy because it must individually delete each row. The sequencing polling strategy can destroy 1000 rows with a single update to a helper table.

Relationship Reading (Batch Attribute and Joined Attribute Reading)

Batch reading of one-to-many and one-to-one relationships is on by default. You can also use joined reading for one-to-one relationships instead, which may offer a slight improvement.

Connection Pooling

You can configure a connection pool if using either the adapter's local connection pool or an application server data source. Creating a database connection is an expensive operation. Ideally you should only exceed the `minConnections` under heavy loads. If you are consistently using more connections than that at once, then you may spend a lot of time setting up and tearing down connections. The database adapter also has a read connection pool. A read connection is more performant because there is no limit

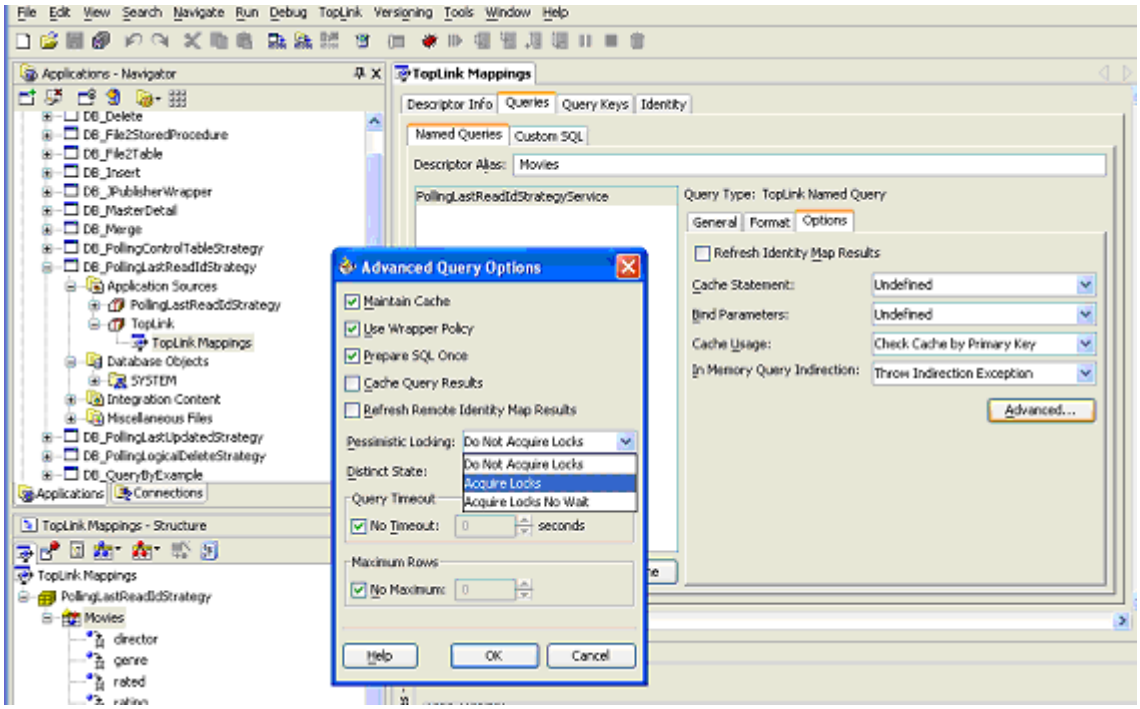
on how many users can use one connection for reading at the same time, a feature that most JDBC drivers support.

Inbound Distributed Polling

The database adapter is designed to scale to the number of unprocessed rows on the database. By default, it is possible to read and process one database row or 10,000 with as little as three round trips to the database. The most expensive operations are limited to a constant number. You can also configure the database adapter for a distributed environment.

Concurrency Control: Pessimistic Locking You can set a simple option that enables the database adapter to work safely in a distributed environment by making the first polling query acquire locks, as shown in Figure 4–24. In SQL terms, you are making your first `SELECT` into a `SELECT...FOR UPDATE`.

Figure 4–24 Acquiring Locks



The behavior of all polling strategies is as follows:

1. Read all unprocessed rows.
2. Process those rows.
3. Commit.

If any adapter instance performs step 1 while another instance is between steps 1 and 3, then duplicate processing occurs. Acquiring locks on the first operation and releasing them in commit solves this problem, and may naturally order the polling instances.

To enable pessimistic locking, run through the wizard once to create an inbound polling query. In the **Applications Navigator** window, expand **Application Sources,** then **TopLink**, and click **TopLink Mappings**. In the **Structure** window, click the table name. In **Diagram View**, click the following tabs: **TopLink Mappings, Queries,**

Named Queries, Options; then the **Advanced…** button, and then **Pessimistic Locking** and **Acquire Locks**. You see the message, “Set Refresh Identity Map Results?" If a query uses pessimistic locking, it must refresh the identity map results. Click **OK** when you see the message, "Would you like us to set Refresh Identity Map Results and Refresh Remote Identity Map Results to true?” Run the wizard again to regenerate everything. In the new `toplink_mappings.xml` file, you see something like this for the query: `<lock-mode>1</lock-mode>`.

Note the following:

- The preceding procedure works in conjunction with every polling strategy where the first operation is a read.
- For the sequencing-based polling strategies, the `SELECT FOR UPDATE` is applied to the `SELECT` on the helper table only. The `SELECT` on the polled table does not acquire locks because you do not have `write` access to those tables.
- If an adapter instance fails while polling records, those records are returned to the unprocessed pool (no commit happens).
- No individual adapter instance is special. In an ideal distributed system, coordination between instances is minimal (here effected with locking). No master acts as a weak link, and every part is identically configured and interchangeable.
- Other than the `SELECT FOR UPDATE`, no extra reads or writes are performed by the database adapter in a distributed environment.

Load Balancing: MaxTransactionSize and Pessimistic Locking After you enable pessimistic locking on a polling query, the `maxTransactionSize` activation property automatically behaves differently.

Assume that there are 10,000 rows at the start of a polling interval and that `maxTransactionSize` is 100. In standalone mode, a cursor is used to iteratively read and process 100 rows at a time until all 10,000 have been processed, dividing the work into 10,000 / 100 = 100 sequential transactional units. In a distributed environment, a cursor is also used to read and process the first 100 rows. However, the adapter instance will release the cursor, leaving 9,900 unprocessed rows (or 99 transactional units) for the next polling interval or another adapter instance.

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 need it.

Third-Party Database Support

The following sections discuss how to connect to a third-party database.

This section contains the following topics:

- Design Time
- Run Time

Note: You can use one vendor’s database for design time and another for run time because BPEL processes are database-platform neutral.

Design Time

Table 4–21 provides information for connecting to some common third-party databases.

To create a database connection when using a third-party JDBC driver:

1. Select **Connection Navigator** from **View**.
2. Right-click **Database** and select **New Database Connection**.
3. Click **Next** in the Welcome window.
4. Enter a connection name.
5. Select **Third Party JDBC Driver** from **Connection Type**.
6. Enter your username, password, and role information.
7. Click **New** for **Driver Class**.
8. Enter the driver name (for example, *some*`.jdbc.`*`Driver`*) for **Driver Class**.
9. Click **New** for **Library**.
10. Click **Edit** to add each JAR file of your driver to **Class Path**.
11. Click **OK** twice to exit the Create Library windows.
12. Click **OK** to exit the Register JDBC Driver window.
13. Enter your connection string name for **URL** and click **Next**.

    The connection URL varies across database vendors. Some sample entries appear in the deployment descriptor file (`oc4j-ra.xml`). See "Deployment" on page 4-44 for file location information.
14. Click **Test Connection**.
15. If the connection is successful, then click **Finish**.

Table 4–21 Information for Connecting to Third-Party Databases

Database	URL	Driver Class	Driver Jar
Oracle	URL: `jdbc:oracle:thin:@localhost:1521:orcl`	`oracle.jdbc.driver.OracleDriver`	`classes12.jar`
DB2	(net driver) `jdbc:db2:localhost:NAME`	(net driver) `COM.ibm.db2.jdbc.net.DB2Driver`	v8.1 (net driver) `.\IBM-net\db2java_81.zip, db2jcc_81.jar`

Table 4–21 (Cont.) Information for Connecting to Third-Party Databases

Database	URL	Driver Class	Driver Jar

SQL Server	(MS JDBC driver)
	`jdbc:microsoft:sqlserver://localhost\\NAME:1433;SelectMethod=cursor;databasename=???`
	(DataDirect driver)
	`jdbc:oracle:sqlserver://localhost`
		(MS JDBC driver)
		`com.microsoft.jdbc.sqlserver.SQLServerDriver`
		(DataDirect driver)
		`com.oracle.ias.jdbc.sqlserver.SQLServerDriver`
			(MS JDBC driver)
			`.\SQLServer2000\msbase.jar, msutil.jar, mssqlserver.jar`
			(DataDirect driver)
			`.\DataDirect\YMbase.jar, YMoc4j.jar, YMutil.jar, YMsqlserver.jar`

Sybase	(jconn driver)
	`jdbc:sybase:Tds:localhost:5001/NAME`
	(DataDirect driver)
	`jdbc:oracle:sybase://localhost:5001`
		(jconn driver)
		`com.sybase.jdbc2.jdbc.SybDriver`
		(DataDirect driver)
		`com.oracle.ias.jdbc.sybase.SybaseDriver`
			(jconn driver)
			`.\Sybase-jconn\jconn2.jar`
			(DataDirect driver)
			`.\DataDirect\YMbase.jar, YMoc4j.jar, YMutil.jar, YMsybase.jar`

Oracle Olite Database	URL: `jdbc:polite4:@localhost:100:orabpel`	Driver Class: `oracle.lite.poljdbc.POLJDBCDriver`	*`Oracle_Home`*`\integration\orabpel\lib\olite40.jar`

Note: To create an Oracle Lite database connection, follow the steps for a third-party JDBC driver exactly (because the existing wizard and libraries for the Oracle Lite database are not what you expect and require extra configuration). Table 4–21 provides information for connecting to an Oracle Lite database.

Run Time

At run time, you must put the driver JAR files in the application server class path. You can do this in the following ways:

- Edit the class path in the following files:

  (standalone) *`Oracle_Home`*`/integration/orabpel/system/appserver/oc4j/j2ee/home/config/application.xml`

  (regular middle tier) *`Oracle_Home`*`/j2ee/OC4J_BPEL/config/application.xml`

- Drop the JAR files into the following directories:

  (standalone) *`Oracle_Home`*`/integration/orabpel/system/appserver/oc4j/j2ee/home/applib`

  (regular middle tier) *`Oracle_Home`*`/j2ee/OC4J_BPEL/applib`

Stored Procedure and Function Support

This section describes how the database adapter supports the use of stored procedures and functions for Oracle databases only.

This section contains the following topics:

- Design Time: Using the Adapter Configuration Wizard

- Design Time: WSDL and XSD Generation
- Run Time: Before Stored Procedure Invocation
- Run Time: After Stored Procedure Invocation
- Advanced Topics

Design Time: Using the Adapter Configuration Wizard

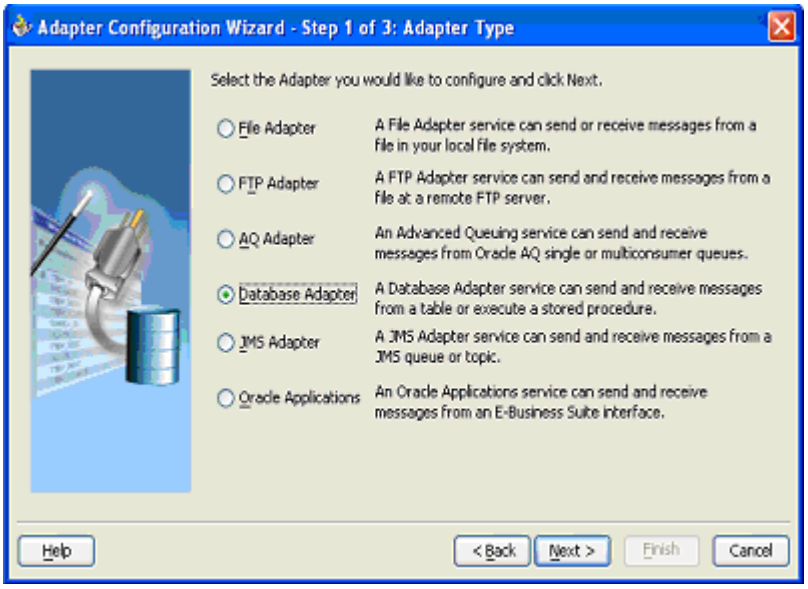
The Adapter Configuration Wizard – Stored Procedures is used to generate an adapter service WSDL and the necessary XSD. The adapter service WSDL encapsulates the underlying stored procedure or function as a Web service with a WSIF JCA binding. The XSD describes the procedure or function, including all the parameters and their types. This XSD provides the definition used to create instance XML that is submitted to the database adapter at run time.

Using Top-Level Standalone APIs

This section describes how to use the 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. See "The Adapter Configuration Wizard" on page 4-18 if you are not familiar with how to start the wizard.

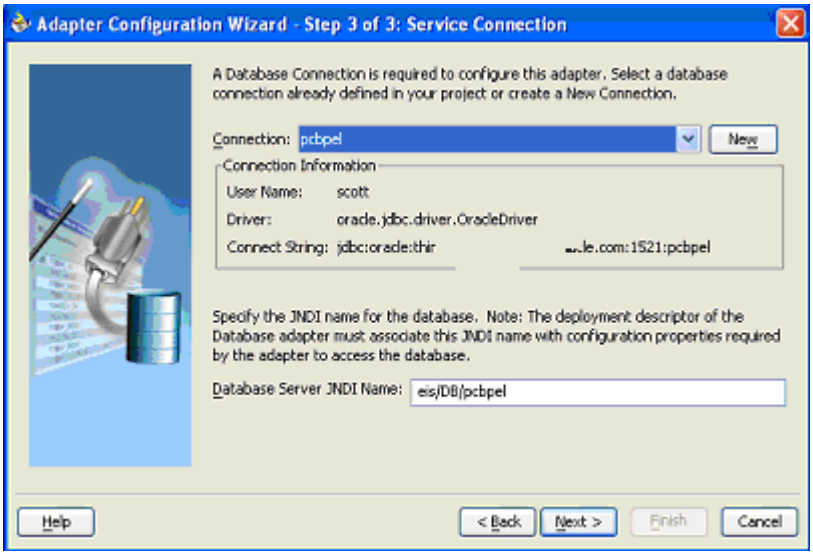
In the wizard, select **Database Adapter**, as shown in Figure 4–25.

Figure 4–25 Selecting the Database Adapter in the Adapter Configuration Wizard



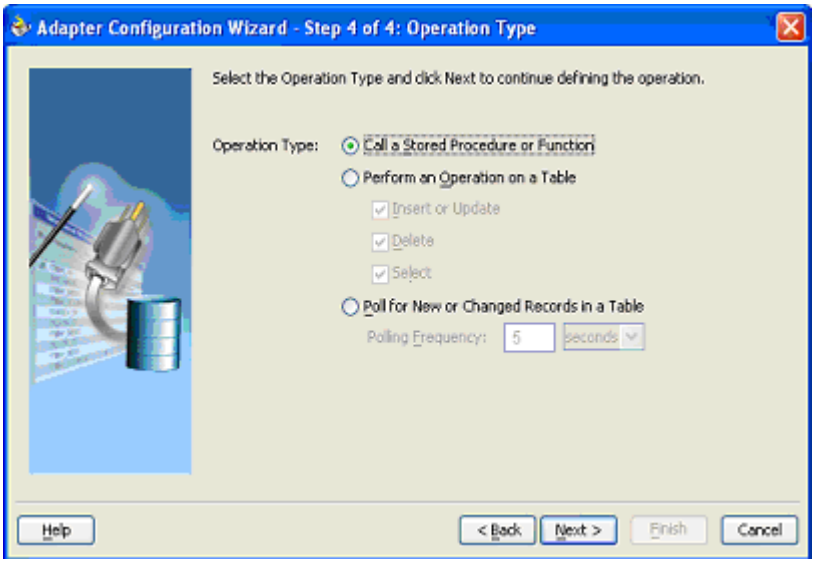
After entering a service name (for example, `ProcedureProc`) and an optional description for the service, you associate a connection with the service, as shown in Figure 4–26. You can select an existing connection from the list or create a new connection.

Figure 4–26 Setting the Database Connection in the Adapter Configuration Wizard



For the **Operation Type**, select **Call a Stored Procedure or Function**, as shown in Figure 4–27.

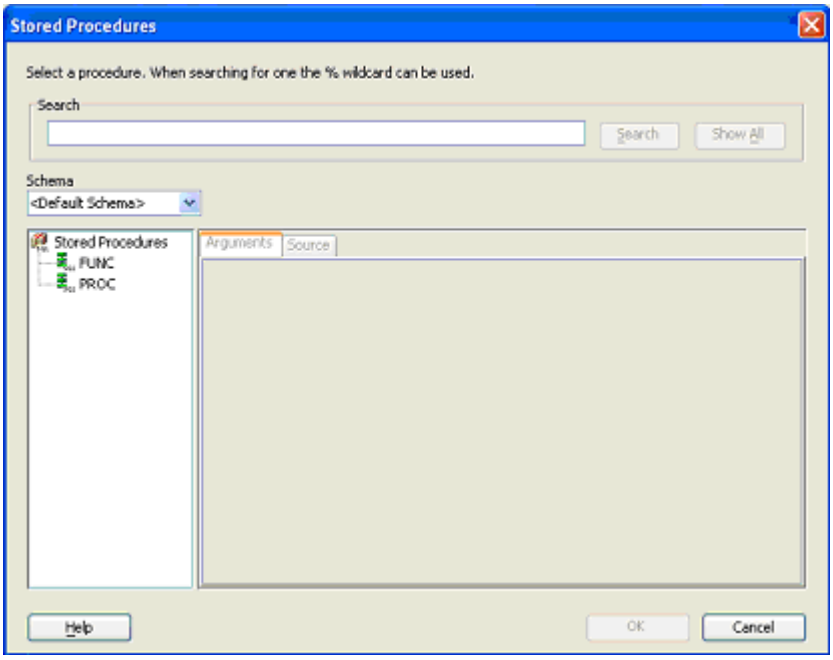
Figure 4–27 Calling for a Stored Procedure or Function in the Adapter Configuration Wizard



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, then 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, shown in Figure 4–28.

Figure 4–28 Searching for a Procedure or Function



Select a schema from the list or select `<Default Schema>`. The available procedures are 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 4–29 shows how you can select the **PROC** procedure and click the **Arguments** tab. The **Arguments** tab displays the parameters of the procedure, including their names, type, mode (`IN`, `IN/OUT` or `OUT`) and the numeric position of the parameter in the definition of the procedure.

Figure 4–29 Viewing the Arguments of a Selected Procedure

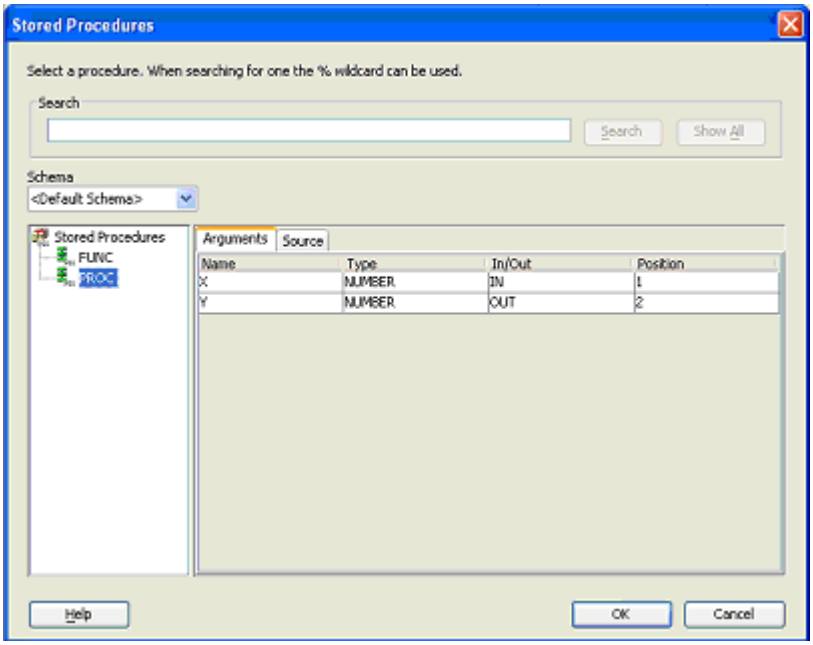
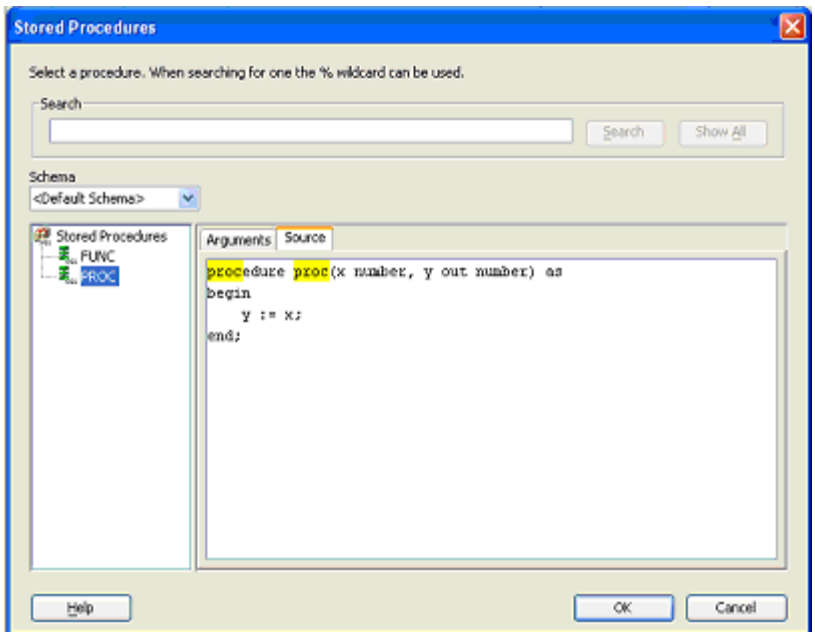


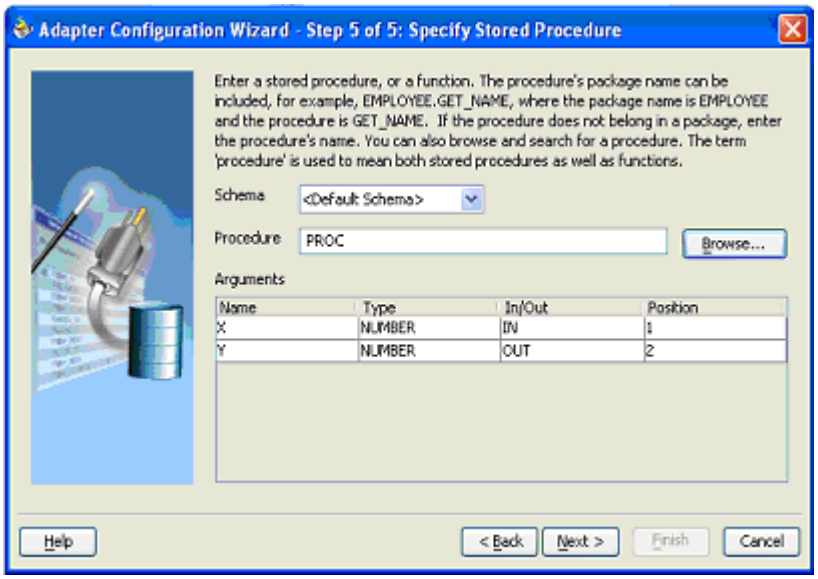
Figure 4–30 shows how the **Source** tab displays the code that implements the procedure. Text that matches the name of the procedure is highlighted.

Figure 4–30 Viewing the Source Code of a Selected Procedure



After you select a procedure or function and click **OK**, information about the API is displayed, as shown in Figure 4–31. Use **Back** or **Browse** to make revisions, or **Next** followed by **Finish** to conclude.

Figure 4–31 Viewing Procedure or Function Details in the Adapter Configuration Wizard



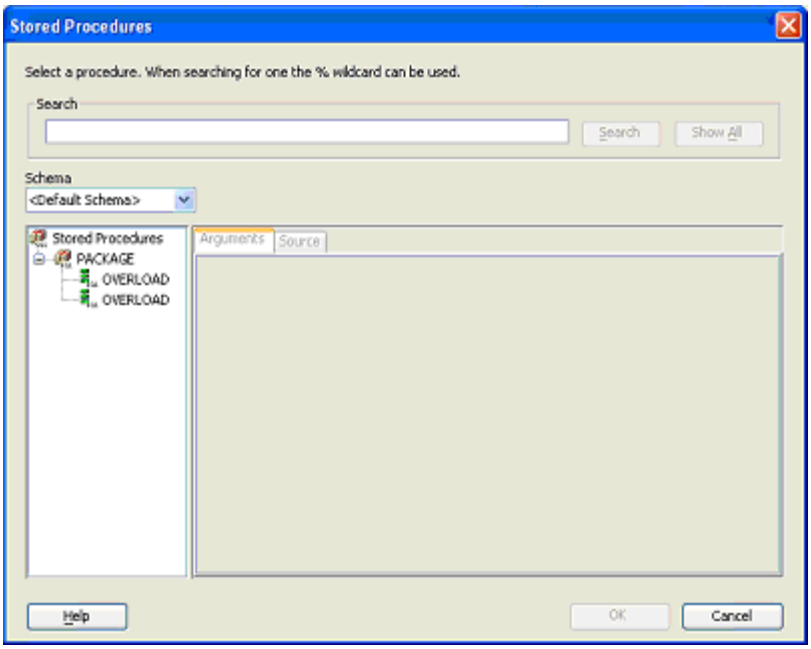
When you have finished using the Adapter Configuration Wizard, two files are added to the existing project: *servicename*`.wsdl` (for example, `ProcedureProc.wsdl`) and the generated XSD. The generated XSD file is named *schema_package_procedurename*`.xsd`. In this case, `SCOTT_PROC.xsd` is the name of the generated XSD file.

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 4–32.

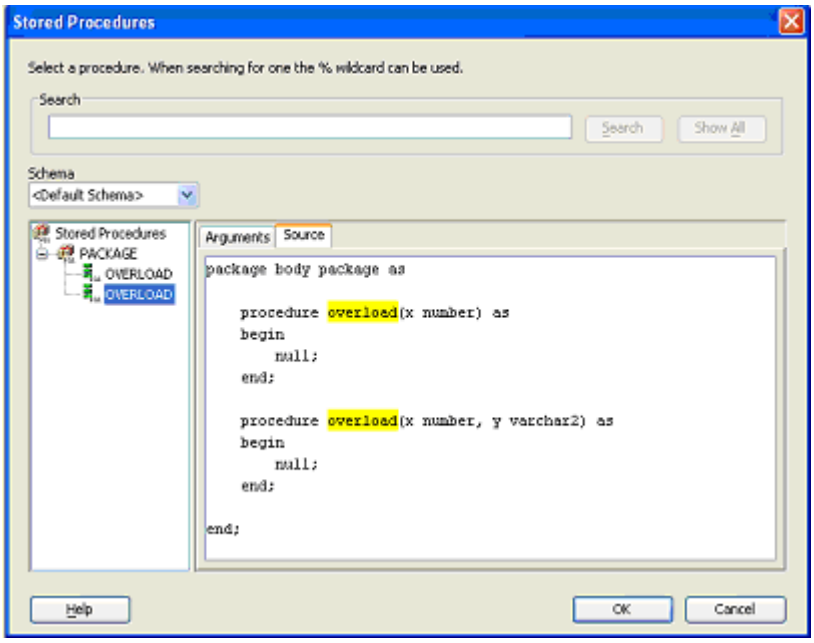
APIs that have the same name but different parameters are called overloaded APIs. As shown in Figure 4–32, the package called **PACKAGE** has two overloaded procedures called **OVERLOAD**.

Figure 4–32 A Package with Two Overloaded Procedures



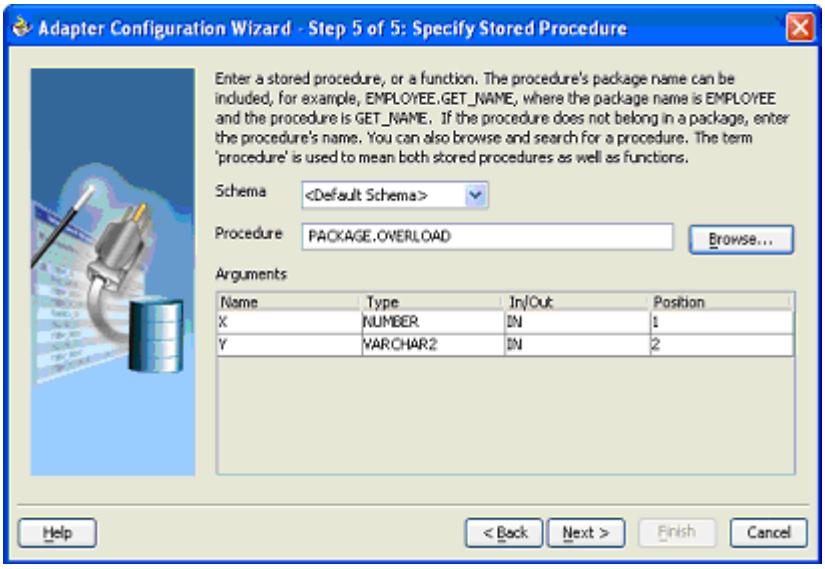
As Figure 4–33 shows, the code for the entire PL/SQL package is displayed, regardless of which API from the package is selected when you view the **Source** tab. Text that matches the name of the procedure is highlighted.

Figure 4–33 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 4–34. The schema, procedure name, and parameter list are displayed. Note how the procedure name is qualified with the name of the package (**PACKAGE.OVERLOAD**). Use **Back** or **Browse** to make revisions, or **Next** followed by **Finish** to conclude.

Figure 4–34 Viewing Procedure or Function Details in the Adapter Configuration Wizard



When you have finished using the Adapter Configuration Wizard, two files are added to the existing project: `Overload.wsdl` and `SCOTT_PACKAGE_OVERLOAD_2.xsd`. The `_2` appended after the name of the procedure in the XSD filename differentiates the overloaded APIs.

Design Time: WSDL and XSD Generation

The Adapter Configuration Wizard – Stored Procedures is capable of creating a WSDL and a valid XSD 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, and their relationship with each other.

The WSDL–XSD Relationship

In the paragraphs that follow, the operation name, `ProcedureProc`, and procedure name, `PROC`, are taken from an example cited previously (see Figure 4–31 on page 4-59). The generated WSDL imports the XSD.

```
<types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema">
    <import
namespace="http://xmlns.oracle.com/pcbpel/adapter/db/SCOTT/PROC/"
      schemaLocation="SCOTT_PROC.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 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 for specifying elements only if there are any `IN/OUT` or `OUT` parameters. Note that `IN/OUT` parameters appear in both root elements.

These root elements are represented in the XSD as an unnamed `complexType` definition whose sequence includes one element for each parameter. If there are no `IN` or `IN/OUT` parameters, the `InputParameters` root element is still created; however,

the complexType is empty. A comment in the XSD indicates that there are no such parameters. An example of one of these 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="db:InputParameters"/>
</message>
<message name="args_out_msg"
  <part name="OutputParameters" element="db:OutputParameters"/>
</message>
```

The db namespace is the same as the targetNamespace of the generated XSD. Note that 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.

An operation is defined in the WSDL whose name is the same as the adapter service and whose input and output messages are defined in terms of these two message types.

```
<portType name="ProcedureProc_ptt">
  <operation name="ProcedureProc">
    <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. 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/ProcedureProc/"
```

The root elements in the XSD 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. 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 that gets generated after the stored procedure has executed. It holds the values of any IN/OUT and OUT parameters.

Supported Primitive Datatypes

Many primitive datatypes have well-defined mappings and therefore are supported by both the design-time and run-time components. In addition, you can use user-defined types such as VARRAY, nested tables, and OBJECT. Table 4–22 lists the primitive datatypes supported by the database adapter for stored procedures.

Table 4–22 Primitive Datatypes Supported by the Database Adapter for Stored Procedures

SQL or PL/SQL Type	XML Schema Type
BINARY_DOUBLE DOUBLE PRECISION	double
BINARY_FLOAT FLOAT REAL	float
BINARY_INTEGER INTEGER SMALLINT	int
BLOB LONG RAW RAW	base64Binary
CHAR CLOB LONG VARCHAR2	string
DATE TIMESTAMP	dateTime
DECIMAL NUMBER	decimal

The datatype of a parameter defined as being of a certain type not shown in this table, such as PLS_INTEGER, is often represented by a type that is listed in the table. For PLS_INTEGER, the underlying database type is BINARY_INTEGER; therefore, those mappings are used. Thus, the wizard can usually handle a parameter whose primitive type is not shown above, but which has an underlying type that is supported.

The context in which a type is used is also relevant to how the type is treated. For example, a parameter of a stored procedure whose type is declared as INTEGER is actually represented as NUMBER; therefore, the NUMBER mappings take effect. In contrast, an attribute of a user-defined OBJECT type whose type is INTEGER is represented by INTEGER; therefore, those mappings are used.

Generated XSD Attributes

Table 4–23 lists the generated XSD attributes.

Table 4–23 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
minOccurs	minOccurs="0"	Minimum occurrences

Table 4–23 (Cont.) Generated XSD Attributes

Attribute	Example	Purpose
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 run-time components to ensure that the parameters are arranged in the proper order. Parameter indices 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 minOccurs value is set to 0 to allow for an IN parameter to be removed from the XML. 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, 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 schema validation, an element such as this, which does have a null value, 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 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.

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 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. Note that 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. 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.

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 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. 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, 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 aforementioned user-defined types. This means that 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 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 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 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 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 wizard generates `complexType` definitions for `FOO` and `BAR` as already 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 together, 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:

```
<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`. Together with the `type` attribute, the element definition indicates that `Y` is a reference to `FOO`.

Note that 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.

Run Time: Before Stored Procedure Invocation

This section discusses important considerations of stored procedure support.

Value Binding

Consider the extraction of values from the XML and how the run time works given those values. The possible cases for data in the XML corresponding to the value of a parameter whose type is one of the supported primitive datatypes value are as follows:

1. The value of an element is specified (for example, `<X>100</X>`).
2. The value of an element is not specified (for example, `<X/>`).
3. The value is explicitly specified as null (for example, `<X xsi:nil="true"/>`)
4. The element is not specified in the XML at all.

Each case is handled differently.

In the first case, the value is taken from the XML 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 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 the same as passing null for parameter `X`.

The fourth case is allowed only when the parameter corresponding to the missing element in the XML is an `IN` parameter and has a declared default clause in the definition of the stored procedure. For example,

```
SQL> CREATE PROCEDURE PROC (X IN INTEGER DEFAULT 0) AS BEGIN … END;
```

Here, no value is bound to the parameter. In fact, the parameter is completely 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 the four cases:

1. "BEGIN PROC (X=>?); END;" - X = 100
2. "BEGIN PROC (X=>?); END;" - X = null
3. "BEGIN PROC (X=>?); END;" - X = null
4. "BEGIN PROC (); END;" - X = 0

These general semantics also apply to item values of a collection or attribute values of an OBJECT whose types are one of the supported primitive datatypes. 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; therefore, the VARRAY instance has only four elements.

Given 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 behavior is that the value of 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, 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. Like 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, the xsi:nil attribute must be added to the element in the XML, as in <X xsi:nil="true"/>.

Datatype Conversions

This section describes the conversion of datatypes such as CLOB, DATE, TIMESTAMP, and binary datatypes including RAW, LONG RAW and BLOB.

For CLOB parameters, a temporary CLOB is first created. The data extracted from the XML is then written to it before binding the CLOB to its corresponding parameter. The temporary CLOB is freed when the interaction completes. For other character types, such as CHAR and VARCHAR2, the data is simply extracted and bound as necessary. Note that it is possible to bind an XML document to a CLOB (or VARCHAR2 if it is large enough). However, appropriate substitutions for <, >, and so on, must first be made (for example, < for < and > for >).

Note that the XML schema type, dateTime, represents both DATE and TIMESTAMP. This means that the XML values for both datatypes 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.

Data for binary datatypes 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. This is then bound either directly, as is the case with RAW and LONG RAW parameters, or is used to create a temporary BLOB, which is then bound to its associated BLOB parameter. The temporary BLOB is freed when the interaction completes.

Conversions for the remaining datatypes 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.

Datatype Conversions

Conversions of the data retrieved are straightforward. However, BLOB, CLOB (and other character data), as well as RAW and LONG RAW conversions, 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. This means that character data, as for types like CLOB, CHAR, and VARCHAR2, is massaged as needed to make any required substitutions so that the value is valid and can be placed in the XML 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 types, is retrieved as a byte array. For BLOBs, 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 for the corresponding element. The MimeUtility decode API must be used to decode this value back into a byte array.

Conversions for the remaining datatypes 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. This means that the xsi namespace is declared in the XML that is generated. Generated XML for a procedure PROC, which has a single OUT parameter, X, whose value is null, looks as follows:

```
<db:OutputParameters … xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <X xsi:nil="true"/>
</db:OutputParameters>
```

The db namespace is also declared (that is, xmlns:db="..."). Note that 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 XML generated in OutputParameters.

Advanced Topics

This section discusses scenarios for types that are not supported directly using the stored procedure functionality that the database adapter provides. The following sections describe workarounds that address the need to use these datatypes.

Support for REF CURSOR

Neither the design-time nor run-time components support REF CURSOR types directly. The solution is to use a collection of an OBJECT type. Because the number of rows returned by a REF CURSOR is usually unknown, it is best to use a nested table as the collection type. This solution involves using a Java stored procedure to convert a ResultSet into an instance of the declared collection type. A sample tutorial illustrating this is provided in the following directory:

Oracle_Home\integration\orabpel\samples\tutorials\122.DBAdapter\ResultSetConverter

Support for PL/SQL BOOLEAN

JDBC does not support passing BOOLEAN parameters. Therefore, the run-time component, as in the case of REF CURSOR, cannot directly support parameters declared to be of this type. The solution is to define a wrapper procedure that substitutes a supported datatype, such as INTEGER, for each BOOLEAN parameter. For example, suppose you defined the following stored procedure:

```
CREATE PROCEDURE BOOLPROC (B BOOLEAN) AS BEGIN...END;
```

A wrapper procedure can be written as follows:

```
CREATE PROCEDURE BOOLWRAP (X INTEGER) AS
BEGIN
   IF (X = 1) THEN BOOLPROC (TRUE);
   ELSE BOOKPROC (FALSE);
   END IF;
END;
```

See *Oracle Database JDBC Developer's Guide and Reference* for more information.

Support for PL/SQL RECORD

Support for PL/SQL RECORD parameters cannot be provided directly. Like the BOOLEAN case, however, the solution involves creating wrapper procedures. For each RECORD parameter, an OBJECT type is defined with attributes that correspond with the fields of the RECORD type. Use JDeveloper BPEL Designer to create a SQL file that contains the necessary OBJECT type definitions, as well as APIs for converting from the RECORD type to the OBJECT type, and vice-versa. A wrapper is written that substitutes each RECORD type with its corresponding OBJECT for each such parameter. The wrapper executes the underlying stored procedure, using the conversion APIs to do the necessary conversions in both directions. For a sample tutorial, go to

Oracle_Home\integration\orabpel\samples\tutorials\122.DBAdapter\JPublisherWrapper

Use Case for Creating and Configuring a Stored Procedure in JDeveloper BPEL Designer

This tutorial describes how to integrate a stored procedure into Oracle BPEL Process Manager with JDeveloper BPEL Designer. Other tutorials that demonstrate stored procedures and functions are File2StoredProcedure, JPublisherWrapper, and ResultSetConverter. Go to

Oracle_Home\integration\orabpel\samples\tutorials\122.DBAdapter

This section contains the following topics:

- Creating a Stored Procedure
- Creating a Database Connection
- Creating a Workspace and a Greeting Process
- Creating a Partner Link
- Creating an Invoke Activity
- Creating an Initial Assign Activity
- Creating a Second Assign Activity

- Validating, Compiling, and Deploying the Greeting Process
- Running the Greeting Process

Note: Stored procedures are supported for Oracle databases only.

Creating a Stored Procedure

1. Connect to the `scott` schema of the Oracle database using SQL*Plus. This is the schema in which to create the stored procedure. This example assumes `tiger` is the password.

   ```
   sqlplus scott/tiger
   ```

2. Create the stored procedure (note the blank space after `Hello`):

   ```
   SQL> CREATE PROCEDURE HELLO (NAME IN VARCHAR2, GREETING OUT VARCHAR2) AS
     2  BEGIN
     3      GREETING := 'Hello ' || NAME;
     4  END;
     5  /

   Procedure created.
   ```

Creating a Database Connection

Use the Create Database Connection Wizard in JDeveloper BPEL Designer to create a connection to the scott schema in which you created the stored procedure.

1. Go to JDeveloper BPEL Designer.
2. Select **Connection Navigator** from the **View** main menu.
3. Right-click **Database** in the **Connection Navigator** window and select **New Database Connection**.

   This starts the Create Database Connection Wizard.
4. Click **Next** on the Welcome window.
5. Enter a name (for example, **myConnection**) in the **Connection Name** field of the Type window.
6. Select the database connection type (for example, **Oracle (JDBC)**) from the **Connection Type** list, and click **Next**.
7. Enter **scott** in the **Username** field of the Authentication window.
8. Enter the password for scott in the **Password** field (**tiger** for this example).
9. Leave the remaining fields as they are, and click **Next**.
10. Enter the following connection information. If you do not know this information, contact your database administrator.

Field	Example of Value
Driver	**thin**
Host Name	**localhost**
JDBC Port	**1521**
SID	**ORCL**

11. Click **Next**.
12. Click **Test Connection** on the Test window.

    If the connection was successful, the following message appears:

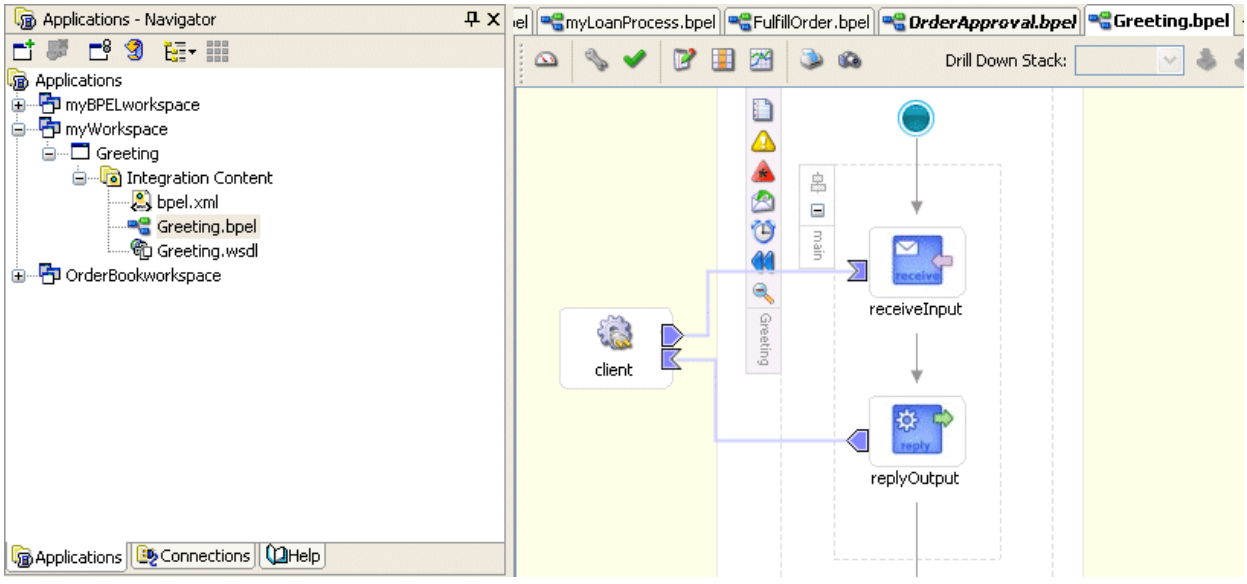
    ```
    Success!
    ```

13. Click **Finish**.

Creating a Workspace and a Greeting Process

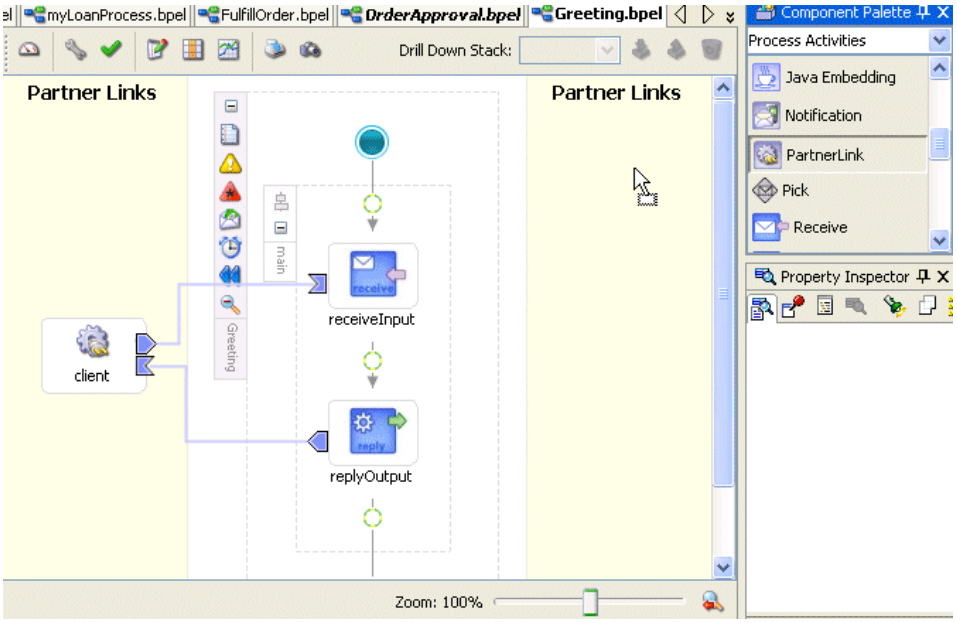
1. Select **Application Navigator** from the **View** main menu in JDeveloper BPEL Designer.
2. Select **New** from the File main menu.
3. Double-click **Workspace** in the **Items** window to display the Create Workspace window.
4. Enter a name (for example, **myWorkspace**) in the **Workspace Name** field and accept the default path in the **Directory Name** field.
5. Deselect the **Add a New Empty Project** check box.
6. Click **OK**.
7. Right-click this new workspace in the **Applications Navigator** section.
8. Select **New Project**.
9. Double-click **BPEL Process Project** in the **Items** window to display the BPEL Process Project window.
10. Enter **Greeting** in the **BPEL Process Name** field.
11. Select **Synchronous BPEL Process** from the **Template** list.
12. Leave the **Use Default** check box selected.
13. Click **OK**.

    The **bpel.xml**, **Greeting.bpel**, and **Greeting.wsdl** files are created in the **Applications Navigator**.



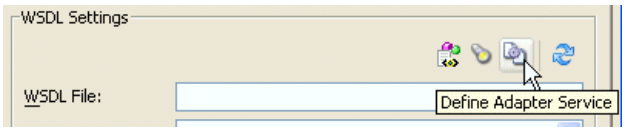
Creating a Partner Link

1. Ensure that **Process Activities** is selected in the drop-down list of the **Component Palette** section in the upper right section of JDeveloper BPEL Designer.
2. Drag and drop a **PartnerLink** activity onto the right side of the designer window under the **Partner Links** header.



The Create Partner Link window appears.

3. Enter **Hello** in the **Name** field.
4. Click the third icon at the top (the **Define Adapter Service** icon). This starts the Adapter Configuration Wizard.



5. Click **Next** on the Welcome window.
6. Select **Database Adapter** on the Adapter Type window and click **Next**.
7. Enter **Hello** in the **Service Name** field on the Service Name window. This is the same name as that of the partner link.
8. Click **Next**.
9. Select the database connection in the **Connection** field on the Service Connection page that you created for the `scott` schema in "Creating a Database Connection" on page 4-72.
10. Ensure that **eis/DB/*connection_name*** displays in the **Database Server JNDI Name** field. The ***connection_name*** is the name you selected in the **Connection** field and the connection to the `scott` schema that you created in "Creating a Database Connection" on page 4-72. The name is case sensitive. Ensure that it correctly matches the case of the connection name.
11. Click **Next**.

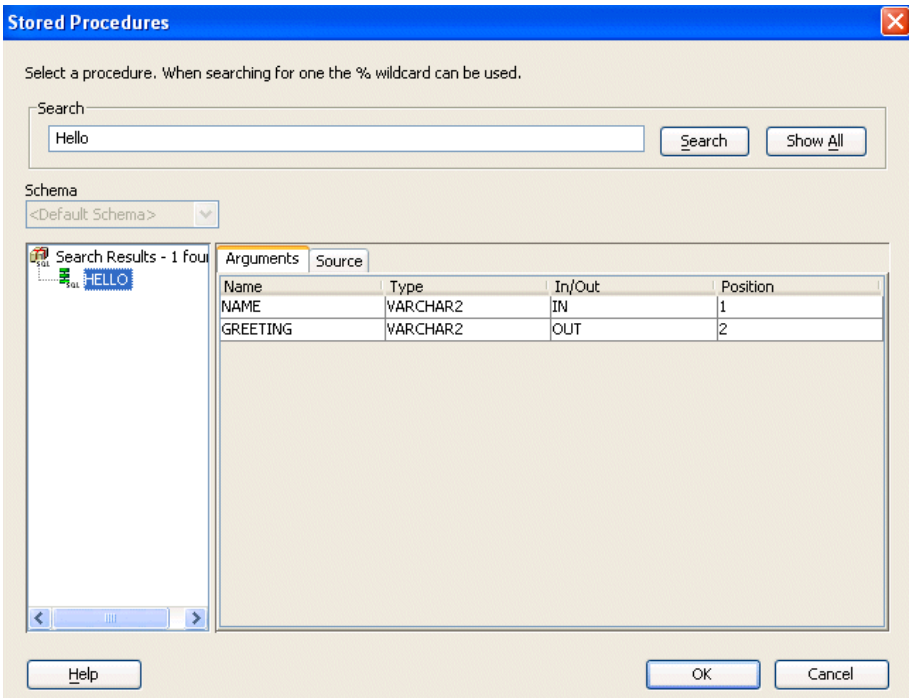
12. Select **Call a Stored Procedure or Function** on the Operation Type window.
13. Click **Next**.

    The Specify Stored Procedure window appears.
14. Click **Browse** to the right of the **Procedure** field.

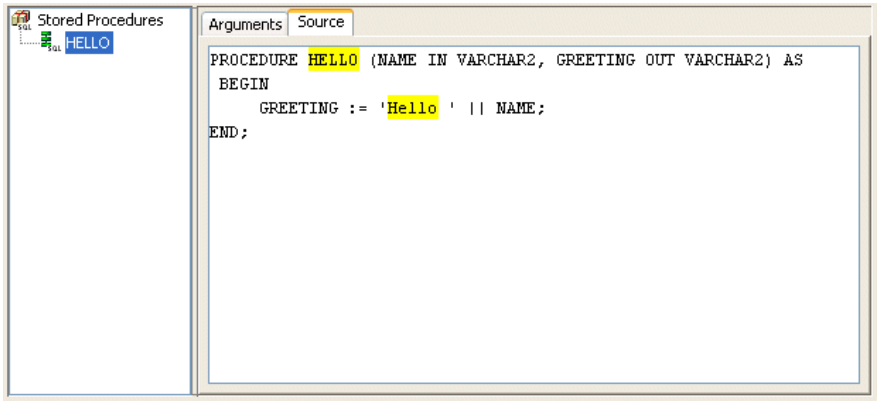
    The Stored Procedures window appears.
15. Leave **<Default Schema>** selected in the **Schema** list. This defaults to the `scott` schema in which the stored procedure is defined.
16. Select **Hello** in the **Stored Procedures** navigation tree.

> **Note:** As an alternative, you can also enter **Hello** in the **Search** field, click **Search** to display this stored procedure for selection in the **Stored Procedures** navigation tree, and then select it.

The **Arguments** tab displays the parameters of the stored procedure.

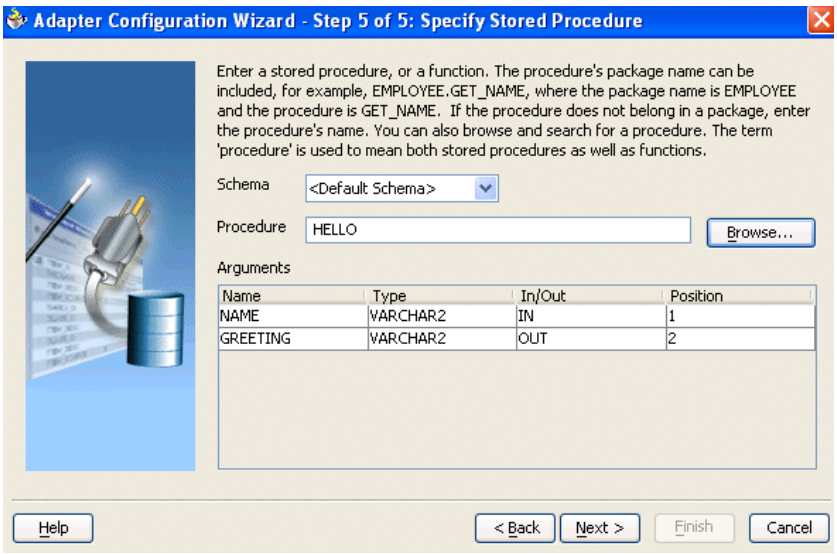


17. Click the **Source** tab to display the **Hello** stored procedure source code. You entered this syntax when you created the stored procedure using SQL*Plus in "Creating a Stored Procedure" on page 4-72.



18. Click **OK**.

    The Specify Stored Procedure window displays your selections. They appear as they did when the **Arguments** tab displayed in the Stored Procedures window in Step 16.



19. Click **Next**.

20. Click **Finish** to complete adapter configuration.

    The Create Partner Link window is automatically completed. The window looks as follows:

Field	Value
Name	**Hello**
WSDL File	**file:/c:/OraBPELPM/integration/jdev/jdev/mywork/myWorkspace/Greeting/Hello.wsdl** **Note: OraBPELPM** is the Oracle home directory used in this example. In addition, this directory path with a drive letter represents an example on Windows operating systems. If running this tutorial on Unix operating systems, your directory path varies.
Partner Link Type	**Hello_plt**
My Role	Leave unspecified.

Field	Value
Partner Role	**Hello_role**

21. Click **Apply**.
22. Click **OK**.
23. Select **Save All** from the **File** main menu.

    The following files appear under **Greeting** > **Integration Content** in the **Applications Navigator**. These files contain the parameters you specified with the Adapter Configuration Wizard.

    - **Hello.wsdl**

      Corresponds with the new stored procedure partner link

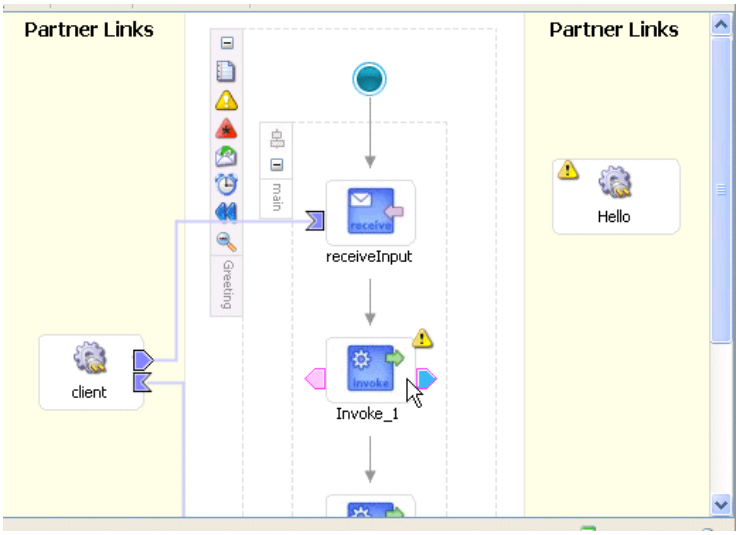
    - **SCOTT_HELLO.xsd**

      Provides the definition of the stored procedure, including its parameters

Creating an Invoke Activity

You now create an **invoke** activity to specify an operation you want to invoke for the service (identified by the **Hello** partner link).

1. Drag and drop an **invoke** activity below the **receiveInput Receive** activity.



2. Double-click the **invoke** activity to display the Invoke window.
3. Enter the following details:

Field	Value
Name	**Greet**
Partner Link	Hello

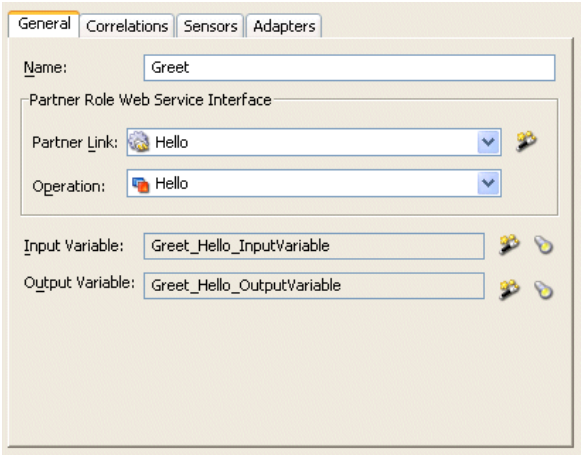
   The **Operation** (**Hello**) field is automatically filled in.

4. Click the first icon to the right of the **Input Variable** field. This is the automatic variable creation icon.

A variable named **Greet_Hello_InputVariable** automatically appears in the **Name** field. This variable provides the value for the `in` parameter of the stored procedure. The type is **http://xmlns.oracle.com/pcbpel/adapter/db/Hello/}args_in_msg**.

5. Ensure that **Global Variable** is selected.
6. Click **OK** on the Create Variable window.
7. Click the first icon to the right of the **Output Variable** field.
8. A variable named **Greet_Hello_OutputVariable** automatically appears in the **Name** field. This variable stores the value of the `out` parameter of the procedure after it executes. The type is **http://xmlns.oracle.com/pcbpel/adapter/db/Hello/}args_out_msg**.
9. Ensure that **Global Variable** is selected.
10. Click **OK** in the Create Variable window.

    Your selections for the Invoke window appear.



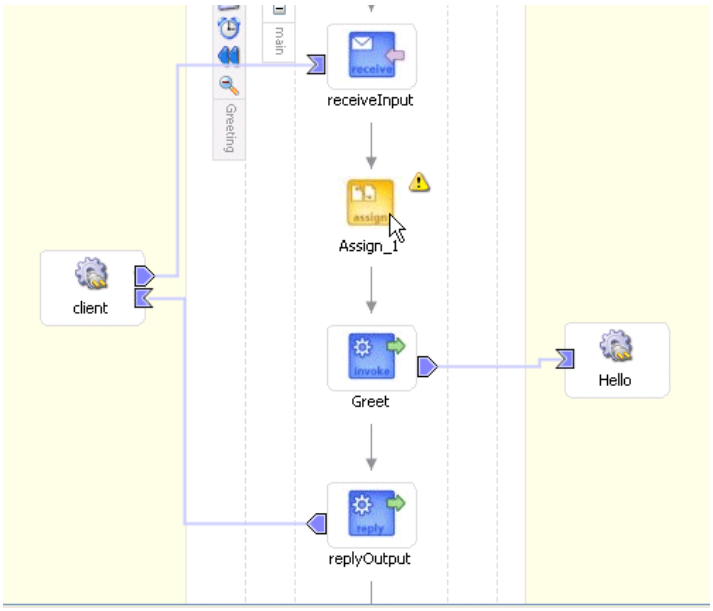
11. Click **OK** in the Invoke window.
12. Select **Save All** from the **File** main menu.

    The process displays a link from the **Greet Invoke** activity to the **Hello** partner link.

Creating an Initial Assign Activity

You now create an Assign activity to assign the input value to the `in` parameter of the stored procedure.

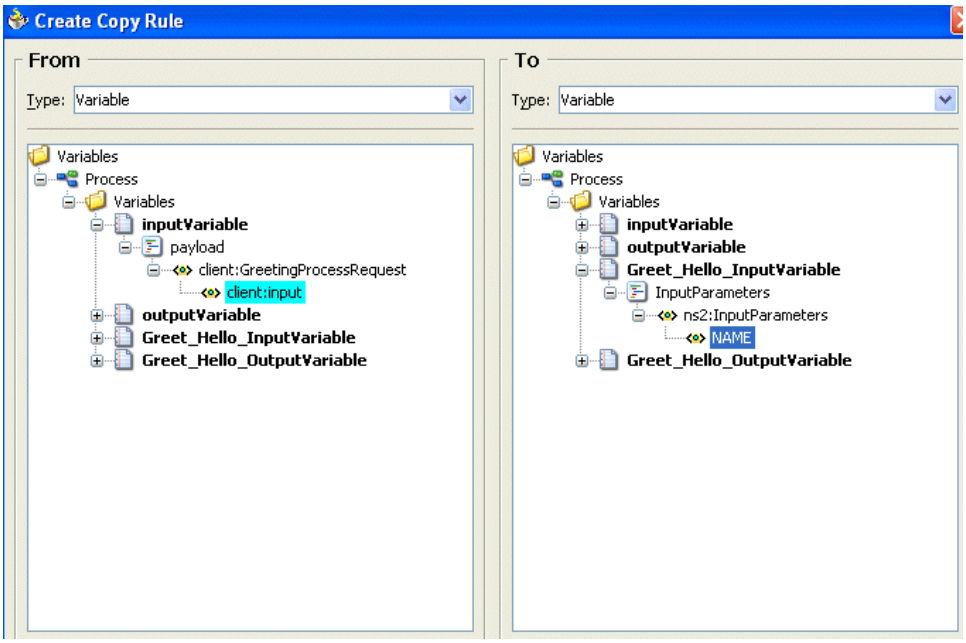
1. Drag and drop an **Assign** activity from the **Component Palette** section to above the **Greet Invoke** activity.



2. Double-click the **assign** icon to display the Assign window.
3. Click the **General** tab.
4. Enter **Input** in the **Name** field.
5. Click **Apply**.
6. Click the **Copy Rules** tab.
7. Click **Create** to display the Create Copy Rule window.
8. Enter the following values:

Field	Value
From	
▪ **Type**	**Variable**
▪ **Variables**	Expand and select **Variables**, then **inputVariable**, then **payload**, then **client:GreetingProcessRequest**, and then **client:input**.
To	
▪ **Type**	**Variable**
▪ **Variables**	Expand and select **Variables**, then **Greet_Hello_InputVariable**, then **InputParameters**, then **ns2:InputParameters**, and then **NAME**.

The Create Copy Rule window appears as follows:

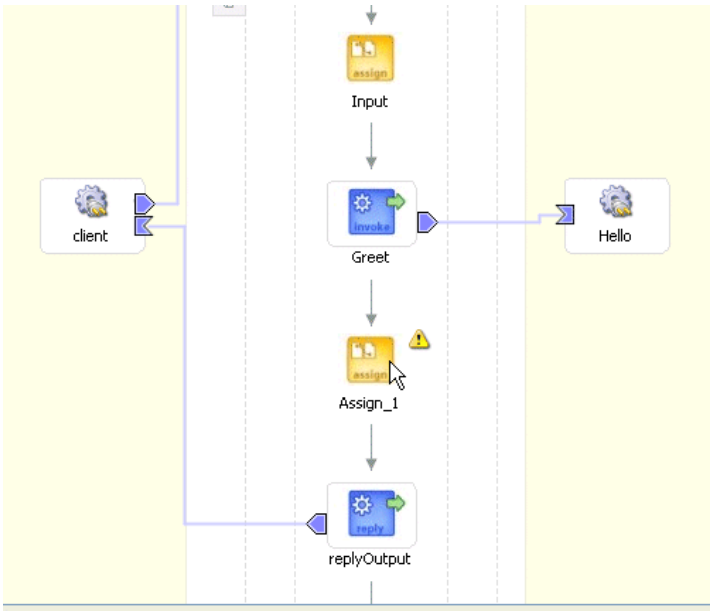


9. Click **OK** to close the Create Copy Rule window.
10. Click **OK** to close the Assign window.
11. Select **Save All** from the **File** main menu.

Creating a Second Assign Activity

You now create an Assign activity to retrieve the value of the `out` parameter of the stored procedure.

1. Drag and drop an **Assign** activity from the **Component Palette** section to below the **Greet Invoke** activity.



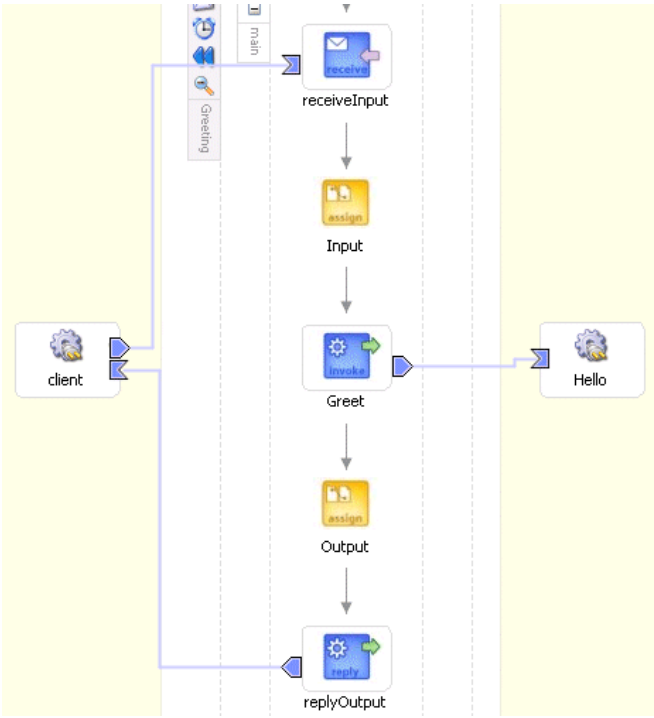
2. Double-click the **assign** icon to display the Assign window.
3. Click the **General** tab.

4. Enter **Output** in the **Name** field.
5. Click **Apply**.
6. Click the **Copy Rules** tab.
7. Click **Create** to display the Create Copy Rule window.
8. Enter the following values:

Field	Value
From	
▪ **Type**	**Variable**
▪ **Variables**	Expand and select **Variable**, then **Greet_Hello_OutputVariable**, then **OutputParameters**, then **ns2:OutputParameters**, and then **GREETING**.
To	
▪ **Type**	**Variable**
▪ **Variables**	Expand and select **Variables**, then **outputVariable**, then **payload**, then **client:GreetingProcessResponse**, and then **client:result**.

9. Click **OK** to close the Create Copy Rule window.
10. Click **OK** to close the Assign window.

    The **Greeting** process appears as follows in JDeveloper BPEL Designer.



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

Validating, Compiling, and Deploying the Greeting Process

1. Go to the **Applications Navigator** section.
2. Right-click **Greeting**.

3. Select **Deploy**, then **LocalBPELServer**, and then **Deploy to default domain**.
4. Enter the domain password (initially set to **bpel**) when prompted.
5. Click **OK**.

   This compiles the BPEL process. Review the bottom of the window for any errors. If there are no errors, deployment was successful.

Running the Greeting Process

1. Log in to Oracle BPEL Console using Internet Explorer by selecting **Start**, then **All Programs**, then **Oracle - *Oracle_Home***, then **Oracle BPEL Process Manager 10.1.2**, and then **BPEL Console,** or by running the `$ORACLE_HOME/integration/orabpel/bin/startorabpel.sh` script for UNIX.
2. Enter the password (initially set to **bpel**) when prompted.

   The **Dashboard** tab of Oracle BPEL Console appears. Note that your BPEL process, **Greeting**, now appears in the **Deployed BPEL Processes** list.
3. Click **Greeting**.

   The Testing this BPEL Process page appears with the **Initiate** tab selected.
4. Enter your first name in the **input** field (for example, **John**).
5. Click **Post XML Message**.

   After the procedure executes and the BPEL process finishes the value appears as follows:

```
Value:  <GreetingProcessResponse>
          <result>Hello John<result>
        </GreetingProcessResponse>
```

6. Click **Audit Instance**.

   The **Instances** tab of Oracle BPEL Console appears, along with the sequence of process activities.
7. Click **More...** on the **Greet** activity to see the input to and output from the stored procedure.

   Note the **<NAME>** tag and its value in the **<InputParameters>** element. This value came from the **inputVariable** and was set by the **Input Assign** activity.

   Note the **<GREETING>** tag and its value in the **<OutputParameters>** element. This value came from the output parameter of the stored procedure. The value was then assigned to the outputVariable by the Output Assign activity.

8. Click the **Flow** tab to view the process flow.

   The process diagram appears.

9. Click any of the activities to view the XML as it passed through the BPEL process. For example, click the **Greet Invoke** activity to display the following:

```
Invoked 2-way operation "Hello" on partner "Hello".
<messages>
<Greet_Hello_InputVariable>
<part xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    name="InputParameters">
<InputParameters
    xmlns="http://xmlns.oracle.com/pcbpel/adapter/db/SCOTT/HELLO/">
<NAME xmlns="">John</NAME>
  </InputParameters>
  </part>
  </Greet_Hello_InputVariable>
<Greet_Hello_OutputVariable>
<part xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    name="OutputParameters">
<db:OutputParameters
    xmlns:db="http://xmlns.oracle.com/pcbpel/adapter/db/SCOTT/HELLO/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<GREETING>Hello John</GREETING>
  </db:OutputParameters>
  </part>
<part xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    name="response-headers">null</part>
  </Greet_Hello_OutputVariable>
  </messages>
```

Summary

This chapter describes how to use the database adapter to communicate with Oracle and third-party databases. The Adapter Configuration Wizard and advanced

configuration capabilities are explained, as is support for stored procedures and functions. A use case is also provided that describes how to create and configure a stored procedure in JDeveloper BPEL Designer.

5

Oracle Application Server Adapter for Java Message Service

This chapter describes how to use the Oracle Application Server Adapter for Java Message Service (JMS adapter), which enables a BPEL process to interact with JMS.

This chapter contains the following topics:

- Introduction to the JMS Adapter
- JMS Adapter Features
- Use Cases for the JMS Adapter
- Summary

Introduction to the 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. For JMS adapter example files, go to

`Oracle_Home\integration\orabpel\samples\tutorials\123.JmsAdapter`

JMS Adapter Features

The JMS adapter includes the following features:

- Based on JMS version 1.0.2b
- Is a generic JMS adapter and can work with any JMS provider. It has been certified against OC4J JMS, Oracle JMS (OJMS based on advanced queuing (AQ)), TIBCO JMS, and IBM Websphere MQSeries. (JMS providers OJMS 8.1.7, 9.0.1.4, and 9.2, and IBM MQSeries JMS 5.2 and 5.3).
- Supports JMS topics and queues
- Supports byte and text message types only in this release. The Adapter Configuration Wizard provides the Native Format Builder Wizard for consuming native data payloads at run time. The Native Format Builder Wizard creates XSD definitions for the underlying native data.
- Supports JMS headers and properties
- 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.

- Supports specifying a durable JMS subscriber
- Supports persistent and nonpersistent modes of a JMS publisher
- 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

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, the messages it received within the transaction are returned to the messaging system and the messages it sent are discarded. The 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.

See Oracle Application Server Adapter Concepts for information on JMS adapter architecture, adapter integration with Oracle BPEL Process Manager, and adapter deployments.

Use Cases for the JMS Adapter

The following use cases demonstrate the procedure for configuring a JMS adapter and examine the resulting WSDL files and associated `oc4j-ra.xml` files.

Concepts

Messaging is any mechanism that allows 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, 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: OC4J 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).

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 they are active. However, JMS API supports durable subscriptions that ensure that consumers receive messages that were published even when they 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.

The JMS API supports both synchronous or 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, optional properties element, and 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.

JMS Header Properties

Table 5–1 describes the JMS header properties.

Table 5–1 JMS Header Properties

Property Name	Description
`JMSDestination`	The destination to which the message is sent, and is set by the JMS producer
`JMSDeliveryMode`	Set to persistent or nonpersistent mode by the JMS consumer
`JMSExpiration`	Duration of the message before the expiration is set by the consumer
`JMSPriority`	Number between `0` and `9` set by the consumer. Larger numbers represent a higher priority.
`JMSMessageID`	Unique message identifier set by the consumer
`JMSTimestamp`	Time stamp when the message was sent to the JMS provider for forwarding
`JMSCorrelationId`	Set by both producers and consumers for linking the response message with the request message. This is an optional attribute.

Table 5–1 (Cont.) JMS Header Properties

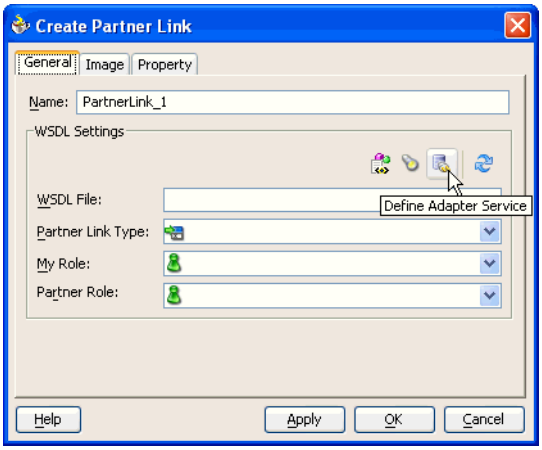
Property Name	Description
`JMSReplyTo`	Optional attribute indicating the destination to which to send a message reply. Can be set by the producer and consumer.
`JMSType`	JMS message type
`JMSRedelivered`	Set by the JMS provider to indicate that the provider has tried to send this message once before to the consumer and has failed

Using the Adapter Configuration Wizard to Configure a JMS Adapter

This section describes how to create an adapter service for a partner link.

1. Click **Define Adapter Service** (third icon) in the Create Partner Link window, as shown in Figure 5–1:

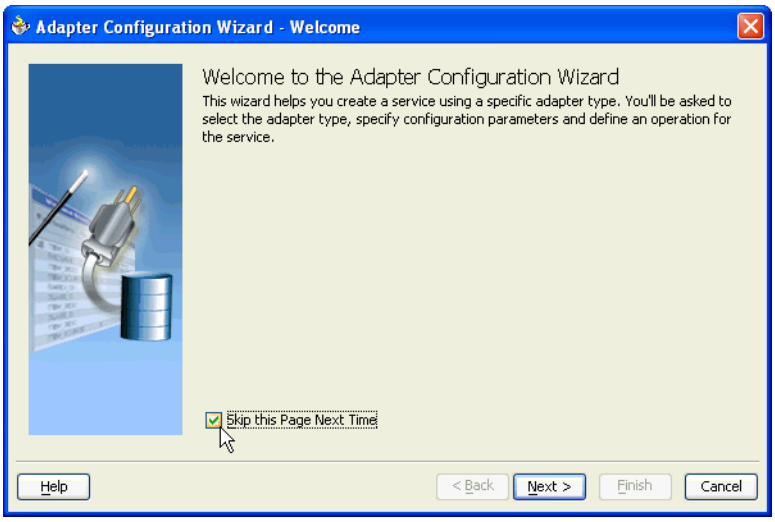
Figure 5–1 Create Partner Link Window



   The Adapter Configuration Wizard - Welcome window appears.

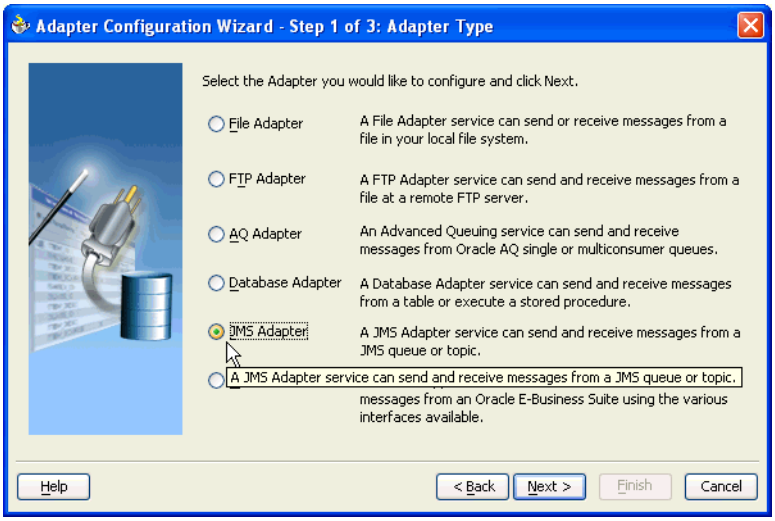
2. If you do not want to see this window each time you use the Adapter Configuration Wizard, select the **Skip this Page Next Time** check box. Then click **Next**.

Figure 5–2 Welcome Window for the Adapter Configuration Wizard



3. Select **JMS Adapter** from the list of available adapter types and click **Next**.

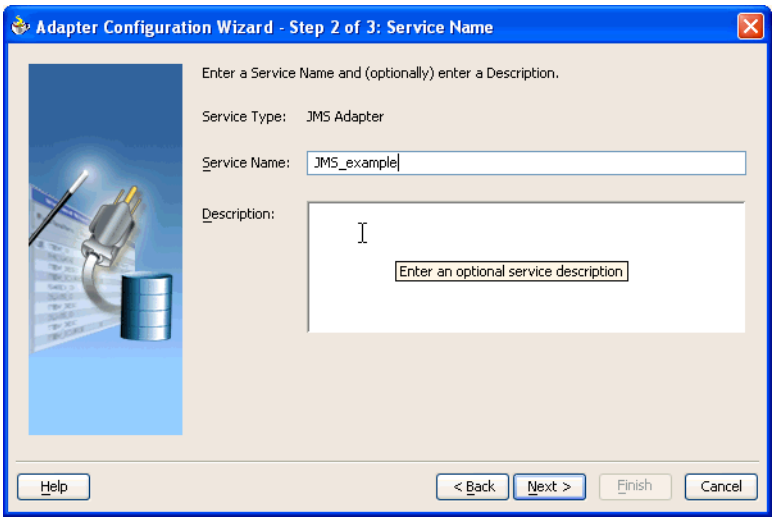
Figure 5–3 Adapter Type Window



The Service Name window appears.

4. Enter a name for the service. You may also add an optional description. Click **Next**.

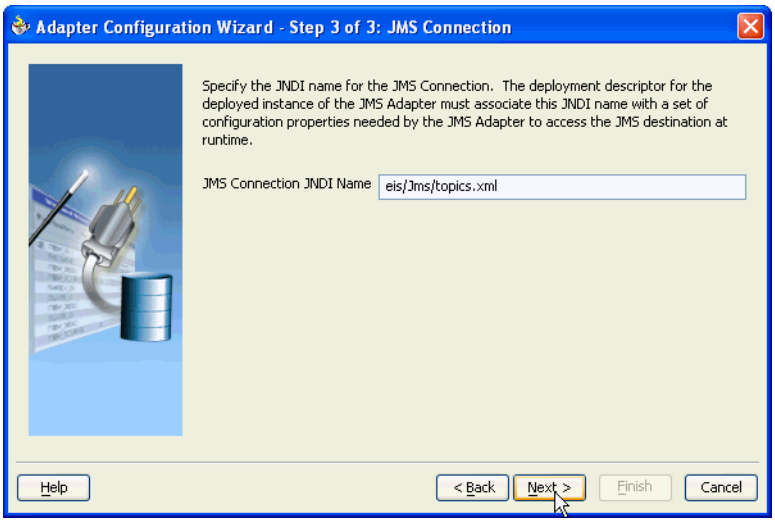
Figure 5–4 Service Name Window



The JMS Connection window appears.

5. Enter the JMS connection JNDI name and click **Next**.

Figure 5–5 JMS Connection Window



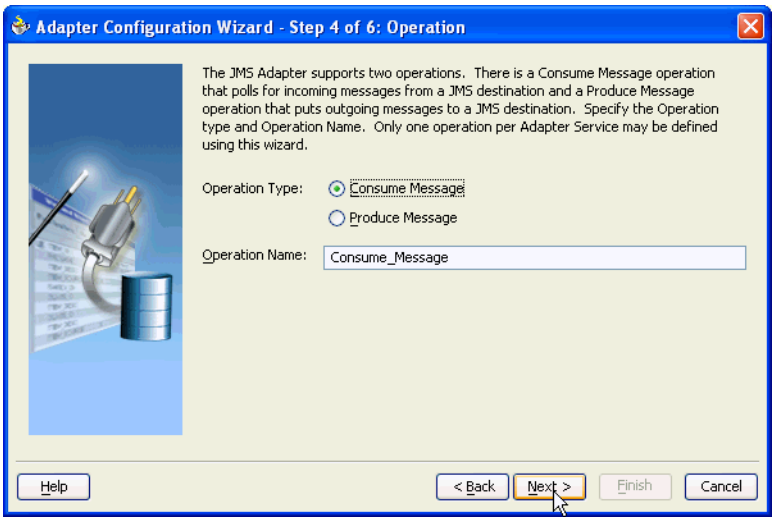
Note: The JNDI name is key in determining the JMS type. The name must match an endpoint in the `oc4j-ra.xml` file, which defines the configuration properties.

See "oc4j-ra.xml file" on page 5-12 for more information.

The Operation window appears.

6. Select **Consume Message** or **Produce Message**. The operation name is filled in automatically. Click **Next**.

Figure 5–6 Operation Window



In this case, **Consume Message** was selected. This enables the adapter to consume (receive) inbound messages from a JMS destination. The Consume Operation Parameters window appears.

7. Enter values for the following fields:

   - **Destination Name**

     The JNDI name of the JMS queue or topic from which to receive the message. The name to enter is based on the type of JMS provider you are using. See the following sections for details:

     – "Configuring for OJMS" on page 5-13

     – "Configuring for OC4J JMS" on page 5-15

     – "Configuring for TIBCO JMS" on page 5-16

     – "Configuring for IBM Websphere JMS" on page 5-17

   - **Message Body Type**

     Select either TextMessage or BytesMessage.

     The **StreamMessage** and **MapMessage** message types are 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. Normally a subscriber loses messages if it becomes disconnected, but a durable subscriber downloads stored messages when it reconnects.

   - **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')

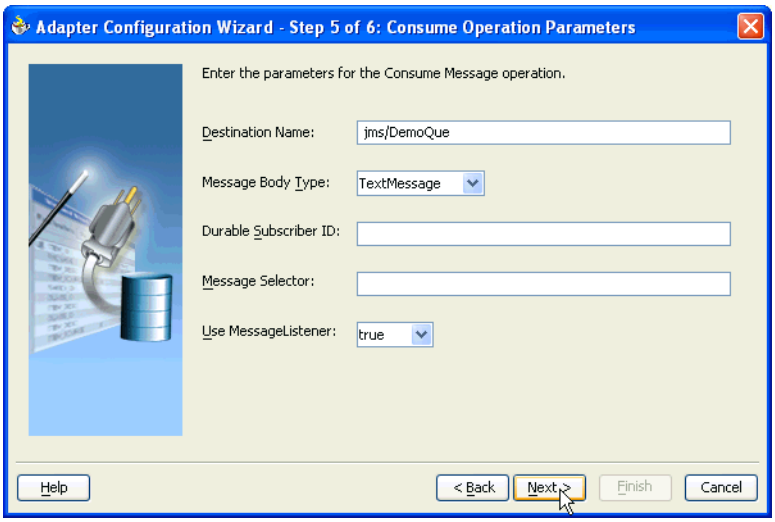
- **Use MessageListener**

  This field is set to **true** by default, which means that the server does an asynchronous callback to the adapter. If this is set to **false**, the adapter performs a synchronous blocking receive.

Note: This example shows a consume message operation. For a produce message operation, this window is different. See "Produce Message Procedure" on page 5-12 to see how this part of the procedure differs.

After you enter the appropriate parameters, click **Next**.

Figure 5–7 Consume Operation Parameters Window



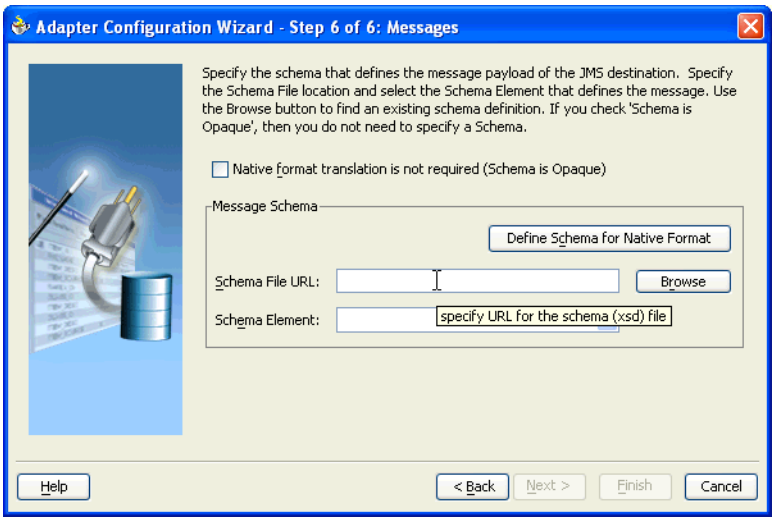
8. The **Messages** window appears. These settings define the correct schema for the message payload.

   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. Click the **Browse** button.

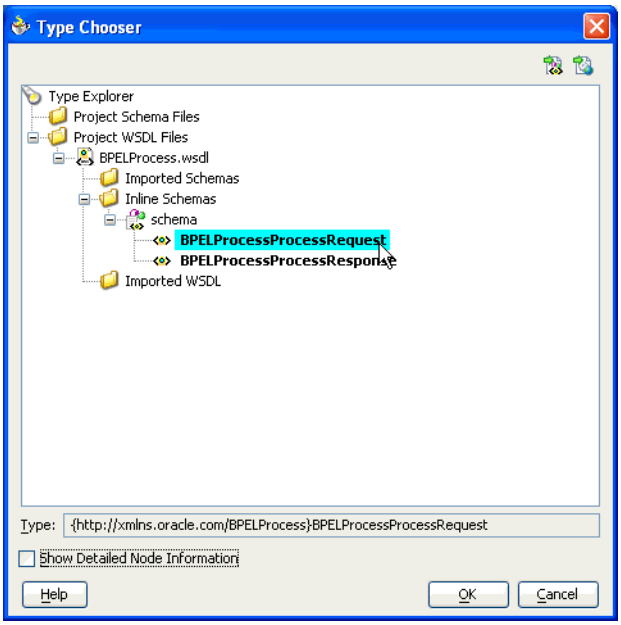
Figure 5–8 Messages Window



The **Type Chooser** window appears, with the **Type Explorer** navigation tree.

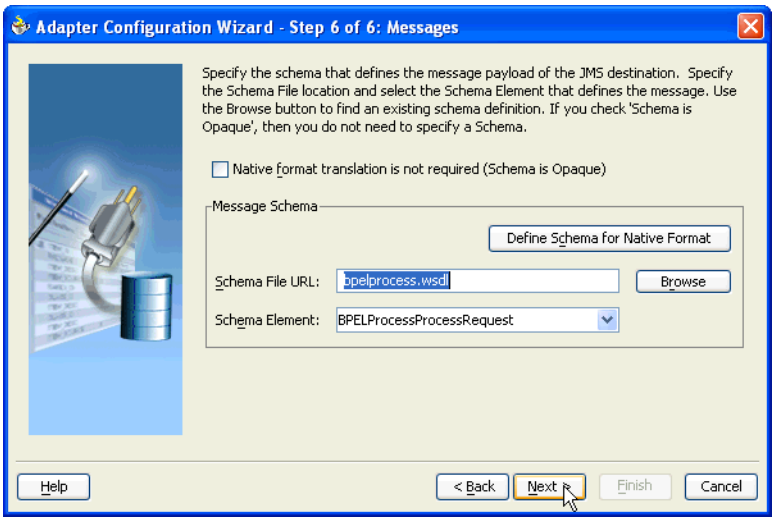
9. Browse the tree and select the appropriate schema type. Then click **OK**.

Figure 5–9 Selecting a Schema from the Type Chooser Window



The Messages window appears again, this time with the **Schema File URL** field and the **Schema Element** field filled in.

Figure 5–10 Completed Messages Window



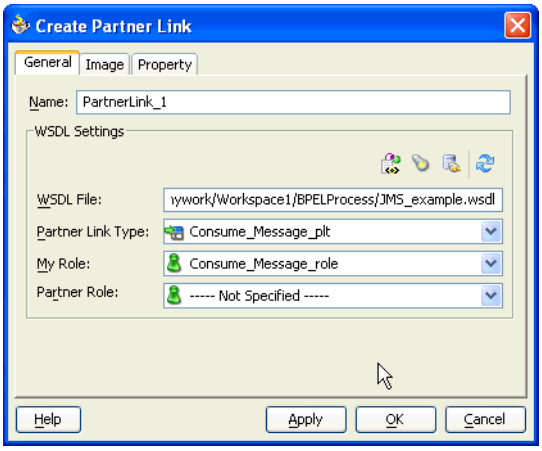
10. Click **Next**.

    The **Finish** window appears. This box shows the path and name of the adapter file that the wizard creates.

11. Click **Finish**.

    The Create Partner Link window appears with the fields populated.

Figure 5–11 Completed Create Partner Link Window



12. Click **OK**.

Generated WSDL File

The following WSDL file is generated by the Adapter Configuration Wizard:

```
<definitions
     name="JMS_Example"
     targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/jms/JMS_Example/"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/jms/JMS_Example/"
     xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
     xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
```

```
 xmlns:opaque="http://xmlns.oracle.com/pcbpel/adapter/opaque/"
 xmlns:pc="http://xmlns.oracle.com/pcbpel/"
 xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/jms/"
>
```

This 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/"
location="jmsAdapterInboundHeader.wsdl"/>
```

This code segment imports the necessary namespace.

```
   <types>
   <schema targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/opaque/"
           xmlns="http://www.w3.org/2001/XMLSchema" >
     <element name="opaqueElement" type="base64Binary" />
   </schema>
   </types>
   <message name="Consume_Message_msg">
       <part name="opaque" element="opaque:opaqueElement"/>
   </message>
   <portType name="Consume_Message_ptt">
       <operation name="Consume_Message">
           <input message="tns:Consume_Message_msg"/>
       </operation>
   </portType>
```

This code segment defines the message type, name, and the port type for the partner link.

```
   <binding name="Consume_Message_binding" type="tns:Consume_Message_ptt">
   <pc:inbound_binding />
       <operation name="Consume_Message">
     <jca:operation
         ActivationSpec="oracle.tip.adapter.jms.inbound.JmsConsumeActivationSpec"
         DestinationName="jms/DemoQue"
         UseMessageListener="true"
         PayloadType="TextMessage"
         OpaqueSchema="true" >
     </jca:operation>
     <input>
       <jca:header message="hdr:InboundHeader_msg" part="inboundHeader"/>
     </input>
       </operation>
   </binding>
```

This code segment defines the necessary bindings for the consume message operation, the target queue, and identifies the message header.

```
   <service name="JMS_Example2">
       <port name="Consume_Message_pt" binding="tns:Consume_Message_binding">
     <jca:address location="eis/Jms/topics.xml" />
       </port>
   </service>
  <plt:partnerLinkType name="Consume_Message_plt" >
    <plt:role name="Consume_Message_role" >
      <plt:portType name="tns:Consume_Message_ptt" />
    </plt:role>
  </plt:partnerLinkType>
</definitions>
```

This last part defines the database connection, the connection factory (as defined in the `oc4j-ra.xml` file), and the name and role of the `partnerLinkType` and `portType`.

oc4j-ra.xml file

The `oc4j-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 `oc4j-ra.xml` file:

```
<?xml version="1.0"?>
<!DOCTYPE oc4j-connector-factories PUBLIC "-//Oracle//DTD Oracle Connector
 9.04//EN" "http://xmlns.oracle.com/ias/dtds/oc4j-connector-factories-9_04.dtd">
<oc4j-connector-factories>
     <connector-factory location="eis/MyJmsTopic1" connector-name="Jms Adapter">
     <config-property name="connectionFactoryLocation"
                     value="jms/TopicConnectionFactory"/>
     <config-property name="factoryProperties" value=""/>
     <config-property name="acknowledgeMode" value="AUTO_ACKNOWLEDGE"/>
     <config-property name="isTopic" value="true"/>
     <config-property name="isTransacted" value="true"/>
     <config-property name="username" value="admin"/>
     <config-property name="password" value="welcome"/>
</connector-factory>
<connector-factory location="eis/MyJmsTopic2" connector-name="Jms Adapter">
     <config-property name="connectionFactoryLocation"

...
</oc4j-connector-factories>
```

Produce Message Procedure

A produce message operation has certain differences in the definition procedure, particularly in Step 7 on page 5-7 of "Using the Adapter Configuration Wizard to Configure a JMS Adapter". 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 window is shown in Figure 5–12.

- **Destination Name:**

  The JNDI name of the JMS queue or topic to which to deliver the message. The name to enter is based on the type of JMS provider you are using. See the following sections for detail:

  - "Configuring for OJMS" on page 5-13
  - "Configuring for OC4J JMS" on page 5-15
  - "Configuring for TIBCO JMS" on page 5-16
  - "Configuring for IBM Websphere JMS" on page 5-17

- **Message Body Type:**

  The supported values are **TextMessage** or **BytesMessage**. The **StreamMessage** and **MapMessage** message types are 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 7 on page 5-7 of "Using the Adapter Configuration Wizard to Configure a JMS Adapter". 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 times to receive all the messages.

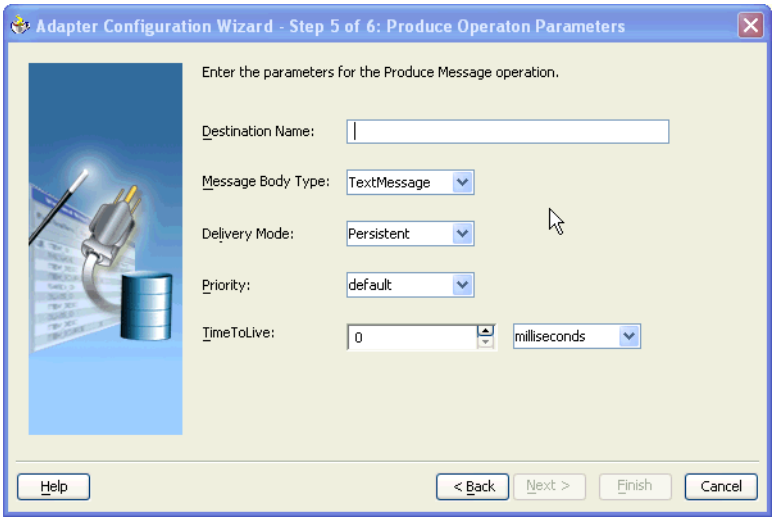
- **Priority:**

  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.

Figure 5–12 Produce Operation Parameters Window



Configuring for OJMS

Configure the OJMS provider within the `resource-provider` element in the global `application.xml` file. You can configure the resource provider with a URL property. The following demonstrates a URL configuration:

```
<resource-provider class="oracle.jms.OjmsContext" name="ojmsdemo">
     <description>OJMS/AQ</description>
     <property name="url" value="jdbc:oracle:thin:@localhost:1521:my" />
     <property name="username" value="jmsuser" />
     <property name="password" value="jmsuser" />
</resource-provider>
```

In the `oc4j-ra.xml` file, add the following code segments:

```
<connector-factory location="eis/aqjms/Topic" connector-name="Jms Adapter">
     <config-property name="connectionFactoryLocation"
          value="java:comp/resource/ojmsdemo/TopicConnectionFactories/myTCF" />
     <config-property name="factoryProperties" value="" />
     <config-property name="acknowledgeMode" value="AUTO_ACKNOWLEDGE" />
```

```
     <config-property name="isTopic" value="true" />
     <config-property name="isTransacted" value="true" />
     <config-property name="username" value="jmsuser" />
     <config-property name="password" value="jmsuser" />
</connector-factory>
<connector-factory location="eis/aqjms/Queue" connector-name="Jms Adapter">
    <config-property name="connectionFactoryLocation" value="
         java:comp/resource/ojmsdemo/QueueConnectionFactories/myQCF" />
    <config-property name="factoryProperties" value="" />
    <config-property name="acknowledgeMode" value="AUTO_ACKNOWLEDGE" />
    <config-property name="isTopic" value="false" />
    <config-property name="isTransacted" value="true" />
    <config-property name="username" value="jmsuser" />
    <config-property name="password" value="jmsuser" />
</connector-factory>
```

In this case, correct JMS connection JNDI names for Step 5 on page 5-6 of "Using the Adapter Configuration Wizard to Configure a JMS Adapter" are `eis/aqjms/Topic` or `eis/aqjms/Queue`.

Set the `isTransacted` value in the `oc4j-ra.xml` file to `true`. Setting it to `false` leads to exception errors.

Access the OJMS Resources

The OJMS syntax for the connection factory is as follows:

```
java:comp/resource + JMS_provider_name + TopicConnectionFactories + user_defined_
name
```

or

```
java:comp/resource + JMS_provider_name + QueueConnectionFactories + user_defined_
name
```

The *user_defined_name* can be anything and does not match any other configuration. The `ConnectionFactories` details what type of factory is being defined. For this example, the JMS provider name is defined in the `resource-provider` element as `ojmsdemo`.

For a queue connection factory: Because the JMS provider name is `ojmsdemo` and you decide to use a name of `myQCF`, the connection factory name is `java:comp/resource/ojmsdemo/QueueConnectionFactories/myQCF`.

For a topic connection factory: Because the JMS provider name is `ojmsdemo` and you decide to use a name of `myTCF`, the connection factory name is `java:comp/resource/ojmsdemo/TopicConnectionFactories/myTCF`.

The user-defined names of `myQCF` and `myTCF` are not used for anything else in your logic. Therefore, you can choose any name.

Destination

The OJMS syntax for any destination is as follows:

```
java:comp/resource + JMS_provider_name + Topics + Destination_name
```

or

```
java:comp/resource + JMS_provider_name + Queues + Destination_name
```

The topic or queue details which type of destination is being defined. The destination name is the actual queue or topic name defined in the database.

For this example, the JMS provider name is defined in the `resource-provider` element as `ojmsdemo`. In the database, the queue name is `aqQueue`.

For a queue: If the JMS provider name is `ojmsdemo` and the queue name is `aqQueue`, the JNDI name for the queue is `java:comp/resource/ojmsdemo/Queues/aqQueue`.

For a topic: If the JMS provider name is `ojmsdemo` and the topic name is `aqTopic`, the JNDI name for the topic is `java:comp/resource/ojmsdemo/Topics/aqTopic`.

OJMS and Remote Databases

To configure the adapter to use a remote database, the entries in the `application.xml` file must look as follows:

```
<resource-provider class="oracle.jms.OjmsContext" name="ojmsdemo">
<description>OJMS/AQ</description>
<property name="url"
value="jdbc:oracle:thin:@remote-host:remote-port:remote-sid" />
<property name="username" value="jmsuser" />
@ <property name="password" value="jmsuser" />
</resource-provider>
```

Configuring for OC4J JMS

If the OC4J JMS server is running on another remote host, you can configure the JMS adapter to talk to the server by using the following connector entry. Note that the only difference with this connector entry is in the factory properties. The factory properties can establish a JNDI context for the adapter. Substitute `[hostname]` with the hostname of the server on which the OC4J JMS server is running. If the RMI port of the remote OC4J instance is not the default value (`23791`), you must specify the RMI port in the provider URL (that is, `ormi://remotehost.domain.com:23795`).

```
<connector-factory location="eis/RemoteOC4JJMS/Queue" connector-name="Jms
Adapter">
     <config-property name="connectionFactoryLocation"
           value="jms/QueueConnectionFactory" />
     <config-property name="factoryProperties"
value="java.naming.factory.initial=com.evermind.server.ApplicationClientInitialCon
textFactory;java.naming.provider.url=ormi://[hostname];
java.naming.security.principal=admin;java.naming.security.credentials=welcome" />
     <config-property name="acknowledgeMode" value="AUTO_ACKNOWLEDGE" />
     <config-property name="isTopic" value="false" />
     <config-property name="isTransacted" value="true" />
     <config-property name="username" value="admin" />
     <config-property name="password" value="welcome" />
</connector-factory>
```

In this case, the correct JMS connection JNDI name for Step 5 on page 5-6 of "Using the Adapter Configuration Wizard to Configure a JMS Adapter" is `eis/RemoteOC4JJMS/Queue`.

In addition, you must have the file `META-INF/application-client.xml` in the classpath. The contents of the file can be the following:

```
<application-client/>
```

To put this file in the classpath, put the above contents into the file at *`Oracle_Home`*`\integration\orabpel\system\classes\META-INF\application-client.xml` and restart Oracle BPEL Server.

If you set `isTransacted` to `true` in the `oc4j-ra.xml` file for an outbound connection, you receive an error. Do not set this value to `true` for outbound connections.

Destination Name

The destination name for OC4J JMS is either a JNDI location (for example, **jms/demoQueue** or **jms/demoTopic**) or the actual name of the destination as configured in `jms.xml` (for example, **Demo Queue** or **Demo Topic**).

Configuring for TIBCO JMS

The BPEL OC4J `application.xml` file should contain the following jar file, where Tibco EMS is installed in `C:\tibco\ems`. The JMS and JNDI jar files are already present in the classpath and do not need to be included.

```
<library path="C:\tibco\ems\clients\java\tibjms.jar" />
```

Create the appropriate endpoints for the JMS connection factories in `oc4j-ra.xml`. After this change is made, restart Oracle BPEL Server. Here are the appropriate code segments. You can modify the necessary parameters and use this for your purpose:

```
<connector-factory location="eis/tibjms/Topic" connector-name="Jms Adapter">
     <config-property name="connectionFactoryLocation"
          value="TopicConnectionFactory" />
     <config-property name="factoryProperties"
value="java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContextFac
tory;java.naming.provider.url=tibjmsnaming://localhost:7222;java.naming.security.p
rincipal=admin;java.naming.security.credentials=password" />
     <config-property name="acknowledgeMode" value="AUTO_ACKNOWLEDGE" />
     <config-property name="isTopic" value="true" />
     <config-property name="isTransacted" value="true" />
     <config-property name="username" value="admin" />
     <config-property name="password" value="password" />
</connector-factory>
<connector-factory location="eis/tibjms/Queue" connector-name="Jms Adapter">
     <config-property name="connectionFactoryLocation"
          value="QueueConnectionFactory" />
     <config-property name="factoryProperties"
value="java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContextFac
tory;java.naming.provider.url=tibjmsnaming://localhost:7222;java.naming.security.p
rincipal=admin;java.naming.security.credentials=password" />
     <config-property name="acknowledgeMode" value="AUTO_ACKNOWLEDGE" />
     <config-property name="isTopic" value="false" />
     <config-property name="isTransacted" value="true" />
     <config-property name="username" value="admin" />
     <config-property name="password" value="password" />
</connector-factory>
```

In this case, correct JMS connection JNDI names for Step 5 on page 5-6 of "Using the Adapter Configuration Wizard to Configure a JMS Adapter" are `eis/tibjms/Topic` or `eis/tibjms/Queue`.

When using Tibco JMS, always set the `ClientID` property as follows in the *`Oracle_Home`*`\integration\orabpel\system\appserver\oc4j\j2ee\home\application-deployments\default\FtpAdapter\oc4j-ra.xml` file:

```
<config-property name="factoryProperties"
    value="ClientID=clientId{time}"/>
```

The substring {time} instructs the run time to replace it with the value of Java System.currentTimeMillis(). The other supported substitutions are:

- {checksum}

  A checksum based on the values of the oc4j-ra.xml connection factory properties (referenced through jca:address).

- {sequence}

  Next member of an increasing series of integers starting at zero.

These settings enable you to specify a fixed or a variable ClientID in oc4j-ra.xml.

Direct Connection

A direct connection can also be defined instead of the JNDI connection. A direct connection is necessary for the Solaris middle tier. Use the following direct connection entry in the oc4j-ra.xml file, instead of the JNDI entry.

```
<connector-factory location="eis/tibjmsDirect/Topic" connector-name="Jms
Adapter">
     <config-property name="connectionFactoryLocation"
                      value="com.tibco.tibjms.TibjmsTopicConnectionFactory"/>
     <config-property name="factoryProperties"
  @ value="ServerUrl=tcp://152.69.159.188:7222;UserName=admin;UserPassword=passwor
d"/>
     <config-property name="acknowledgeMode" value="AUTO_ACKNOWLEDGE"/>
     <config-property name="isTopic" value="true"/>
     <config-property name="isTransacted" value="true"/>
     <config-property name="username" value="admin"/>
     @ <config-property name="password" value="password"/>
</connector-factory>
```

Destination Name

The destination name is the name of the topic or queue as listed in the Tibco JMS server. For example, the sample queue is called queue.sample while the sample topic is called topic.sample.

When you need to distinguish between a queue or topic, you can prefix the name with either $topics: or $queues: depending on the type of destination. For example, the destination name $queues:common refers to a queue that is distinct from the topic called $topics:common. Note that you must escape the $ character in the .wsdl file for the partner link because the adapter framework processes the $ as a special character. The escape character is the backslash (\).

Configuring for IBM Websphere JMS

The BPEL OC4J application.xml file should contain the following jar files, assuming MQ Series is installed in the C:\mqseries directory.

```
<library path="C:\mqseries\java\lib\com.ibm.mq.jar" />
```

```
<library path="C:\mqseries\java\lib\com.ibm.mqjms.jar" />
```

Create the appropriate endpoints for the JMS connection factories in oc4j-ra.xml. After this change is made, you must restart Oracle BPEL Server. Here are the appropriate code segments. You can modify the necessary parameters and use this for your purpose.

```
<connector-factory location="eis/mqseries/Queue" connector-name="Jms Adapter">
     <config-property name="connectionFactoryLocation"
```

```
          value="com.ibm.mq.jms.MQQueueConnectionFactory" />
     <config-property name="factoryProperties"
value="QueueManager=my.queue.manager;TransportType=1;HostName=myhost.com;Port=1414
;Channel=MYCHANNEL" />
     <config-property name="acknowledgeMode" value="AUTO_ACKNOWLEDGE" />
     <config-property name="isTopic" value="false" />
     <config-property name="isTransacted" value="true" />
     <config-property name="username" value="MUSR_MQADMIN" />
     <config-property name="password" value="password" />
</connector-factory>
```

In this case, the correct JMS connection JNDI name for Step 5 on page 5-6 of "Using the Adapter Configuration Wizard to Configure a JMS Adapter" is `eis/mqseries/Queue`.

Destination Name

The destination name is the name of the topic or queue listed in your MQ Series configuration. For example, the name of the queue can be `queue:///MYQUEUE?targetClient=1`.

Summary

This chapter describes how to use the Adapter Configuration Wizard to configure a JMS adapter, the options available for this adapter, and how to configure adapters for different application servers.

6

Native Format Builder Wizard

This chapter describes the Native Format Builder Wizard, which enables you to create native schemas used for translation. Use cases and constructs for the schema are also provided.

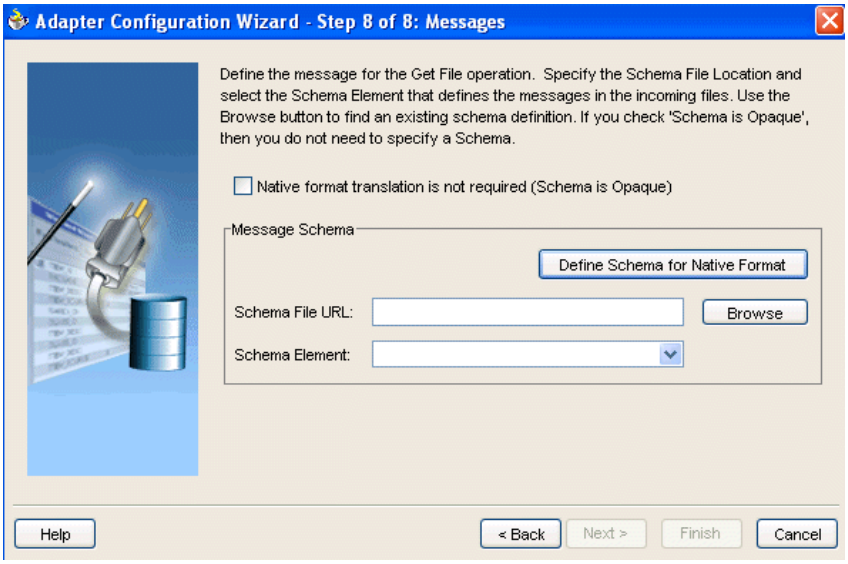
This chapter contains the following topics:

- Creating Native Schema Files with the Native Format Builder Wizard
- Understanding Native Schema
- Native Schema Constructs
- Summary

Creating Native Schema Files with the Native Format Builder Wizard

Oracle BPEL Process Manager requires native schemas for translation, which are based on XML schema. However, not all commonly used formats use XML schema files. To address this situation, Oracle BPEL Process Manager provides the Native Format Builder Wizard. This wizard is accessible from the **Define Schema for Native Format** button of the Messages window of the Adapter Configuration Wizard shown in Figure 6–1. The Messages window is the last window to display in the Adapter Configuration Wizard prior to the Finish window.

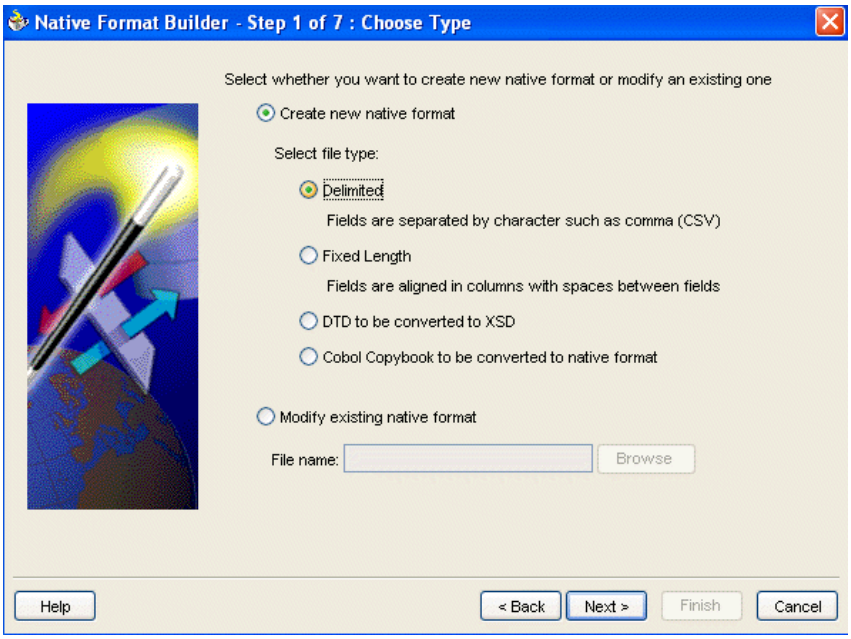
Figure 6–1 Starting the Native Format Builder Wizard



Supported Formats

The Native Format Builder Wizard guides you through the creation of a native schema file from the following formats shown in Figure 6–2. A sample data file format for the selected type must already exist; you cannot create a native schema without one. You can also select to modify an existing native schema previously created with this wizard, except for those generated from a DTD or COBOL Copybook.

Figure 6–2 Native Format Builder Wizard



Delimited (such as CSV files)

This option enables you to create native schemas for records, where the fields are separated by a value such as a comma or pound sign.

Fixed Length (Positional)

This option enables you to create native schemas for records, where the fields are all fixed lengths.

DTD

This option enables you to generate native schema from the user-supplied DTD.

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 copybook such 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 exactly the same data layout as `BuyerName`.

Group items in COBOL can be mapped to elements in XML schema with the `complexType`. Similarly, elementary items can be mapped to elements of type simple type with certain native format annotations to help the run time 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 newline 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 the clause is supported.

- N indicates the clause is not supported.
- I indicates 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 1 byte.
PIC A(*n*)	Y	Y	AA...	Alphabetic – Any letter of the alphabet or space. Each A corresponds to 1 byte.
PIC 9(*n*) DISPLAY	Y	Y	9999...	Any character position that contains a numeral. Each 9 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.
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		
BLANK WHEN ZERO	I	I		Ignored
RENAMES	N	N		This is rarely seen in COBOL Copybooks
INDEX	N	N		4-byte index
SYNCHRONIZED	I	I	SYNC	
POINTER	N	N		
PROCEDURE-POINTER				
FILLER	Y	Y		

The numeric types covered in Table 6–1 are stored as one character per digit. Table 6–2 covers 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.

Table 6–2 (Cont.) COBOL Clauses (Numeric Types Stored More Efficiently)

COBOL Clause	Design Time Support	Runtime Support	Supported Synonyms	Comments
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 4 bytes long.
COMP-2	Y	Y	COMPUTATIONAL-2	Double precision, floating point number that is 8 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 in case custom behavior must be added.
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		
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, a custom data handler for that type must be written.

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 insertion are handled by this method. Floating insertion, zero suppression, and replacement insertion are not supported.

Native Format Builder Wizard Windows

For delimited and fixed-length files, you are guided through windows that prompt you for the following information to create definitions in native schema format:

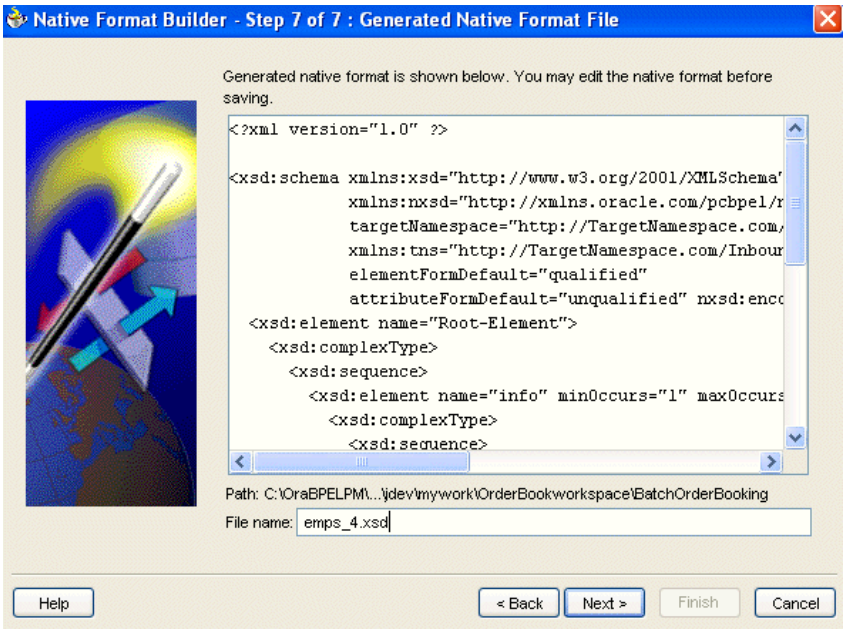
- The data file to sample (from which to create a native schema) and its encoding
- The number of rows to skip and the number of rows to sample in the file
- If the file contains multiple records, are they the same type or different types
- The target namespace, container element name, and record name
- The record delimiter (for example, end-of-line) and field delimiter (for example, comma or pound sign) or field length
- The field properties (such as name, datatype, delimiter, and length)

DTD and COBOL Copybook files already include definitions in their native formats. For these formats, the Native Format Builder Wizard prompts you for the following information to create native schema versions of these definitions.

- The filename of the DTD or COBOL Copybook definition, namespace, and root element
- The character set, byte order, and records delimiter (for COBOL Copybook only)

As you move through the wizard windows, the native schema file being created displays at the bottom. This enables you to watch the native schema file being built. The final window displays the generated native schema for the native format shown in Figure 6–3. You can edit this format before clicking **Next**.

Figure 6–3 Native Schema Generated From Native Format



When you click **Finish**, you are returned to the Messages window of the Adapter Configuration Wizard shown in Figure 6–1 on page 6-1. The **Schema File URL** and **Schema Element** fields are filled in with details about your newly created native schema file. You can now use the Adapter Configuration Wizard to create a WSDL file for the adapter to communicate with your BPEL process.

Understanding Native Schema

This section provides use cases and explains various constructs of native schema to translate the native format data to XML.

This section contains the following topics:

- Use Cases for the Native Format Builder
- Native Schema Constructs

> **Note:** Not all native schemas can be generated from the Native Format Builder Wizard. This wizard can handle only basic scenarios. This section describes the capabilities of the native schema using various examples and use cases.

Use Cases for the Native Format Builder

This section contains the following topics:

- Defining a Comma-Separated Value File Structure
- Defining a * Separated Value File Structure
- Defining a Fixed Length Structure
- Defining a More Complex Structure - Invoice
- COBOL Copybook

Defining a Comma-Separated Value File Structure

A comma-separated value (CSV) file is a common non-XML file structure. A CSV file may or may not have the first few line as headers, in which case, you may want to ignore them.

Native Data Format to Be Translated The following native data format is provided:

```
Name,Street,City,State,Country
Oracle India Private Limited, Lexington Towers, Bangalore, Karnataka, India
Intel India Private Limited, Ring Road, Bangalore, Karnataka, India
```

Native Schema The corresponding native schema definition can be defined as follows:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            targetNamespace="http://www.oracle.com/ias/processconnect"
            xmlns:tns="http://www.oracle.com/ias/processconnect"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            nxsd:encoding="US-ASCII"
            nxsd:headerLines="1"
            nxsd:stream="chars"
            nxsd:version="NXSD">

  <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="," >
              </xsd:element>
             <xsd:element name="Street" type="xsd:string" nxsd:style="terminated"
                nxsd:terminatedBy="," >
             </xsd:element>
             <xsd:element name="City" type="xsd:string" nxsd:style="terminated"
                nxsd:terminatedBy="," >
             </xsd:element>
             <xsd:element name="State" type="xsd:string" nxsd:style="terminated"
                nxsd:terminatedBy="," >
             </xsd:element>
             <xsd:element name="Country" type="xsd:string" nxsd:style="terminated"
                nxsd:terminatedBy="${eol}" >
             </xsd:element>
           </xsd:sequence>
         </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

The `nxsd:headerLines="1"` in the schema above, at the `xsd:schema` construct, means to skip one line in the native data before actually translating the rest of the data. This is because the first line is a header line. If set, `nxsd:stream="chars"` means the data is to be read as characters. If set, `nxsd:stream="bytes"` means to read the native data as bytes. For each of the element declarations, `Name`, `Street`, `City`,

State, Country, which have a corresponding scalar data, the nxsd:style="terminated" 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" on page 6-24 for details on the terminated style.

Translated XML Using the Native Schema The native data using the corresponding native schema format is translated to the following XML:

```
<AddressBook xmlns="http://www.oracle.com/ias/processconnect">
  <Address>
    <Name>Oracle India Private Limited</Name>
    <Street>Lexington Towers</Street>
    <City>Bangalore</City>
    <State>Karnataka</State>
    <Country>India</Country>
  </Address>
  <Address>
    <Name>Intel India Private Limited</Name>
    <Street>Ring Road</Street>
    <City>Bangalore</City>
    <State>Karnataka</State>
    <Country>India</Country>
  </Address>
</AddressBook>
```

Defining a * Separated Value File Structure

The use case defined above 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 + separated value).

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 in lieu of nxsd:terminatedBy="," you now define the terminated by format as nxsd:terminatedBy="+". See "Defining Terminated Data" on page 6-24 for details on the terminated style.

Defining a Fixed Length Structure

In this example, the native data used is the same as in the CSV case. The only difference is that here the data is fixed length, and not CSV.

Native Data Format to Be Translated The following native data format is provided:

```
Name                          Street            City      State     Country
Oracle India Private Limited  Lexington Towers  Bangalore Karnataka India
Intel India Private Limited   Outer Ring Road   Bangalore Karnataka India
```

Native Schema The corresponding native schema definition is similar to the definition of the CSV, but the style now changes from nxsd:style="terminated" to nxsd:style="fixedLength" along with the relevant attributes for the fixed length style. For the style fixed length, the one mandatory attribute is the length: nxsd:length. The value of nxsd:length is the actual length of the data to be read.

The complete definition for fixed length style for the native data above can be defined as follows:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            targetNamespace="http://www.oracle.com/ias/processconnect"
            xmlns:tns="http://www.oracle.com/ias/processconnect"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            nxsd:encoding="US-ASCII"
            nxsd:headerLines="1"
            nxsd:stream="chars"
            nxsd:version="NXSD">

  <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="fixedLength"
                nxsd:length="31">
             </xsd:element>
             <xsd:element name="Street" type="xsd:string" nxsd:style="fixedLength"
                nxsd:length="19">
             </xsd:element>
             <xsd:element name="City" type="xsd:string" nxsd:style="fixedLength"
                nxsd:length="10">
             </xsd:element>
             <xsd:element name="State" type="xsd:string" nxsd:style="fixedLength"
               nxsd:length="10">
             </xsd:element>
             <xsd:element name="Country" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="${eol}">
             </xsd:element>
           </xsd:sequence>
         </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

See "Defining Fixed Length Data" on page 6-21 for details on the fixed length style.

Defining a More Complex Structure - Invoice

An invoice is a more complex structure than the structure in the previous CSV, *SV, and fixed length use cases. An invoice usually contains buyer information, seller information, and line items.

Native Data Format to Be Translated The following native data format for an invoice is provided:

```
6335722^Company One^First Street   999       San Jose       95129USCA650-801-6250
       ^Oracle^Bridge 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
```

The first line in the native data is purchaser details, followed by seller details, followed by line items, and finally the total for the line items. Both purchaser and seller have the same structure:

- The first 7 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

This address contains a fixed length street, city, and so on. The last line item ends with the sharp symbol "#", followed by the line-item total.

Native Schema The native schema definition corresponding to the preceding native data can be defined as follows:

```
<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>
```

Translated XML Using the Native Schema The translated XML looks as follows:

```
<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>
```

COBOL Copybook

A demonstration is provided that shows how the file adapter and FTP adapter process a file in COBOL Copybook format (through use of the Native Format Builder Wizard) to create a native schema file for translation. For a demonstration that uses the Native Format Builder Wizard to convert a COBOL Copybook file to a native schema file, go to

Oracle_Home\integration\orabpel\samples\tutorials\121.FileAdapter\CobolCopyBook

The following COBOL Copybook examples are also 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, each such group implicitly redefines the other.

```
01  PAYROLL-E-RECORD.
    05  PAYROLL-E-EMPLOYEE-NUMBER     PIC X(10).
    05  PAYROLL-E-TRANS-CODE          PIC X(02).
    05  PAYROLL-E-NAME                PIC X(08).
    05  FILLER                        PIC X(25).

01  PAYROLL-F-RECORD.
    05  PAYROLL-F-EMPLOYEE-NUMBER     PIC X(10).
    05  PAYROLL-F-TRANS-CODE          PIC X(02).
    05  PAYROLL-F-IDENTIFIER-VALUE    PIC X(03).
    05  PAYROLL-F-NAME                PIC X(30).


01  PAYROLL-H-RECORD.
    05  PAYROLL-H-EMPLOYEE-NUMBER    PIC X(10).
    05  PAYROLL-H-TRANS-CODE         PIC X(02).
    05  PAYROLL-H-HED-NUMBER         PIC 9(03).
    05  FILLER                       PIC X(30).
```

The generated schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--Native format was generated from COBOL copybook :
```

```
D:\work\jDevProjects\CCB\Copybooks\payroll.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/implicitRedefines"
       xmlns:tns="http://TargetNamespace.com/ccb/implicitRedefines"
       elementFormDefault="qualified" attributeFormDefault="unqualified"
       nxsd:version="NXSD" nxsd:encoding="cp037" nxsd:byteOrder="bigEndian"
       nxsd:stream="chars">
 <xsd:element name="Payroll-Records">
  <xsd:complexType>
   <!--Please add values for nxsd:lookAhead attributes for the elements in the
         choice model group.-->
  <xsd:choice minOccurs="1" maxOccurs="unbounded">
   <!--COBOL declaration : 01 PAYROLL-E-RECORD -->
   <xsd:element name="PAYROLL-E-RECORD" nxsd:lookAhead="" nxsd:lookFor="">
    <xsd:complexType>
     <xsd:sequence>
      <!--COBOL declaration : 05 PAYROLL-E-EMPLOYEE-NUMBER PIC X(10)-->
      <xsd:element name="PAYROLL-E-EMPLOYEE-NUMBER" type="xsd:string"
              nxsd:style="fixedLength" nxsd:padStyle="tail"
              nxsd:paddedBy=" " nxsd:length="10"/>
      <!--COBOL declaration : 05 PAYROLL-E-TRANS-CODE PIC X(02)-->
      <xsd:element name="PAYROLL-E-TRANS-CODE" type="xsd:string"
              nxsd:style="fixedLength" nxsd:padStyle="tail"
              nxsd:paddedBy=" " nxsd:length="2"/>
      <!--COBOL declaration : 05 PAYROLL-E-NAME PIC X(08)-->
      <xsd:element name="PAYROLL-E-NAME" type="xsd:string"
              nxsd:style="fixedLength" nxsd:padStyle="tail"
              nxsd:paddedBy=" " nxsd:length="8"/>
      <!--COBOL declaration : 05 FILLER PIC X(25)-->
      <xsd:element name="FILLER" type="xsd:string"
              nxsd:style="fixedLength" nxsd:padStyle="tail"
              nxsd:paddedBy=" " nxsd:length="25"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <!--COBOL declaration : 01 PAYROLL-F-RECORD -->
   <xsd:element name="PAYROLL-F-RECORD" nxsd:lookAhead="" nxsd:lookFor="">
    <xsd:complexType>
     <xsd:sequence>
      <!--COBOL declaration : 05 PAYROLL-F-EMPLOYEE-NUMBER PIC X(10)-->
      <xsd:element name="PAYROLL-F-EMPLOYEE-NUMBER" type="xsd:string"
              nxsd:style="fixedLength" nxsd:padStyle="tail"
              nxsd:paddedBy=" " nxsd:length="10"/>
      <!--COBOL declaration : 05 PAYROLL-F-TRANS-CODE PIC X(02)-->
      <xsd:element name="PAYROLL-F-TRANS-CODE" type="xsd:string"
              nxsd:style="fixedLength" nxsd:padStyle="tail"
              nxsd:paddedBy=" " nxsd:length="2"/>
      <!--COBOL declaration : 05 PAYROLL-F-IDENTIFIER-VALUE PIC X(03)-->
      <xsd:element name="PAYROLL-F-IDENTIFIER-VALUE" type="xsd:string"
              nxsd:style="fixedLength" nxsd:padStyle="tail"
              nxsd:paddedBy=" " nxsd:length="3"/>
      <!--COBOL declaration : 05 PAYROLL-F-NAME PIC X(30)-->
      <xsd:element name="PAYROLL-F-NAME" type="xsd:string"
              nxsd:style="fixedLength" nxsd:padStyle="tail"
              nxsd:paddedBy=" " nxsd:length="30"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
```

```
   <!--COBOL declaration : 01 PAYROLL-H-RECORD -->
   <xsd:element name="PAYROLL-H-RECORD" nxsd:lookAhead="" nxsd:lookFor="">
    <xsd:complexType>
     <xsd:sequence>
      <!--COBOL declaration : 05 PAYROLL-H-EMPLOYEE-NUMBER PIC X(10)-->
      <xsd:element name="PAYROLL-H-EMPLOYEE-NUMBER" type="xsd:string"
               nxsd:style="fixedLength" nxsd:padStyle="tail"
               nxsd:paddedBy=" " nxsd:length="10"/>
      <!--COBOL declaration : 05 PAYROLL-H-TRANS-CODE PIC X(02)-->
      <xsd:element name="PAYROLL-H-TRANS-CODE" type="xsd:string"
              nxsd:style="fixedLength" nxsd:padStyle="tail"
              nxsd:paddedBy=" " nxsd:length="2"/>
      <!--COBOL declaration : 05 PAYROLL-H-HED-NUMBER PIC 9(03)-->
      <xsd:element name="PAYROLL-H-HED-NUMBER" type="xsd:long"
              nxsd:style="fixedLength" nxsd:padStyle="head"
              nxsd:paddedBy="0" nxsd:length="3"/>
       <!--COBOL declaration : 05 FILLER PIC X(30)-->
       <xsd:element name="FILLER" type="xsd:string"
               nxsd:style="fixedLength" nxsd:padStyle="tail"
               nxsd:paddedBy=" " nxsd:length="30"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:choice>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

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. Therefore, 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 generated schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--Native format was generated from COBOL copybook :
D:\work\jDevProjects\CCB\Copybooks\multipleRoot.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/multipleRoots"
       xmlns:tns="http://TargetNamespace.com/ccb/multipleRoots"
       elementFormDefault="qualified" attributeFormDefault="unqualified"
       nxsd:version="NXSD" nxsd:encoding="ASCII"
       nxsd:byteOrder="littleEndian" nxsd:stream="chars">
 <xsd:element name="emp-info">
  <xsd:complexType>
```

```
   <xsd:sequence minOccurs="1" maxOccurs="unbounded">
    <!--COBOL declaration : 05 ORG-NUM PIC 99-->
    <xsd:element name="ORG-NUM" type="xsd:long" nxsd:style="fixedLength"
            nxsd:padStyle="head" nxsd:paddedBy="0" nxsd:length="2"/>
    <!--COBOL declaration : 05 EMP-RECORD-->
    <xsd:element name="EMP-RECORD">
     <xsd:complexType>
      <xsd:sequence>
       <!--COBOL declaration : 10 EMP-SSN PIC 9(4)V(6)-->
       <xsd:element name="EMP-SSN" type="xsd:decimal"
               nxsd:style="virtualDecimal" extn:assumeDecimal="4"
               extn:picSize="9"/>
       <!--COBOL declaration : 10 EMP-WZT PIC 9(6)-->
       <xsd:element name="EMP-WZT" type="xsd:long"
               nxsd:style="fixedLength" nxsd:padStyle="head"
               nxsd:paddedBy="0" nxsd:length="6"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

In this (non-`01` level) case, an unbounded sequence of the root level items is generated.

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's behavior is the same. 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:

```
<?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:

```
<?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:

```
<?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 the numeric types follow formats specified according to IBM COBOL formats. If the data file originates from a different system using different layouts, the generated schema requires modification.

Native Schema Constructs

This section contains the following topics:

- Defining Fixed Length Data
- Defining Terminated Data
- Defining Surrounded Data
- Defining Lists
- Defining Arrays
- Conditional Processing
- Defining Dates
- Using Variables

Defining Fixed Length Data

Fixed length data in the native format can be defined in the native schema 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 the `paddedBy` and `padStyle` as head or tail. When the data is read, the pads are trimmed accordingly.

```
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.

```
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.

```
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 the `nxsd:quotedBy` to be safe. Specifying `nxsd:quotedBy` means the corresponding native data may or may not be quoted. If it is quoted, the actual data is read from the begin quote to the end quote as specified in `nxsd:quotedBy`. Otherwise, it is read until the `terminatedBy` is found.

The following examples are provided:

- Optionally quoted
- Not quoted

Native Data Format to Be Translated: Optionally Quoted

```
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.

```
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>
```

Defining Surrounded Data

This is used when the native data is surrounded by a mark.

The following examples are provided:

- 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

```
(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

```
.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 is for lists with the following characteristics:

- All items separated by the same mark, except the last item (bounded)

- All items separated by the same mark, including the last item (unbounded)

Native Data Format to Be Translated: All Items Separated by the Same Mark, But the Last Item Terminated by a Different Mark (Bounded)

```
125,200,255
```

Native Schema: All Items Separated by the Same Mark, But the Last Item Terminated by a Different Mark (Bounded)

```
<?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: All Items Separated by the Same Mark, But the Last Item Terminated by a Different Mark (Bounded)

```
<list xmlns="http://www.oracle.com/ias/processconnect">
   <Red>125</Red>
   <Green>200</Green>
   <Blue>255</Blue>
</list>
```

Native Data Format to Be Translated: All Items Separated by the Same Mark, Including the Last Item (Unbounded)

```
configure;startup;runtest;shutdown;
```

Native Schema: All Items Separated by the Same Mark, Including the Last Item (Unbounded)

```
<?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: All Items Separated by the Same Mark, Including the Last Item (Unbounded)

```
<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 examples are provided:

- All cells separated by the same mark, except the last cell (bounded)
- All cells separated by the same mark, including the last cell (unbounded)
- Cells not separated by any mark, except the last cell (bounded)
- The number of cells also being read from the native data

Native Data Format to Be Translated: All Cells Separated by the Same Mark, But the Last Cell Terminated by a Different Mark (Bounded)

```
"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: All Cells Separated by the Same Mark, But the Last Cell Terminated by a Different Mark (Bounded)

```
<?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: All Cells Separated by the Same Mark, But the Last Cell Terminated by a Different Mark (Bounded)

```
<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>
```

Native Data Format to Be Translated: All Cells Separated by the Same Mark, Including the Last Cell (Unbounded)

```
"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: All Cells Separated by the Same Mark, Including the Last Cell (Unbounded)

```
<?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: All Cells Separated by the Same Mark, Including the Last Cell (Unbounded)

```
<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>
```

Native Data Format to Be Translated: Cells Not Separated by Any Mark, But the Last Cell Terminated by a Mark (Bounded)

```
"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: Cells Not Separated by Any Mark, But the Last Cell Terminated by a Mark (Bounded)

```
<?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: Cells Not Separated by Any Mark, But the Last Cell Terminated by a Mark (Bounded)

```
<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>
```

Native Data Format to Be Translated: The Number of Cells Being Read from the Native Data

```
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: The Number of Cells 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="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: The Number of Cells Being Read from the Native Data

```
<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>
```

Conditional Processing

The following examples are provided:

- Processing one element within a choice model group based on the condition
- Processing elements based within a sequence model group on the condition

Native Data Format to Be Translated: Processing One Element within a Choice Model Group Based on the Condition

```
PO28/06/2004^|ABCD Inc.|Oracle
OracleApps025070,000.00
Database  021230,000.00
ProcessCon021040,000.00
PO01/07/2004^|EFGH Inc.|Oracle
Websphere 025070,000.00
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: Processing One Element within a Choice Model Group Based on the Condition

```
<?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: Processing One Element Within a Choice Model Group Based on the Condition

```
<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>Websphere </Id>
            <Quantity>50</Quantity>
            <Price>70,000.00</Price>
         </Line-Item>
         <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>
```

Native Data Format to Be Translated: Processing Elements within a Sequence Model Group Based on the Condition

```
PO28/06/2004^|ABCD Inc.|Oracle
OracleApps025070,000.00
Database  021230,000.00
ProcessCon021040,000.00
PO01/07/2004^|EFGH Inc.|Oracle
Websphere 025070,000.00
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: Processing Elements within a Sequence Model Group Based on the Condition

```
<?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: Processing Elements within a Sequence Model Group Based on the Condition

```
<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>Websphere </Id>
         <Quantity>50</Quantity>
         <Price>70,000.00</Price>
      </Line-Item>
      <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 example shows how to define dates.

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
```

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>
```

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>
```

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. Any legal encoding supported by `java.io.InputStreamReader`.
`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`.
`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 case of 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`.

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
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 already declared variable
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, against the specified string in the conditionValue
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, 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, the node is processed; otherwise, it is skipped.
paddedBy	The string used for padding
padStyle	head, tail, or none
quotedBy	The native data being quoted by the specified string
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.

Table 6–5 (Cont.) Constructs Applicable On All Tags Other Than the <schema> Tag

Construct	Description
`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 already declared variables a valid value

Summary

Oracle BPEL Process Manager requires native schemas for translation, which are based on XML schema. However, not all commonly used formats use XML schema files. This chapter describes the Native Format Builder Wizard, which enables you to create native schemas used for translation. The Native Format Builder Wizard guides you through the creation of a native schema file from delimited formats (such as CSVs), fixed length formats, DTD formats, and COBOL Copybook formats. This chapter also provides use cases and constructs for the schema.

A

Troubleshooting and Workarounds

This appendix describes Oracle BPEL Process Manager troubleshooting methods.

This appendix contains the following topics:

- Troubleshooting the Oracle Application Server Adapter for Databases
- Troubleshooting the Oracle Application Server Adapter for Databases When Using Stored Procedures
- Troubleshooting the Oracle Application Server Adapter for Files/FTP
- Troubleshooting the Oracle Application Server Adapter for Advanced Queuing
- Summary

Troubleshooting the Oracle Application Server Adapter for Databases

The following sections describe possible issues and solutions when using the Oracle Application Server Adapter for Databases (database adapter).

Could Not Create OracleAS TopLink Session Exception

Problem

At run time, you may see the `"Could not create the TopLink session"` exception.

Solution

This common error occurs when the run-time connection is not configured properly. See "Deployment" on page 4-44 for more information.

Could Not Find Adapter for eis/DB/*my_connection*

Problem

You may see the `"Could not find adapter for eis/DB/`*`my_connection`*`/...."` exception.

Solution

See "Deployment" on page 4-44 for more information.

Changes Through TopLink Mapping Workbench

Changes through TopLink Mapping Workbench require you to run the Adapter Configuration Wizard again in edit mode to force a refresh of the `toplink_mappings.xml` file.

Redeploying from the Command Line

If you redeploy using `obant`, unless the `bpel.xml` or `.bpel` files have changed, the redeployment is skipped by design.

Cannot Change Customers_table.xsd

Problem

Changes to `Customers_table.xsd` are not reflected, or you get an exception.

Solution

You cannot specify the XSD format that the database adapter produces. See "XML Schema Definition (XSD)" on page 4-43 for details.

No Target Foreign Keys Error

Problem

After clicking **Finish**, or at deployment, you may see the following exception:

```
Caused by Exception [TOPLINK-0] (OracleAS TopLink - 10g (9.0.4.4) (Build 040705)):
oracle.toplink.exceptions.IntegrityException

Descriptor Exceptions:
----------------------

Exception [TOPLINK-64] (OracleAS TopLink - 10g (9.0.4.4) (Build 040705)):
oracle.toplink.exceptions.DescriptorException
Exception Description: No target foreign keys have been specified for this
mapping.
Mapping: oracle.toplink.mappings.OneToManyMapping[phonesCollection]
Descriptor: Descriptor(Test.Customers --> [DatabaseTable(CUSTOMERS)])
```

This generally means that there was a problem in the wizard.

Solution

The simplest solution is to create all constraints on the database first. Also, depending on the problem, you may only need to fix something in the offline tables, and then run the wizard again.

If you want to create a one-to-many mapping from **CUSTOMERS** to **PHONES**, you need a foreign key constraint on **PHONES**.

This procedure assumes that this constraint does not exist on the database, and that you tried to create it with the wizard and it generated an exception.

- In your JDeveloper BPEL Designer project, click the plus sign (+) in the **Applications Navigator** to add files to your project. Select **database** > ***schemaName*** > ***schemaName*.schema**. This imports all your database objects.
- Open the **PHONES** table and manually create the foreign key constraint from **PHONES** to **CUSTOMERS**.

- Save all.
- Now open the OracleAS TopLink project. In JDeveloper BPEL Designer, go to **Application Sources** > **TopLink** > **TopLink Mappings**. In the **Structure** window, open **CUSTOMERS** and double-click the **phonesCollection** mapping.

  You should now see a **Table Reference** tab in the main window. This was probably blank previously. From the menu, select the one you just created.
- Save again.
- Edit the database partner link.

  Click **Next** to the end in the wizard, and then click **Finish** and **Close**.

  This refreshes `toplink_mappings.xml` for your project.
- Open the `toplink_mappings.xml` file in JDeveloper BPEL Designer. You may need to add it to the project first.
- Search for `phonesCollection`.
  - You should find a tag like this:

    ```
    <database-mapping>
    <attribute-name>phonesCollection</attribute-name>
    ```

  - Scroll down and you should now see something like this:

    ```
    <source-key-fields>
    <field>CUSTOMERS.someColumn</field>
    </source-key-fields>
    <target-foreign-key-fields>
    <field>PHONES.someColumn</field>
    </target-foreign-key-fields>
    ```

  - If you do not see the tags shown here, then manually add them. This is not the preferred method because `toplink_mappings.xml` is a generated file, which gets refreshed whenever you edit a database partner link.
- Undeploy the old process in Oracle BPEL Console, and redeploy your fixed process with a new revision number.

No Primary Key Exception

Problem

After clicking **Finish**, or at deployment, you may see the following exception:

```
Caused by Exception [TOPLINK-0] (OracleAS TopLink - 10g (9.0.4.4) (Build 040705)):
oracle.toplink.exceptions.IntegrityException

Descriptor Exceptions:
----------------------
Exception [TOPLINK-46] (OracleAS TopLink - 10g (9.0.4.4) (Build 040705)):
oracle.toplink.exceptions.DescriptorException
Exception Description: There should be one non-read-only mapping defined for the
primary key field [PHONES.ID].
Descriptor: Descriptor(Test.Phones --> [DatabaseTable(PHONES)])
```

This probably means that no primary key was defined for `PHONES`.

Solution

If this exception appears in conjunction with the **No Target Foreign Keys** error, then see "No Target Foreign Keys Error" on page A-2 and resolve that problem first. Otherwise, do the following:

- Open **Application Sources > TopLink > TopLink Mappings.**
  - In the **Structure** window, double-click **PHONES**.

    On the first page, you should see **Primary Keys:**. Ensure that columns are selected and that they are mapped in the project.
  - Save.
- Edit the database partner link.

  Click **Next** to the end in the wizard, and then click **Finish** and **Close**.
- Open `toplink_mappings.xml`. For the `PHONES` descriptor, you should see something like this:

  ```
  <primary-key-fields>
  <field>PHONES.ID</field>
  </primary-key-fields>
  ```

  - Make sure that there is at least one field, and for that field, make sure you can find it somewhere else in the `toplink_mappings.xml` file. If you can, then this means that the database adapter can detect it. If not, then you must map this field.
- Open **Application Sources > Test > Phones**.

  You should see Java code. Add a line as follows:

  ```
  long id;
  ```
- Save.
- Now open **Application Sources > TopLink > TopLink Mappings.**

  Double-click **PHONES** and go to the Structure window.

  You should be able to map the id as **DirectToField**, and set the database field to **ID**.
- Save and then edit the database partner link.

  Click **Next** to the end in the wizard, and then click **Finish** and **Close**.
- Undeploy the old process in Oracle BPEL Console, and redeploy your fixed process with a new revision number.

dateTime Conversion Exceptions

Problem

You may get a conversion exception when you pass in an `xs:dateTime` value to the database adapter.

Solution

If an attribute is of type `xs:dateTime`, then the database adapter is expecting a string in one of the following formats:

```
1999-12-25T07:05:23-8:00
```

```
1999-12-25T07:05:23.000-8:00
1999-12-25T15:05:23:000Z
1999-12-25T15:05:23
```

The format `1999-12-25` is accepted, although it is not a valid `xs:dateTime` value. The `xs:dateTime` format is `yyyy-MM-ddTHH:mm:ss.SSSZ`, where

- `yyyy` is the year (2005, for example)
- `MM` is the month (01 through 12)
- `dd` is the day (01 through 31)
- `HH` is the hour (00 through 23)
- `mm` is the minute (00 through 59)
- `ss` is the second (00 through 59)
- `SSS` is milliseconds (000 through 999), optional
- `Z` is the time zone designator (`+hh:mm` or `-hh:mm`), optional

A `DATE` column may exist on an Oracle Database, which can accept the `25-DEC-1999` date format. However, this is not a date format that the database adapter can accept. The following workaround applies to TopLink only.

- If you want to pass in the `25-DEC-1999` date format, then map the attribute as a plain string. The database adapter passes the value through as-is.
  - To do this, you must edit the offline database table and change the column datatype from `DATE` to `VARCHAR2`.
- Save.
- Edit the database partner link.

  Click **Next** to the end in the wizard, and then click **Finish** and **Close**.

While not a valid `xs:dateTime` format, the format `yyyy-mm-dd` is a valid `xs:date` format.

Issues with Oracle DATE

Problem

The *time* portion of `DATE` fields may be truncated on Oracle9 or greater platforms when using `oracle.toplink.internal.databaseaccess.DatabasePlatform`. For example, `2005-04-28 16:21:56` becomes `2005-04-28T00:00:00.000+08:00`.

Or, the millisecond portion of `DATE` fields may be truncated on Oracle9 or greater platforms when using `oracle.toplink.internal.databaseaccess.Oracle9Platform`. For example, `2005-04-28 16:21:56.789` becomes `2005-04-28T16:21:56.000+08:00`.

Or, you may have trouble with `TIMESTAMPTZ` (time stamp with time zone) or `TIMESTAMPLTZ` (time stamp with local time zone).

Solution

You must set the `platformClassName` for Oracle platforms, because these include special workarounds for working with date-time values on Oracle. So, if you are

connecting to an Oracle9 platform, you must set the `platformClassName` accordingly.

Due to an issue with the time portion of `DATE` being truncated with Oracle9 JDBC drivers, the property `oracle.jdbc.V8Compatible` was set when using any Oracle platform class name. Therefore, use `oracle.toplink.internal.databaseaccess.Oracle9Platform` to solve the time truncation problem.

However, starting with Oracle9, dates started to include millisecond precision. Setting `oracle.jdbc.V8Compatible` in response had the drawback of returning the milliseconds as `000`, as an Oracle8 database did. (This also introduced an issue with null `IN/OUT DATE` parameters for stored procedure support.) You do not see any truncation (of the time portion or milliseconds) when using the `Oracle9Platform` class.

You must also use the `Oracle9Platform` class if you have `TIMESTAMPTZ` and `TIMESTAMPLTZ`.

If you want `DATE` to be treated like a date (with no time portion), set the attribute-classification in the `toplink_mappings.xml` to `java.sql.Date`.

In general, if you are having an issue with a particular database, check to see if TopLink has a custom `platformClassName` value for that database, and whether you are using it.

See "Deployment" on page 4-44 and Table 4–19, " Application Server Connection Pooling" on page 4-47 for more information.

Handling a Database Adapter Fault

To understand how to handle faults, such as a unique constraint violation on inserts or when a database or network is temporarily unavailable, see the `InsertWithCatch` tutorial at *`Oracle_Home`*`\integration\orabpel\samples\tutorials\122.DBAdapter`.

BPEL Process Does Not Run Against Another Database

Problem

A BPEL process modeled against one database does not run against another database.

The most likely cause for this problem is that you are using a different schema in the second database. For example, if you run the wizard and import the table `SCOTT.EMPLOYEE`, then, in the `toplink_mappings.xml` file, you see `SCOTT.EMPLOYEE`. If you run the sample in the `USER` schema on another database, you get a `"table not found"` exception.

Solution

Until qualifying all table names with the schema name is made optional, manually edit `toplink_mappings.xml` and replace `SCOTT.` with nothing, as shown in the bold portions of the following example.

Change:

```
<project>
   <project-name>toplink_mappings</project-name>
   <descriptors>
      <descriptor>
         <java-class>BPELProcess1.A</java-class>
```

```
        <tables>
           <table>SCOTT.A</table>
        </tables>
```

To:

```
<project>
   <project-name>toplink_mappings</project-name>
   <descriptors>
      <descriptor>
         <java-class>BPELProcess1.A</java-class>
         <tables>
            <table>A</table>
         </tables>
```

You must repeat this step every time after running the Adapter Configuration Wizard.

> **Note:** Having `EMPLOYEE` on both the `SCOTT` and `USER` schemas, and querying against the wrong table, can result in a problem that is difficult to detect. For this reason, the database adapter qualifies the table name with the schema name.

Only One Employee Per Department Appears

Problem

Many departments with many employees are read in, but only one employee per department appears.

Solution

You must use a transform with a `for-each` statement. An **Assign** activity with a too-simplistic XPath query can result in only the first employee being copied over.

For an example of how to use a transform for database adapter outputs, go to

Oracle_Home\integration\orabpel\samples\tutorials\122.DBAdapter\MasterDetail

Outbound SELECT on a CHAR(X) or NCHAR Column Returns No Rows

Problem

If you use an outbound `SELECT` to find all employees where `firstName = some_parameter`, then you have a problem if `firstName` on the database is a `CHAR` column, as opposed to a `VARCHAR2` column.

It is a known problem with some databases that if you insert a `CHAR` value (for example, `'Jane'`) into a `CHAR(8)` field, then the database pads the value with extra spaces (for example, `'Jane '`).

If you then execute the query

```
SELECT ... WHERE firstName = 'Jane';
```

no rows may be returned. Although you are querying for the same value that you inserted, and some tools such as SQL*Plus and SQL Worksheet operate as expected, the query does not work with the database adapter.

Solution

The best practice is to use a CHAR column for fields that must be fixed, such as SSN, and VARCHAR2 for columns that can take a variable length, such as firstName.

Transforming the value to add padding may be difficult, and using SELECT to trim the value on the database (as opposed to padding the other side) requires using SQL statements. For example:

```
SELECT ... WHERE trim(firstName) = #firstName;
```

Note that # is an OracleAS TopLink convention for denoting input parameters.

ORA-00932: Inconsistent Datatypes Exception Querying CLOBs

Problem

When querying on table A, which has a one-to-one relationship to B, where B contains a CLOB, you may see the following exception:

```
Exception Description: java.sql.SQLException: ORA-00932: inconsistent
datatypes: expected - got CLOB
```

Solution

A SELECT returning CLOB values must not use the DISTINCT clause. The simplest way to avoid DISTINCT is to disable batch attribute reading from A to B. Batch reading is a performance enhancement that attempts to simultaneously read all Bs of all previously queried As. This query uses a DISTINCT clause. Use joined reading instead, or neither joined reading nor batch attribute reading.

Because both DISTINCT and CLOBs are common, you may see this problem in other scenarios. For example, an expression like the following uses a DISTINCT clause:

```
SELECT DISTINCT dept.* from Department dept, Employee emp WHERE ((dept.ID =
emp.DEPTNO) and (emp.name = 'Bob Smith'));
```

See "Relational-to-XML Mappings (toplink_mappings.xml)" on page 4-36 for more information about batch reading and joined reading.

Merge Sometimes Does UPDATE Instead of INSERT, or Vice Versa

Problem

You may sometimes notice that merge performs an UPDATE when it should do an INSERT, or vice versa.

Solution

Merge works by first determining, for each element in the XML, whether the corresponding database row exists or not. For each row, it does an existence check. There are two known limitations with the existence check.

First, you can configure the existence check to either **Check cache** or **Check database**. You can configure this for each descriptor (mapped table) in your Mapping Workbench Project. The default is **Check database**, but TopLink's check database works like "check cache first, then database" for performance reasons. If a row exists in the cache, but was deleted from the database (the cache is stale), then you may see an UPDATE when you expect an INSERT. You can configure caching and a WeakIdentityMap is used by default, meaning rows are only held in memory while being processed. However, Java garbage collection is not controlled by the adapter.

Therefore, if you insert a row, delete it in a separate process, and insert it again, all within a very short time, you may see an INSERT and then an UPDATE. One solution is to use NoIdentityMap. However, performance may suffer, and if you are doing SELECT statements on a mapped schema with complex cycles (which you should avoid!), then the adapter may become trapped in an endless loop when building the XML.

Second, there is a timing issue to doing a read first and then later an INSERT or UPDATE. If the same row is simultaneously inserted by multiple invokes, then each may do an existence check that returns false, and then all attempt an INSERT. This does not seem realistic, but the following scenario did come up:

A polling receive reads 100 employee rows and their departments from database A. With maxRaiseSize set to 1, 100 business process instances were initiated. This led to 100 simultaneous invokes to database B, one for each employee row. No problems were encountered when existence checking on employee, but some employees had the same department. Hence, many of the 100 invokes failed because the existence checks on department were more or less simultaneous.

There are two solutions to this problem. The first is to avoid it. In a data synchronization-style application, setting maxRaiseSize to unlimited boosts performance and eliminates this problem. A second solution is to retry the merge in your BPEL process. Optimistic lock and concurrency exceptions are common, and the best solution is usually to wait and try again a short time later.

Integrity Violation Occurs with Delete or DeletePollingStrategy

Problem

Child records found an integrity violation with DeletePollingStrategy.

When deleting rows, you must be aware of integrity constraints. For example, if DEPARTMENT has a one-to-many relationship to EMPLOYEE, that means DEPTID is a foreign key on EMPLOYEE. If you delete a DEPARTMENT record, but not its employees, then DEPTID becomes a broken link and this can trigger an integrity constraint.

This problem occurs because you imported a table by itself and did not import its related tables. For example, if you import only the DEPARTMENT table from the database and not the EMPLOYEE table, which has an integrity constraint on column DEPTID, then the database adapter does not know about EMPLOYEE and it cannot delete a record from DEPARTMENT. You receive an exception.

Solution

Make sure you import the master table and all its privately-owned relationships. Or set the constraint on the database to CASCADE for deletions, or use a nondelete polling strategy.

Make sure that the one-to-many relationship between DEPARTMENT and EMPLOYEE is configured to be privately owned. It is by default, but if the above fails, check the run-time X-R mappings file. See "Relational-to-XML Mapping" on page 4-4 for more information.

If the problem is not this simple, OracleAS TopLink supports shallow/two-phase inserts (but does not support this for DELETE). For example, if A has a foreign key pointing to B, and B has a foreign key pointing to A, then there is no satisfactory order by which you can delete both A and B. If you delete A first, then you orphan B. If you delete B first, then you orphan A. The safest DELETE is a two-phase DELETE that performs an UPDATE first as follows:

```
UPDATE B set A_FK = null;
DELETE from A;
DELETE from B;
```

Some Queried Rows Appear Twice or Not at All in the Query Result

Problem

When you execute a query, you may get the correct number of rows, but some rows appear multiple times and others do not appear at all.

This behavior is typically because the primary key is configured incorrectly. If the database adapter reads two different rows that it thinks are the same (for example, the same primary key), then it writes both rows into the same instance and the first row's values are overwritten by the second row's values.

Solution

- Open **Application Sources > TopLink > TopLink Mappings**. In the Structure window, double-click **PHONES**. On the first page, you should see **Primary Keys**. Make sure that the correct columns are selected to make a unique constraint.
- Save and then edit the database partner link.

  Click **Next** to the end, and then click **Finish** and **Close**.
- Open your `toplink_mappings.xml` file. For the `PHONES` descriptor, you should see something like this:

```
<primary-key-fields>
<field>PHONES.ID1</field>

<field>PHONES.ID2</field>
</primary-key-fields>
```

Importing a Same-Named Table, with Same Schema Name, but Different Databases

Problem

Importing a table from a database on one host and also importing a table with the same name, and the same schema name, from a database on another host raises an error.

Solution

Create one project against database #1 and model the adapter service. Next create a second project against database #2 and model the adapter service. (Because the databases are on different hosts, you use different database connections.) Then create a third project, but do not run the Adapter Configuration Wizard. Instead, copy the BPEL artifacts (WDSL, XSD, and `toplink_mapings.xml`) from projects one and two. Deploy only the third project.

If the two tables are identical, or if the data you are interested in is identical, then you need not follow the preceding procedure.

Problems Creating a Relationship Manually for a Composite Primary Key

Problem

In the Relationship window of the Adapter Configuration Wizard, all elements of the primary key appear and cannot be removed. Therefore, a foreign key referring to only part of the composite primary key cannot be created.

Solution

Because foreign key constraints must map to every part of the primary key (not a subset), there is no solution. The database adapter allows a foreign key only with a corresponding primary key at the other end.

Must Fully Specify Relationships Involving Composite Primary Keys

The wizard does not let you create an ambiguous relationship. For example, assume that a `PurchaseOrder` has a 1-1 `billTo` relationship to a `Contact`. For uniqueness, the primary key of `Contact` is `name` and `province`. This means `PurchaseOrder` must have two foreign keys (`bill_to_name` and `bill_to_province`). If there is only one foreign key (`bill_to_name`), then the wizard does not allow you to create that ambiguous 1-1 relationship. Otherwise, the same purchase order can be billed to multiple people.

Database Adapter Throws an Exception When Using a BFILE

The `BFILE`, `USER DEFINED`, `OBJECT`, `STRUCT`, `VARRAY`, and `REF` types are not supported.

During Design-Time, Wizard Does Not Allow Deletion of a Table

Because a table may be used for other services inside the same project, it cannot be deleted within the interface. No problems occur if the unneeded table remains as part of your project.

Changes to JDeveloper Project Are Made Even If Wizard Is Cancelled

Problem

When you run the Adapter Configuration Wizard, any changes that affect the TopLink project (importing tables, creating or removing mappings, specifying an expression, and so on) are applied immediately, and are *not undone* if you cancel the wizard.

Solution

If you remove one or more of the autogenerated relationships and later want to get them back, you must reimport the tables containing the corresponding database constraints.

Problems Removing a Relationship, Then Adding a New Relationship with the Same Name

Problem

The database adapter can become unstable if, within the same wizard session, you remove a relationship and then immediately create a relationship with the same name.

Solution

You can do one of the following:

- Give the new relationship a different name from the one you removed.
- Finish the wizard after you remove the first relationship. Then, start the wizard again in edit mode to add the new relationship, using the same name as the deleted relationship.

Problems Importing Third-Party Database Tables with Unsupported Database Types

Problem

When you import tables from some third-party databases, JDeveloper can encounter problems handling certain datatypes. You may see error messages such as `"Columns of type VARCHAR cannot have a size specified"` or `"A primary or unique key must define at least one column"`.

Solution

Use the following workaround:

1. Click **OK** to dismiss the error message; then cancel the wizard.
2. Edit the offline table definition to change the type of the columns mentioned in the error message to the closest supported type.
3. Run the Adapter Configuration Wizard again and continue with the rest of the wizard.

The offline table definitions for your project can be found under the **Database Objects** > ***schema name*** node in the **Applications Navigator** of JDeveloper BPEL Designer. If you do not see this node after importing your tables, then you can add it manually by clicking the **Add to *project name*.jpr** button in the **Applications Navigator**. Then select the **database** > ***schemaName*** > ***schemaName*.schema** file.

See "Configuring Offline Database Tables" on page 4-35 for more information.

Problems Importing Object Tables

Problem

JDeveloper does not currently support importing object tables. If you try to import an object table, then you see the following message: `"The following tables were 'Object Tables' and aren't supported offline."`

Solution

There is currently no workaround for this problem.

Relationships Not Autogenerated When Tables Are Imported Separately

Problem

If tables are imported one at a time, relationships are not generated even if foreign key constraints exist on the database.

Solution

Relationship mappings can be autogenerated only if all the related tables are imported in one batch. When importing tables, you can select multiple tables to be imported as a group. If you have related tables, then they should all be imported at the same time.

Primary Key Is Not Saved

Problem

If you try to create a relationship that has the same name as the primary key field name, then you encounter a problem in which the PK field becomes unmapped.

Solution

To add the PK mapping back manually, follow these instructions:

1. Open the Java source for the descriptor to which you want to add the mapping (for example, `Movies.java`).
2. Add a new Java attribute appropriate for the field to which you are mapping. For example, if the PK of the `Movies` table is a `VARCHAR` field named `TITLE`, then create a new attribute: `"private String title;"`
3. Save the Java file.
4. Click the **TopLink Mappings** node in the **Applications - Navigator** pane; then choose the **Descriptor** from the **TopLink Mappings - Structure** pane. You see the newly created attribute in the **Descriptor** as unmapped (in this example, **title**).
5. Right-click the new attribute and select **Map As > Direct To Field**.
6. Double-click the new attribute. The TopLink Mappings editor should appear in the main JDeveloper window. Change the database field to match the PK field on the database (in this example, **TITLE**).
7. Click the **Descriptor** in the **TopLink Mappings - Structure** pane. Ensure that the PK field has a check box next to it in the **Primary Keys** list.
8. Run the Adapter Configuration Wizard again and continue with the rest of the wizard.

Troubleshooting the Oracle Application Server Adapter for Databases When Using Stored Procedures

The following sections describe possible issues and solutions when using the database adapter for stored procedures.

Design-Time Problems: Unsupported Parameter Types

Problem

Using an unsupported parameter type in the chosen API is a common problem. In the `BOOLEAN` example described in "Support for PL/SQL BOOLEAN" on page 4-71, the procedure `BOOLPROC` has a single parameter, `B`, whose type is `PL/SQL BOOLEAN`, which is not supported. When the XSD is generated, you see a WSDL write error indicating that the database type is either not supported or is not implemented.

To generate an XSD for APIs containing parameters whose types are user-defined, those types must first be defined in the database and be accessible through the

associated service connection. This error also occurs if such types have not been created (that is, *implemented*) in the database or if the database is inaccessible.

Solution

Ensure that only supported datatypes are used as types for parameters when choosing an API. If the types are user-defined, check to ensure that the types are defined in the database and that the database is accessible when the attempt to generate the XSD is made.

Problem

When the type of one or more of the parameters in the chosen API is a user-defined type that belongs to a different schema, a design-time problem can occur. Assume type `OBJ` is created in `SCHEMA2`, as in

```
CREATE TYPE OBJ AS OBJECT (X NUMBER, Y VARCHAR2 (10));
```

And, a procedure is created in `SCHEMA1` that has a parameter whose type is `SCHEMA2.OBJ`, as in

```
CREATE PROCEDURE PROC (O SCHEMA2.OBJ) AS BEGIN … END;
```

If you attempt to create the XSD for procedure `PROC`, you see an error message similar to the following:

```
Error while writing wsdl file
D:\OraBPEL\integration\jdev\mywork\Workspace\Test\Schema1.wsdl.
Exception: WSDLException: faultCode=OTHER_ERROR: The datatype, OBJ, which belongs
to the schema, SCHEMA2 is currently not supported. The datatype must belong to the
same schema as the stored procedure.
```

Solution

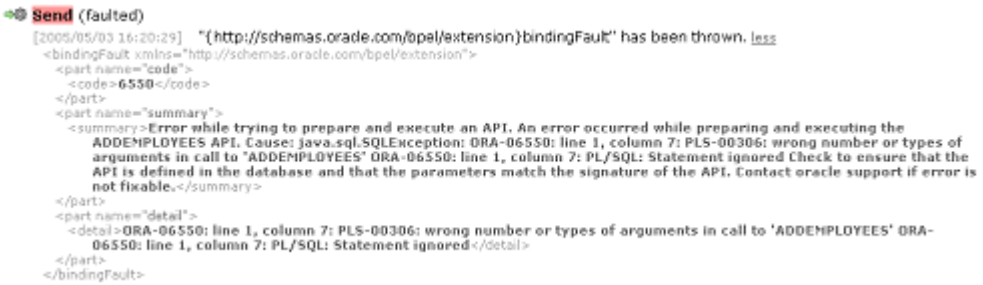
This is a known limitation of the XSD generator. Therefore, the preceding procedure is not supported at this time, although the procedure declaration is legal, provided that the appropriate privileges to access object `OBJ` have been granted to `SCHEMA1`.

Run-Time Problems: Parameter Mismatches

Problem

A mismatch between the formal parameters provided by the instance XML and the actual parameters that are defined in the signature of the stored procedure is a common run-time problem. When this type of error occurs, the **invoke** activity that tried to execute the stored procedure faults, as shown in Figure A–1.

Figure A–1 Example of a Faulted Invoke Due to MIsmatched Parameters



The `bindingFault` has three parts—code, summary, and detail. The information for these parts comes from the `java.sql.SQLException` that gets thrown by JDBC when it attempts to execute the stored procedure. The code is the ORA error number, seen in Figure A–1 as 6550 and as ORA-06550 in the summary and detail parts. The summary includes an adapter-specific error message followed by the message from the `SQLException` and a suggested resolution to the issue. The detail contains just the message from the `SQLException`.

As shown in Figure A–1, the `ADDEMPLOYEES` stored procedure failed to execute due to "wrong number or types of arguments" passed into the API. Possible causes for this problem include:

- An element corresponding to one of the required parameters was not provided in the instance XML.

  Solution: Add the necessary element to resolve the issue.

- More elements than were specified in the XSD were included in the instance XML.

  Solution: Remove the extra elements from the XML.

- The XSD does not accurately describe the signature of the stored procedure. For example, if the type of one of the parameters were to change and the XSD was not regenerated to reflect that change, a type mismatch can occur between the `db:type` of the element and the new type of the modified parameter.

  Solution: Ensure that the parameters match the signature of the API, as indicated in the summary part of the `bindingFault`.

Run-Time Problems: Stored Procedure Not Defined in the Database

Problem

A `bindingFault` can also occur if the stored procedure is not defined in the database when an attempt to execute it is made. In this example, the `ADDEMPLOYEES` API is invoked, but fails to execute because it is not defined. The **invoke** activity faults, as shown in Figure A–2.

Figure A–2

PL/SQL is revealing that "... identifier ADDEMPLOYEES must be declared," which is an indication that the stored procedure may not be defined in the database. This can occur, for example, if the procedure was dropped sometime between when the process was deployed and when the procedure was invoked. This can also occur if the required privileges to execute the stored procedure have not been granted.

Solution

Ensure that the API is defined in the database, as indicated in the summary, and that the appropriate privileges to execute that procedure have been granted.

Some run-time errors can occur if the instance XML does not conform to the XSD generated for the chosen API. XML validation can be enabled in the partner link that coincides with the execution of the API. Edit the `partnerLinkBinding` in the BPEL suitcase located in your process `bpel.xml` file, as shown in bold:

```
<partnerLinkBinding name="PROC">
  <property name="wsdlLocation">Proc.wsdl</property>
  <property name="validateXML">true</property>
</partnerLinkBinding>
```

Adding the `validateXML` property and setting it to `true` enables XML validation for all instance XML passed into the service associated with the partner link that executes the API.

XML validation can also be enabled globally in Oracle BPEL Console. Click **Manage BPEL Domain** and look for the **validateXML** property. Setting the value to **true** causes all XML to be validated. Turning on XML validation helps catch and avoid problems that are associated with the instance XML (rather than the adapter run time).

Troubleshooting the Oracle Application Server Adapter for Files/FTP

The following sections describe possible issues and solutions when using the Oracle Application Server Adapter for Files/FTP (file and FTP adapters).

Changing Logical Names with the Adapter Configuration Wizard

If you later rerun the Adapter Configuration Wizard and change a previously specified logical name to a different name, both the old and new logical names appear in the `bpel.xml` file. You must manually edit the `bpel.xml` file to remove the old logical name.

Creating File Names with Spaces with the Native Format Builder Wizard

While the Native Format Builder Wizard does not restrict you from creating native schema file names with spaces, it is recommended that your file names do not have spaces in them.

Common User Errors

This section describes common user errors.

- On the Adapter Configuration Wizard - Messages window (Figure 2–5 on page 2-12), you can select the **Native format translation is not required (Schema is Opaque)** check box. Opaque cannot be selected in only one direction. Opaque must be selected in the both inbound and outbound directions.
- Messages have a different meaning based on whether they are inbound or outbound. For example, assume you make the following selections:
  - Select **2** from the **Publish Messages in Batches of** list (Figure 2–3 on page 2-7) in the inbound direction.
  - Select **3** from the **Number of Messages Equal** list (Figure 2–7 on page 2-20) in the outbound direction.

If an inbound file contains two records, it is split (debatched) into two messages. However, because **3** was specified in the outbound direction, a file is not created. This is because there are not three outbound messages available. Ensure that you understand the meaning of inbound and outbound messages before selecting these options.

- If the file adapter or the FTP adapter is not able to read or get your file, respectively, it may be because you selected to match file names using the regular expression (regex), but are not correctly specifying the name (Figure 2–3 on page 2-7). See Table 2–2 on page 2-9 for details.
- You may have content that does not require translation (for example, a JPG or GIF image) that you just want to send "as is." The file is passed through in base-64 encoding. This content is known as opaque. To do this, select the **Native format translation is not required (Schema is Opaque)** check box on the Adapter Configuration Wizard - Messages window (Figure 2–5 on page 2-12). If you select this check box, you do not need to specify an XSD file for translation.
- The inbound directory *must* exist for the file adapter or the FTP adapter to read or get your file, respectively.
- If the FTP adapter cannot connect to a remote host, ensure that you have configured the *Oracle_Home*`\integration\orabpel\system\appserver\oc4j\j2ee\home\application-deployments\default\FtpAdapter\oc4j-ra.xml` deployment descriptor file for adapter instance JNDI name and FTP server connection information. See "FTP Adapter for Get File Concepts" on page 2-27 for instructions.
- You *cannot* use completely static names such as `po.txt` for outbound files. Instead, outgoing file names *must* be a combination of static and dynamic portions. This is to ensure the uniqueness of outgoing files names, which prevents files from being inadvertently overwritten. See "Specifying the Outbound File Naming Convention" on page 2-21 for instructions on creating correct outbound file names.
- Two header files are created in the **Applications Navigator** after you finish running the Adapter Configuration Wizard in both directions:
  - *type*`AdapterInboundHeader.wsdl`

    Provides information such as the name of the file being processed and its directory path, as well as data about which message and part define the header operations
  - *type*`AdapterOutboundHeader.wsdl`

    Provides information about the outbound file name

  where *type* is either `ftp` or `file`.

  You can define properties in these header files. For example, you can specify dynamic inbound and outbound file names through use of the `InboundHeader_msg` and `OutboundHeader_msg` message names in the *type*`AdapterInboundHeader.wsdl` and *type*`AdapterOutboundHeader.wsdl` files, respectively.

  You can also set header variables that appear in the BPEL process file. Header variables are useful for certain scenarios. For example, in a file propagation scenario, files are being moved from one file system to another using the file adapter. In this case, it is imperative that you maintain file names across the two systems. Use file headers in both directions and set the file name in the outbound file header to use the file name in the inbound file header.

See the online help available with the Adapters tab of invoke, receive, reply, and pick - OnMessage branch activities for more information.

- The Adapter Configuration Wizard - File Modification Time window (Figure 2–11 on page 2-30) prompts you to select a method for obtaining the modification times of files on the FTP server.

  You must perform the following procedure to obtain this information:

  1. Determine the modification time format supported by the FTP Server by running the command `mdtm` or `ls -al` (whichever is supported by the operating system).
  2. Determine the time difference between the system time (time on which Oracle BPEL Server is running) and the file modification time. Obtain the file modification time by running either `mdtm` or `ls -al` on the FTP server.
  3. Manually add the time difference to the `bpel.xml` as a property:

     ```
     <activationAgents>
       <activationAgent ...>
         <property name="timestampOffset">2592000000</property>
     ```

  4. Specify the **Substring Begin Index** field and **End Index** field values that you determine by running the `mdtm` or `ls -al` command on the FTP server.

Troubleshooting the Oracle Application Server Adapter for Advanced Queuing

The following sections describe possible issues and solutions when using the Oracle Application Server Adapter for Advanced Queuing (AQ adapter).

Inbound Errors

The following sections describe possible issues and solutions for inbound errors when using the AQ adapter.

JNDI Lookup Failed

Sample error:

```
<timestamp> <WARN> <default.collaxa.cube.activation> <AdapterFramework::Inbound>
 JNDI lookup of 'eis/AQ/aqSample2' failed due to: eis/AQ/aqSample2 not found

<timestamp> <ERROR> <default.collaxa.cube.activation> <AdapterFramework::Inbound>
 Error while performing endpoint Activation: ORABPEL-12510

Unable to locate the JCA Resource Adapter via WSDL port element jca:address.

The Adapter Framework is unable to startup the Resource Adapter specified in the
 WSDL jca:address element:  {http://xmlns.oracle.com/pcbpel/wsdl/jca/}address:
 location='eis/AQ/aqSample2'
```

Problem

It is likely that either 1) the resource adapter's RAR file has not been deployed successfully to the OC4J Application Server or 2) the `location` attribute in `$J2EE_HOME/application-deployments/default/deployed-adapter-name/oc4j-ra.xml` has not been set to `eis/AQ/aqSample2`. In the last case, you may have to add a new

connector-factory entry (connection) to the oc4j-ra.xml. Correct this and then restart the BPEL/OC4J Application Server.

Solution

1. Look for the file $J2EE_HOME/application-deployments/default/aqAdapter/oc4j-ra.xml.

   This file should be created when the adapter is deployed, which occurs the first time Oracle BPEL Process Manager is started. If the adapter is undeployed for some reason, deploy the adapter with the following command, then follow step 2:

   ```
   java -jar $J2EE_HOME/admin.jar ormi://localhost admin welcome
   -deployconnector -file <path to AQAdapter.rar> -name
   AqAdapter>
   ```

2. If $J2EE_HOME/application-deployments/default/aqAdapter/oc4j-ra.xml exists, make sure the JNDI location is defined in oc4j-ra.xml.

   For example:

   ```
   <connector-factory location="eis/AQ/aqSample2" connector-name="AQ Adapter">
           <config-property name="connectionString"
                                   value="jdbc:oracle:thin:@myhost:1521:appdb2"/>
           <config-property name="userName" value="scott"/>
           <config-property name="password" value="tiger"/>
       </connector-factory>
   ```

During Initialization, I/O Exception: Network Adapter Did Not Establish the Connection

Sample error:

```
<timestamp> <ERROR> <default.collaxa.cube.activation> <AQ Adapter::Inbound>
 DBConnection_connect: database error while try to connect to
 jdbc:oracle:thin:@localhost:1521:ORCL : Io exception: The Network Adapter could
 not establish the connection
```

Solution

If the connectionString is correct, make sure the database and listener are up, then redeploy the process.

If the connectionString is not correct:

1. Change the connectionString in $J2EE_HOME\application-deployments\default\AqAdapter\oc4j-ra.xml.
2. Restart Oracle BPEL Process Manager.

Incorrect Username/Password

Sample error:

```
<timestamp> <ERROR> <default.collaxa.cube.activation> <AdapterFramework::Inbound>
 Error while performing endpoint Activation: ORABPEL-11929
SQL error while creating managed (database) connection.
SQL error while creating managed (database) connection: Error while trying to
 connect to database.
Error while trying to connect to database using connect string
 "jdbc:oracle:thin:@localhost:1521:appdb2 - java.sql.SQLException: ORA-01017:
 invalid username/password; logon denied
```

Solution

1. Make sure you have the correct username and password in `$J2EE_HOME\application-deployments\default\AqAdapter\oc4j-ra.xml`.
2. Restart Oracle BPEL Process Manager.

Queue Not Found

Sample error:

```
<timestamp> <ERROR> <default.collaxa.cube.activation> <AQ Adapter::Inbound>
 oracle.AQ.AQException: JMS-190: Queue SCOTT.AQ_SUPPORTED_ADT_IN not found
at oracle.AQ.AQUtil.throwAQEx(AQUtil.java:196)
at oracle.AQ.AQOracleSession.getQueue(AQOracleSession.java:720)
at oracle.tip.adapter.aq.database.Queue.connect(Queue.java:102)
at oracle.tip.adapter.aq.database.MessageReader.init(MessageReader.java:575)
```

Solution

Create the queue and redeploy the process. If this process is deployed from the samples, all queue creation scripts are located in `sql\create_queues.sql` under each project.

User Does Not Have DBMS_AQIN Privileges, Which Are Required by the AQ Java API

Sample error:

```
2005-04-20 16:10:52,695> <ERROR> <default.collaxa.cube.activation> <AQ
 Adapter::Inbound> oracle.AQ.AQOracleSQLException: ORA-06550: line 1, column 7:
PLS-00201: identifier 'DBMS_AQIN' must be declared
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
at oracle.AQ.AQOracleQueue.dequeue(AQOracleQueue.java:1795)
at oracle.AQ.AQOracleQueue.dequeue(AQOracleQueue.java:1307)
at
 oracle.tip.adapter.aq.database.MessageReader.readMessage(MessageReader.java:399)
at
oracle.tip.adapter.aq.inbound.AQActivationSpecDequeuer.run(AQActivationSpecDequeue
r.java:163)
at oracle.tip.adapter.fw.jca.work.WorkerJob.go(WorkerJob.java:51)
at oracle.tip.adapter.fw.common.ThreadPool.run(ThreadPool.java:267)
at java.lang.Thread.run(Thread.java:534)
```

Solution

Log on to the database using `sys` as `sysdba`, `GRANT EXECUTE ON SYS.DBMS_AQIN to <username>;`. No deployment is necessary because this failure occurs after the connection has succeeded, the adapter automatically reconnects until the error is gone or the process is undeployed.

Translation Error

Sample error:

```
<timestamp> <ERROR> <default.collaxa.cube.activation> <AQ Adapter::Inbound>
 MessageReader_readMessage: Received TranslationException
<timestamp> <ERROR> <default.collaxa.cube.activation> <AQ Adapter::Inbound>
 ORABPEL-11211
DOM Parsing Exception in translator.
DOM parsing exception in inbound XSD translator while parsing InputStream.
Check the error stack and fix the cause of the error. Contact oracle support if
 error is not fixable.
```

```
at
oracle.tip.pc.services.translation.xlators.xsd.XSDTranslator.translateFromNative(X
SDTranslator.java:131)
```

Solution

Look for the rejected message and find out why it has failed translation. For instance, the message may not be XML, or the XML root element may be incorrect, or the message may be blank. If a rejection handler has been defined for this process, look for the message in the rejection handler. Otherwise, look for the message in the default rejection handler, which is located at

```
Oracle_Home\integration\orabpel\domains\default\archive\jca
\AQMessageRejectionHandler\rejectedMessages
```

For an example of how a user can define the rejection handler, look in

```
ORACLE_
HOME\integration\orabpel\samples\tutorials\124.AQAdapter\AQMessa
geRejectionHandler
```

Subscriber Already Exists When Using MessageRuleSelector

Sample error:

```
<timestamp> <INFO> <default.collaxa.cube.activation> <AQ Adapter::Inbound>
 Subscriber PriorityOneDequeuer already exists in the database.  If the subscriber
 does not contain the rule that you want, please undeploy the business process,
 drop the subscriber with the following sql*plus command, and redeploy.  DECLARE
subscriber sys.aq$_agent;
BEGIN
  subscriber := sys.aq$_agent('<subscriber_name>', NULL, NULL);
  DBMS_AQADM.REMOVE_SUBSCRIBER(
  queue_name => '<queue_name>',
  subscriber => subscriber); END;
```

Solution

This is not a problem if the subscriber has been generated with the rule the user expects. The adapter can create new rule-based subscribers, but cannot modify existing ones. Hence, the first time you deploy the adapter with a nonnull value for `MessageSelectorRule`, a subscriber is created if the consumer does not already exist, using the consumer as the subscriber and the `MessageSelectorRule` as the rule. This message appears in any subsequent redeployment or restart of Oracle BPEL Process Manager.

You can determine if the rule is what you want for the subscriber by entering the following SQL command:

```
SQL> select name, rule, queue from AQ$RuleBased_Raw_In_R;
NAME
------------------------------
RULE
--------------------------------------------------------------------------------
QUEUE
------------------------------
PRIORITYONEDEQUEUER
priority = 1
RULEBASED_RAW_IN
```

Outbound Errors

As a general note, problems in the outbound direction are often not caught at deployment time, because an outbound thread is only activated if there is a message going to outbound.

JNDI Lookup Failed

Sample error:

```
Adapter Framework unable to create outbound JCA connection.
file:/C:/050420/integration/orabpel/domains/default/tmp/.bpel_File2AQBLOB_
1.0.jar/EnqueueBlobPayload.wsdl [ Enqueue_ptt::Enqueue(opaque) ] - : The Adapter
 Framework was unable to establish an outbound JCA connection due to the following
 issue: ORABPEL-12510
Unable to locate the JCA Resource Adapter via WSDL port element jca:address.
The Adapter Framework is unable to startup the Resource Adapter specified in the
 WSDL jca:address element:  {http://xmlns.oracle.com/pcbpel/wsdl/jca/}address:
 location='eis/AQ/aqSample3'
```

The reason for this is most likely that either 1) the resource adapter's RAR file has not been deployed successfully to the OC4J Application server or 2) the `location` attribute in `$J2EE_HOME/application-deployments/default/deployed-adapter-name/oc4j-ra.xml` has not been set to `eis/AQ/aqSample3`. In the last case you may have to add a new `connector-factory` entry (connection) to `oc4j-ra.xml`. Correct this and then restart the BPEL/OC4J Application Server

Solution

See the solution section for the same problem in the inbound section, as described in "Inbound Errors" on page A-18.

I/O Exception: Network Adapter Could Not Establish the Connection

```
2005-04-20 18:41:40,570> <ERROR> <default.collaxa.cube.ws>
 <AdapterFramework::Outbound>
 file:/C:/050420/integration/orabpel/domains/default/tmp/.bpel_File2AQBLOB_
1.0.jar/EnqueueBlobPayload.wsdl [ Enqueue_ptt::Enqueue(opaque) ] - Could not
 invoke operation 'Enqueue' against the 'AQ Adapter' due to: ORABPEL-12511
Adapter Framework unable to create outbound JCA connection.
file:/C:/050420/integration/orabpel/domains/default/tmp/.bpel_File2AQBLOB_
1.0.jar/EnqueueBlobPayload.wsdl [ Enqueue_ptt::Enqueue(opaque) ] - : The Adapter
 Framework was unable to establish an outbound JCA connection due to the following
 issue: ORABPEL-11929
SQL error while creating managed (database) connection.
SQL error while creating managed (database) connection: Error while trying to
 connect to database.
Error while trying to connect to database using connect string
 "jdbc:oracle:thin:@localhost:1521:appdb - java.sql.SQLException: Io exception:
 The Network Adapter could not establish the connection"
```

Solution

If the `connectionString` is not correct, do the following:

1. Change the `connectionString` in `$J2EE_HOME\application-deployments\default\AqAdapter\oc4j-ra.xml`.
2. Restart Oracle BPEL Process Manager.

If the connectionString is correct, make sure the database and listener are up. If you had enabled outbound retry, the message should be automatically retried when the database and its listener are up.

To configure outbound retry, set the retryMaxCount property and retryInterval property for the partner link in bpel.xml.

For example, the following configuration means 10 retries, at 60 second intervals.

```
<partnerLinkBinding name="EnqueueBLOBPayload">
            <property name="wsdlLocation">EnqueueBlobPayload.wsdl</property>
            <property name="retryMaxCount">10</property>
            <property name="retryInterval">60</property>
 </partnerLinkBinding>
```

Queue Not Found

```
<timestamp> <ERROR> <default.collaxa.cube.ws> <AQ Adapter::Outbound>
 oracle.AQ.AQException: JMS-190: Queue SCOTT.BLOBPAYLOAD_QUEUE not found
     at oracle.AQ.AQUtil.throwAQEx(AQUtil.java:196)
     at oracle.AQ.AQOracleSession.getQueue(AQOracleSession.java:720)
     at oracle.tip.adapter.aq.database.Queue.connect(Queue.java:102)
     at oracle.tip.adapter.aq.database.MessageWriter.init(MessageWriter.java:231)
```

Solution

Same solution as the inbound Queue not found problem. Create the queue and redeploy the process. If this process is deployed from the samples, all queue creation scripts are located in sql\create_queues.sql under each project.

Incorrect Username/Password

Sample error:

```
file:/C:/050420/integration/orabpel/domains/default/tmp/.bpel_File2AQBLOB_
1.0.jar/EnqueueBlobPayload.wsdl [ Enqueue_ptt::Enqueue(opaque) ] - : The Adapter
 Framework was unable to establish an outbound JCA connection due to the following
 issue: ORABPEL-11929
SQL error while creating managed (database) connection.
SQL error while creating managed (database) connection: Error while trying to
 connect to database.
Error while trying to connect to database using connect string
 "jdbc:oracle:thin:@localhost:1521:appdb2 - java.sql.SQLException: ORA-01017:
 invalid username/password; logon denied.
```

Solution

1. Make sure you have the correct username and password in $J2EE_HOME\application-deployments\default\AqAdapter\oc4j-ra.xml.
2. Restart Oracle BPEL Process Manager.

User Does Not Have DBMS_AQIN Privileges, Which Are Required by the AQ Java API

Sample error:

```
<timestamp> <ERROR> <default.collaxa.cube.ws> <AQ Adapter::Outbound>
 oracle.AQ.AQOracleSQLException: ORA-06550: line 1, column 7:
PLS-00201: identifier 'DBMS_AQIN' must be declared
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
    at oracle.AQ.AQOracleQueue.enqueue(AQOracleQueue.java:1267)
```

Solution

Log on to the database using `sys` as `sysdba`, `GRANT EXECUTE ON SYS.DBMS_AQIN to <username>;`. Again, if you have retry configured for this partner link, retry automatically happens.

Translation Error

Sample error:

```
<timestamp> <ERROR> <default.collaxa.cube.ws> <AQ Adapter::Outbound> ORABPEL-11101
Translation Failure.
Translation to native failed. Invalid text 'blahblah' in element: 'Root-Element'.
Check the error stack and fix the cause of the error. Contact oracle support if
 error is not fixable.
at
oracle.tip.pc.services.translation.xlators.nxsd.NXSDTranslatorImpl.translateToNati
ve(NXSDTranslatorImpl.java:502)
at
oracle.tip.adapter.aq.database.MessageWriter.translateToNative(MessageWriter.java:
1102)
at oracle.tip.adapter.aq.database.MessageWriter.doEnqueue(MessageWriter.java:494)
```

Solution

1. From Oracle BPEL Console, click the **Instances** tab.
2. Click the failed instance and select **Debug**.
3. Look for the variable passed to the invoke activity that failed. You should notice this variable match schema definition. Back track in the **Debug** window to find out why.

JDeveloper BPEL Designer Errors

I have an AQ inbound to AQ outbound end-to-end scenario. How do I copy the priority from an inbound queue to an outbound queue?

Solution: Create both the inbound header and outbound header, and use an assign activity to copy the priority.

Create the inbound header:

1. Click the **receive** activity that is linked to the AQ Inbound partner link.
2. Select the **Adapters** tab and click the **flashlight** icon to the right of the **Header Variable** field.
3. Right click **Variables** and select **Create Variable**.
4. Enter a name such as ***inbound_header***.
5. Click **Message type** and the **flashlight** icon.
6. If a `payloadHeader` is not required (most cases):

   In the Type Chooser window, click **Type Explorer** > **Message Types** > **Partner Links** > *inbound_partnerlink_name* > *inbound_service*.**wsdl** > **Imported WSDL** > **aqAdapterInboundHeader.wsdl** > **Message Types** > **Header**. Go to step 8.
7. If a `payloadHeader` *is* required (only if `PayloadHeaderRequired="true"` in the JCA operation section of *`service_name`*`.wsdl`):

   In the Type Chooser window, click **Type Explorer** > **Message Types** > **Partner Links** > inbound_partnerlink_name > **Message Types** > **Header_msg**.

8. Click **OK** to exit the Type Chooser window.
9. While the variable is still highlighted, click **OK** to pick this variable.

   The variable now appears as the header variable.
10. Click **OK** to exit the **receive** activity.

Create the outbound header:

1. Click the **invoke** activity that is linked to the AQ outbound partner link.
2. Select the **Adapters** tab and click the **flashlight** icon to the right of the **Header Variable** field.
3. Right click **Variables** and select **Create Variable**.
4. Enter a name such as ***outbound_header***.
5. Click **Message type** and click the **flashlight** icon.
6. If a `payloadHeader` is not required (most cases):

   In the Type Chooser window, click **Type Explorer** > **Message Types** > **Partner Links** > ***outbound_partnerlink_name*** > ***outbound_service*.wsdl** > **Imported WSDL** > **aqAdapterOutboundHeader.wsdl** > **Message Types** > **Header**. Go step 8.
7. If a `payloadHeader` *is* required, only if `PayloadHeaderRequired="true"` in the JCA operation section of *`service_name`*`.wsdl`:

   In the Type Chooser window, click **Type Explore**r > **Message Types** > **Partner Links** > ***outbound_partnerlink_name*** > **Message Types** > **Header_msg**.
8. Click **OK** to exit the Type Chooser window.
9. While the variable is still highlighted, click **OK** to pick this variable.
10. The variable now appears as the header variable. Click **OK** to exit the **receive** activity.

In an Assign activity:

1. Click the **Copy Rules** tab and select **Create**.
2. In the **From** section: drill down to **priority** for the **inbound header variable**.
3. In the **To** section: drill down to **priority** for the **outbound header variable**.
4. Click **OK** to exit Create Copy Rule window.
5. Click **OK** to exit the Assign window.

I redefined the adapter WSDL by stepping through the wizard again. Why doesn't my service_name.wsdl change?

Solution:

It did work, but JDeveloper BPEL Designer did not refresh the file properly. Close the *`service_name`*`.wsdl` file and open it again.

I redefined the adapter service WSDL using the wizard. But at deployment time, I got the following error:

```
Process "AQSupportedADTTypes" (revision "1.0") compilation failed.
<timestamp> <ERROR> <default.collaxa.cube.engine.deployment>
 <CubeProcessLoader::create> BPEL validation failed.
BPEL source validation failed, the errors are:
[Error ORABPEL-10007]: unresolved messageType
[Description]: in line 16 of
```

```
 "C:\050420\integration\orabpel\domains\default\tmp\.bpel_AQSupportedADTTypes_
1.0.jar\AQSupportedADTTypes.bpel", WSDL messageType
 "{http://xmlns.oracle.com/pcbpel/adapter/aq/Enqueue/}Header_msg" of variable
 "out_header" is not defined in any of the WSDL files.
[Potential fix]: Make sure the WSDL messageType
 "{http://xmlns.oracle.com/pcbpel/adapter/aq/Enqueue/}Header_msg" is defined in
 one of the WSDLs referenced by the deployment descriptor.
```

Solution:

When you redefine the adapter service WSDL, you also need to redefine the header variables. The creation of header variables the `input` element in the JCA binding section which defines the header.

For example:

```
<input>
   <jca:header message="tns:Header_msg" part="Header"/>
</input>
```

When the adapter service is redefined, the old WSDL file is overwritten. Delete the old header variables and recreate them.

Translation Error

Sample error:

```
<timestamp> <ERROR> <default.collaxa.cube.activation> <AQ Adapter::Inbound>
MessageReader_readMessage: Received TranslationException
<timestamp> <ERROR> <default.collaxa.cube.activation> <AQ Adapter::Inbound>
ORABPEL-11211
DOM Parsing Exception in translator.
DOM parsing exception in inbound XSD translator while parsing InputStream.
Check the error stack and fix the cause of the error. Contact oracle support
if error is not fixable.
at
oracle.tip.pc.services.translation.xlators.xsd.XSDTranslator.translateFromNative(X
SDTranslator.java:131)
```

Solution:

Look for the rejected message and find out why it has failed translation. The message may not be XML or the XML root element maybe incorrect, or the message may be blank. If a rejection handler has been defined for this process, look for the message in the rejection handler. Otherwise, look for the message in the default rejection handler which is located at:

ORACLE_HOME`\integration\orabpel\domains\default\archive\jca\AQMessageRejectionHandler\rejectedMessages`

For an example of how a user can define the rejection handler, look in

ORACLE_HOME`\integration\orabpel\samples\tutorials\124.AQAdapter\AQMessageRejectionHandler`

Other Problems

I have a new adapter *.RAR file; how do I redeploy the adapter?

1. Save a copy of `$J2EE_HOME\application-deployments\default\AqAdapter\oc4j-ra.xml` so you have your endpoint information.
2. Undeploy the adapter by entering the following command:

   ```
   java -jar $JE22_HOME\admin.jar ormi://localhost admin welcome
   -undeployconnector -name AqAdapter
   ```

3. Deploy the new `.rar` file by entering the following command:

   ```
   java -jar $J2EE_HOME/admin.jar ormi://localhost admin welcome
   -deployconnector -file <rarfile> -name AqAdapter
   ```

4. Modify `$J2EE_HOME\application-deployments\default\AqAdapter\oc4j-ra.xml` to add your endpoints.
5. Restart Oracle BPEL Process Manager. You should not have to redeploy your business processes.

Subscriber already exists when using MessageRuleSelector

```
<timestamp> <INFO> <default.collaxa.cube.activation> <AQ Adapter::Inbound>
Subscriber PriorityOneDequeuer already exists in the database.  If the
subscriber  does not contain the rule that you want, please undeploy the
business process,  drop the subscriber with the following sql*plus command,
and redeploy.
DECLARE
  subscriber sys.aq$_agent;
BEGIN
  subscriber := sys.aq$_agent('<subscriber_name>', NULL, NULL);
DBMS_AQADM.REMOVE_SUBSCRIBER(
  queue_name => '<queue_name>',
  subscriber => subscriber);
END;
```

Solution:

This is not a problem if the subscriber has been generated with the rule the user expects. The adapter can create new rule-based subscribers, but cannot modify existing ones. Therefore, the first time you deploy the adapter with a nonnull value for `MessageSelectorRule`, a subscriber is created if the consumer does not already exist, using the consumer as the subscriber and the `MessageSelectorRule` as the rule. This message would appear in any subsequent redeployment or restart of Oracle BPEL Process Manager.

You can determine if the rule is what you want for the subscriber by entering the following SQL command: `select name, rule, queue from AQ$ QUEUE_TABLE_NAME_R;`

```
SQL> select name, rule, queue from AQ$RuleBased_Raw_In_R;
NAME
------------------------------
RULE
--------------------------------------------------------------------------------
QUEUE
------------------------------
```

```
PRIORITYONEDEQUEUER
priority = 1
RULEBASED_RAW_IN
```

You do not have DBMS_AQIN privileges, which are required by the AQ Java API

```
2005-04-20 16:10:52,695> <ERROR> <default.collaxa.cube.activation> <AQ
Adapter::Inbound>
oracle.AQ.AQOracleSQLException: ORA-06550: line 1, column 7:
PLS-00201: identifier 'DBMS_AQIN' must be declared
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
at oracle.AQ.AQOracleQueue.dequeue(AQOracleQueue.java:1795)
at oracle.AQ.AQOracleQueue.dequeue(AQOracleQueue.java:1307)
at
oracle.tip.adapter.aq.database.MessageReader.readMessage(MessageReader.java:399)
at
oracle.tip.adapter.aq.inbound.AQActivationSpecDequeuer.run(AQActivationSpecDeq
ueuer.java:163)
at oracle.tip.adapter.fw.jca.work.WorkerJob.go(WorkerJob.java:51)
@ at oracle.tip.adapter.fw.common.ThreadPool.run(ThreadPool.java:267)
at java.lang.Thread.run(Thread.java:534)
```

Solution:

Log on to the database using `sys as sysdba 'GRANT EXECUTE ON SYS.DBMS_AQIN to` *`username`*`;'`. No deployment is necessary because this failure occurs after the connection has succeeded. The adapter automatically attempts to reconnect until the error is gone or the process is undeployed.

Failed JNDI Lookup

```
<timestamp> <WARN> <default.collaxa.cube.activation>
<AdapterFramework::Inbound> JNDI lookup of 'eis/AQ/aqSample2' failed due to:
eis/AQ/aqSample2 not found
<timestamp> <ERROR> <default.collaxa.cube.activation>
<AdapterFramework::Inbound> Error while performing endpoint Activation:
ORABPEL-12510
Unable to locate the JCA Resource Adapter via WSDL port element jca:address.
The Adapter Framework is unable to startup the Resource Adapter specified in
the WSDL jca:address element:
@ {http://xmlns.oracle.com/pcbpel/wsdl/jca/}address:
location='eis/AQ/aqSample2'
```

Solution:

The reason for this is most likely that either:

1. The resource adapter's RAR file has not been deployed successfully to the OC4J Application Server.
2. The Resource Adapters RAR file has not been deployed successfully to the `location` attribute in:

   `$J2EE_HOME/application-deployments/default/deployed-adapter-name/oc4j-ra.xml has not been set to eis/AQ/aqSample2`.

   In this case, you may have to add a new `connector-factory` entry (connection) to the `oc4j-ra.xml` file. Add the correct entry and then restart the BPEL Application Server.

How to fix:

1. Look for the file `$J2EE_HOME/application-deployments/default/aqAdapter/oc4j-ra.xml`. This file should be created when the adapter is deployed, which occurs the first time Oracle BPEL Process Manager is started. If the adapter is undeployed for some reason, deploy the adapter with the following command, then follow step 2:

   ```
   java -jar $J2EE_HOME/admin.jar ormi://localhost admin welcome
   -deployconnector -file <path to AQAdapter.rar> -name
   AqAdapter
   ```

2. If `$J2EE_HOME/application-deployments/default/aqAdapter/oc4j-ra.xml` exists, make sure the JNDI location is defined in the `oc4j-ra.xml` file.

   ```
   <connector-factory location="eis/AQ/aqSample" connector-name="AQ Adapter">
   <config-property name="connectionString"
   value="jdbc:oracle:thin:@myhost:1521:appdb2"/>
   <config-property name="userName" value="scott"/>
   @ <config-property name="password" value="tiger"/>
   </connector-factory>
   ```

Summary

This appendix describes Oracle BPEL Process Manager troubleshooting methods.

Index

A

activationAgent property
  fatalErrorFailoverProcess, 2-13
ActivationSpec parameters, 2-16
Adapter Configuration Wizard
  configuring the database adapter, 4-74
  configuring the file adapter, 4-19
  starting, 1-1, 2-3, 4-18
  stored procedures, 4-55
Adapter Configuration wizard
  understanding what happens internally during design time, 4-31
  using with the AQ adapter, 3-6
  using with the database adapter, 4-18
  using with the file adapter, 2-3
  using with the FTP adapter, 2-3
  using with the JMS adapter, 5-4
adapter services
  defined in the WSDL file, 4-38
adapters
  configuring, 1-1
  creating header variables, 3-21
  definition, 1-1
  error handling, 2-12
  in JDeveloper BPEL Designer, 1-1
  service names, 1-2
ADT payload types
  AQ adapter, 3-4
after-read strategy
  database adapter, 4-28
APIs, 4-59
application.xml file
  configuring OJMS, 5-13
  configuring Tibco JMS, 5-16
AQ adapter
  ADT payload types, 3-4
  AQ header properties, 3-3
  correlation identifier, 3-2
  dequeue mode, 3-3
  enqueue-specific features, 3-2
  features, 3-1
  generated WSDL file, 3-14
  message priority, 3-2
  multiconsumer queue, 3-2
  payload schema, 3-5
  use cases, 3-6
AQ header properties
  AQ adapter, 3-3
AQ headers, 3-20
arrayIdentifierLength
  construct, 6-43
arrayLength
  construct, 6-43
arrays
  native schema, 6-28
arrayTerminatedBy
  construct, 6-43
assign
  construct, 6-43
audit link
  following the instance execution process, 4-82

B

batch processing
  support with the file and FTP adapters, 2-7
batching
  definition, 2-10
byteOrder
  construct, 6-42

C

cellSeparatedBy
  construct, 6-43
choiceCondition
  construct, 6-43
clauses
  clauses to add to impact the sign position in COBOL Copybook, 6-5
  supported COBOL Copybook clauses, 6-3
COBOL Copybook
  clauses to add to impact the sign position, 6-5
  definition, 6-2
  multiple root levels, 6-13
  Native Format Builder wizard support, 6-2
  numeric types, 6-20
  single root level, virtual decimal, fixed length array, 6-16
  supported clauses, 6-3
  use cases, 6-13

user inputs, 6-3
variable length array, 6-18
compiling
FulfillOrder process, 4-81
complex structure
native schema, 6-10
conditional processing
native schema, 6-33
conditionValue
construct, 6-43
config timeout parameter
configuring in the server.xml file, 2-11
connection pooling
database adapter, 4-50
constructs
arrayIdentifierLength, 6-43
arrayLength, 6-43
arrayTerminatedBy, 6-43
assign, 6-43
byteOrder, 6-42
cellSeparatedBy, 6-43
choiceCondition, 6-43
conditionValue, 6-43
dateFormat, 6-43
encoding, 6-42
headerLines, 6-42
headerLinesTerminatedBy, 6-42
identifierLength, 6-43
itemSeparatedBy, 6-43
leftsurroundedBy, 6-43
length, 6-43
listTerminatedBy, 6-43
lookAhead, 6-43
native schema, 6-42
paddedBy, 6-43
padStyle, 6-43
quotedBy, 6-43
rightsurroundedBy, 6-43
skip, 6-43
skipLines, 6-43
skipUntil, 6-43
standalone, 6-42
startsWith, 6-43
stream, 6-42
style, 6-44
surroundedBy, 6-44
terminatedBy, 6-44
uniqueMessageSeparator, 6-42
variable, 6-44
variables, 6-44
version, 6-42
xmlversion, 6-42
control table
database adapter, 4-14
copy rules
creating, 4-79, 4-81
correlation identifier
AQ adapter, 3-2
creating relationships
database adapter, 4-23
CSV
native schema, 6-8

D

database adapter
after-read strategy, 4-28
choosing a polling strategy, 4-50
connection pooling, 4-50
control table, 4-14
creating a partner link, 4-74
creating a project, 4-73
creating an invoke activity, 4-77
creating relationships, 4-23
creating the object model, 4-26
datatype conversions, 4-69
defining keys, 4-22
defining the WHERE clause, 4-26
deleting the rows that were read, 4-29
deployment, 4-44
design overview, 4-2
existence checking, 4-50
features, 4-1
function return values, 4-70
generated XML schema, 4-43
importing and selecting tables, 4-21
inbound distributed polling, 4-51
last updated, 4-14
last-read ID, 4-13
locking, 4-51
logical delete, 4-11
mapping any relational schema to any XML schema types, 4-8
maxRaiseSize, 4-50
merge operations, 4-9
null values, 4-70
operation type, 4-20
performance, 4-49
physical delete, 4-10
PL/SQL Boolean support, 4-71
PL/SQL RECORD support, 4-71
polling strategies, 4-10
query by example operations, 4-9
REF CURSOR support, 4-70
relational types to XML schema types, 4-7
relational-to-XML mappings, 4-4, 4-36
relationship reading, 4-50
running the FulfillOrder process, 4-82
sequencing table, 4-13, 4-14, 4-30
SQL operations as Web services, 4-8
stored procedure design time WSDL and XSD generation, 4-61
stored procedures and functions, 4-54
stored procedures at run time, 4-67
supported primitive datatypes, 4-62
third-party database support, 4-52
TopLink mapping workbench project, 4-32
toplink_mappings.xml file, 4-3
tutorials, 4-16
update a field in the table, 4-29

use cases, 4-9, 4-16
validating, compiling, and deploying, 4-81
value binding, 4-67
database connection
configuring in the oc4j-ra.xml file, 4-3, 4-19
database operations
DML operations, 4-8
datatype conversions
database adapter, 4-69
dateFormat
construct, 6-43
dates
native schema, 6-40
DBActivationSpec, 4-41
DBInsert process
selecting for a partner link, 4-76
DBReadInteractionSpec, 4-40
DBWriteInteractionSpec, 4-39
debatching
definition, 2-3, 2-10
file adapter use case, 2-40
supported with the file and FTP adapters, 2-3
defining keys
database adapter, 4-22
deleting rows that were read
database adapter, 4-29
delimited format
Native Format Builder wizard support, 6-2
delimiters
file adapter use case, 2-40
deploying
FulfillOrder process, 4-81
deployment
of database adapter, 4-44
dequeue mode
AQ adapter, 3-3
DML operations
merge, 4-9
query by example, 4-9
with the database adapter, 4-8
domain
passwords, 4-82
DTD format
Native Format Builder wizard support, 6-2

E

encoding
construct, 6-42
enqueue-specific features
AQ adapter, 3-2
error handling
fatalErrorFailoverProcess property, 2-13
inbound direction for file and FTP adapters, 2-12
outbound direction for file and FTP adapters, 2-25
rejectedMessageHandlers property, 2-12
uniqueMessageSeparater property, 2-14
errors
default error directory, 2-15
Esc key
stopping creation of XPath expressions, 4-79, 4-81
existence checking
database adapter, 4-50

F

fatalErrorFailoverProcess property
error handling, 2-13
file adapter
ActivationSpec parameters, 2-16
architecture, 2-3
archiving successfully processed files, 2-6
batch processing, 2-7
batching multiple inbound messages, 2-10
batching multiple outbound files, 2-24
binary data format support, 2-2
Cobol Copybook format support, 2-2
debatching use case, 2-40
delimited format support, 2-2
dynamic outbound file names, 2-23
features, 2-1
file delimiter use case, 2-40
file polling, 2-10
file read concepts, 2-4
file read operation, 2-2
file reading use case, 2-39
file size delivery limitations, 2-6
file write concepts, 2-18
file writing use case, 2-40
fixed positional format support, 2-2
fixed positional use case, 2-40
guaranteed delivery and recovery, 2-15
inbound direction, 2-4
inbound header WSDL file, 2-16
inbound WSDL file, 2-15
including and excluding files, 2-8
limitations on outbound file name lengths, 2-21
logical and physical directory paths, 2-6, 2-20
opaque support, 2-2
outbound file directory, 2-19
outbound file naming conventions, 2-21
outbound header WSDL file, 2-27
outbound WSDL file, 2-26
processing large files, 2-11
supported formats, 2-1
supported naming patterns, 2-7
synchronous reads using an invoke activity, 2-17
translating native data, 2-11
use cases, 2-39
write read operation, 2-2
XML format support, 2-2
file names
limitations on file adapter outbound file name lengths, 2-21
file polling, 2-10
file reading
file adapter use case, 2-39
file writing
file adapter use case, 2-40

fixed length data
native schema, 6-21
fixed length structure
native schema, 6-9
fixed positional
file adapter use case, 2-40
fixed-length positional format
Native Format Builder wizard support, 6-2
from Oracle BPEL Console, 2-15
FTP adapter
ActivationSpec parameters, 2-16
architecture, 2-3
archiving successfully processed files, 2-6
batch processing, 2-7
batching multiple inbound messages, 2-10
batching multiple outbound files, 2-24
binary data format support, 2-1
Cobol Copybook format support, 2-1
creating an Oracle Wallet, 2-38
delimited format support, 2-1
dynamic outbound file names, 2-23
features, 2-1
file inbound file directory, 2-19
file modification times, 2-30
file polling, 2-10
file read concepts, 2-4
file read operation, 2-2
file write concepts, 2-18
file write operation, 2-2
fixed positional format support, 2-1
guaranteed delivery and recovery, 2-15
inbound direction, 2-4, 2-29
inbound header WSDL file, 2-16
inbound WSDL file, 2-15
including and excluding files, 2-8
installing and configuring OpenSSL, 2-35
installing and configuring vsftpd, 2-36
logical and physical directory paths, 2-6, 2-20
opaque support, 2-2
outbound direction, 2-33
outbound file naming conventions, 2-21
outbound header WSDL file, 2-27
outbound WSDL file, 2-26
processing large files, 2-11
restrictions on use of RESTART and RECOVERY commands, 2-28
secure FTP overview, 2-34
serverLineSeparator property for determining line separations during binary file transfers, 2-29
specifying connection information to an FTP server, 2-27
SSL, 2-34
supported formats, 2-1
supported naming patterns, 2-7
translating native data, 2-11
use cases, 2-39, 2-40
using secure FTP, 2-34
XML format support, 2-1
FTP server
specifying configuration information in the oc4j-ra.xml file, 2-28
specifying connection information to, 2-27
FulfillOrder process
deploying, 4-81
running, 4-82
function return values
database adapter, 4-70
functions
See stored procedures and functions, 4-54

H

header properties
JMS adapter, 5-3
header variables
creating for an adapter, 3-21
specifying, 2-23
headerLines
construct, 6-42
headerLinesTerminatedBy
construct, 6-42

I

IBM Websphere JMS configuration, 5-17
identifierLength
construct, 6-43
importing tables
database adapter, 4-21
inbound distributed polling
database adapter, 4-51
invoke activities
creating, 4-77
invoke activity
synchronous reads using the file adapter, 2-17
itemSeparatedBy
construct, 6-43

J

Java pattern letters
supported with file and FTP adapters, 2-21
JDeveloper BPEL Designer
adapters, 1-1
JDK regular expressions
supported with the file and FTP adapters, 2-7, 2-9
JMS adapter
concepts, 5-1
generated WSDL files, 5-10
header properties, 5-3
IBM Websphere JMS configuration, 5-17
OC4J JMS configuration, 5-15
oc4j-ra.xml file, 5-12
OJMS configuration, 5-13
point-to-point, 5-2
produce message procedure, 5-12
publish/subscribe, 5-3
Tibco JMS configuration, 5-16
use cases, 5-2

L

last updated
  database adapter, 4-14
last-read ID
  database adapter, 4-13
leftsurroundedBy
  construct, 6-43
length
  construct, 6-43
lists
  native schema, 6-26
listTerminatedBy
  construct, 6-43
LocalBPELServer
  using to deploy a process, 4-82
locking
  database adapter, 4-51
logical delete
  database adapter, 4-11
logical directory paths, 2-6, 2-20
lookAhead
  construct, 6-43

M

maxRaiseSize
  database adapter, 4-50
merge
  DML operations, 4-9
message priority
  AQ adapter, 3-2
message recovery, 2-15
messages
  file size delivery limitations with the file adapter, 2-6
multiconsumer queue
  AQ adapter, 3-2

N

namespaces
  schema must have a namespace, 2-12
naming patterns
  supported with the file and FTP adapters, 2-7
Native Format Builder wizard
  COBOL Copybook format support, 6-2
  creating native schema files, 6-1
  delimited format support, 6-2
  DTD format support, 6-2
  fixed-length positional format, 6-2
  overview of windows, 6-6
  starting, 2-12
  supported formats, 6-2
native schema files
  creating, 6-1
native schemas
  arrayIdentifierLength construct, 6-43
  arrayLength construct, 6-43
  arrays, 6-28
  arrayTerminatedBy construct, 6-43
  assign construct, 6-43
  byteOrder construct, 6-42
  cellSeparatedBy construct, 6-43
  choiceCondition construct, 6-43
  complex structure, 6-10
  conditional processing, 6-33
  conditionValue construct, 6-43
  constructs, 6-42
  CSV, 6-8
  dateFormat construct, 6-43
  dates, 6-40
  encoding construct, 6-42
  fixed length data, 6-21
  fixed length structure, 6-9
  headerLines construct, 6-42
  headerLinesTerminatedBy construct, 6-42
  identifierLength construct, 6-43
  itemSeparatedBy construct, 6-43
  leftsurroundedBy construct, 6-43
  length construct, 6-43
  lists, 6-26
  listTerminatedBy construct, 6-43
  lookAheadconstruct, 6-43
  paddedBy construct, 6-43
  padStyle construct, 6-43
  quotedBy construct, 6-43
  rightsurroundedBy construct, 6-43
  separated value file structure, 6-9
  skip construct, 6-43
  skipLines construct, 6-43
  skipUntil construct, 6-43
  standalone construct, 6-42
  startsWith construct, 6-43
  stream construct, 6-42
  style construct, 6-44
  surrounded data, 6-25
  surroundedBy construct, 6-44
  terminated data, 6-24
  terminatedBy construct, 6-44
  understanding, 6-7
  uniqueMessageSeparator construct, 6-42
  use cases, 6-7
  variable construct, 6-44
  variables, 6-41
  variables construct, 6-44
  version construct, 6-42
  xmlversion construct, 6-42
null values
  database adapter, 4-70

O

object models
  database adapter, 4-26
object type inheritance, 4-66
OC4J JMS configuration, 5-15
oc4j-ra.xml file
  configuring a database connection in, 4-3, 4-19
  configuring an FTP server connection in, 2-28
  JMS connection factory definitions, 5-12

location of, 2-28, 4-46
OJMS configuration, 5-13
opaque format
supported with file and FTP adapters, 2-2
OpenSSL
installing and configuring, 2-35
Oracle Applications adapter, 1-2
Oracle BPEL Console
accessing, 4-82
manually performing message recovery, 2-15
running a process, 4-82
Oracle Wallet
installing and configuring, 2-38
OracleAS Adapter for AQ
See AQ adapter, 3-1
OracleAS Adapter for Files
See file adapter, 2-1
OracleAS Adapter for FTP
See FTP adapter, 2-1
OracleAS Adapter for JMS
See JMS adapter, 5-1
overloading, 4-59

P

paddedBy
construct, 6-43
padStyle
construct, 6-43
partner links
creating, 4-74
passwords
domain, 4-82
payload schema
AQ adapter, 3-5
performance
database adapter, 4-49
physical delete
database adapter, 4-10
PL/SQL Boolean support
database adapter, 4-71
PL/SQL RECORD support
database adapter, 4-71
point-to-point
JMS adapter, 5-2
polling strategies
database adapter, 4-10
polling strategy
database adapter, 4-50
primitive datatypes
supported, 4-62
procedures
See stored procedures and functions, 4-54
processing large files
with the file adapter, 2-11
produce message procedure
JMS adapter, 5-12
projects
creating, 4-73
publish/subscribe
JMS adapter, 5-3

Q

query by example
DML operations, 4-9
quotedBy
construct, 6-43

R

recovery
of messages from Oracle BPEL Console, 2-15
REF CURSOR support
database adapter, 4-70
regex constructs
supported, 2-7, 2-9
rejectedMessageHandlers property
error handling, 2-12
relational-to-XML mappings, 4-36
for database adapter, 4-4
relationship reading
database adapter, 4-50
rightsurroundedBy
construct, 6-43
running
FulfillOrder, 4-82

S

schemas
must have a namespace, 2-12
searching for, 4-56
secure FTP
using with the FTP adapter, 2-34
separated value file structure
native schema, 6-9
sequencing table
database adapter, 4-13, 4-14
sequencing table updates
database adapter, 4-30
serverLineSeparator property
determines line separations during binary file transfers, 2-29
server.xml file
location of, 2-11
processing large files, 2-11
service names
creating, 2-3
in adapters, 1-2
skip
construct, 6-43
skipLines
construct, 6-43
skipUntil
construct, 6-43
SSL
creating an Oracle Wallet, 2-38
installing and configuring OpenSSL, 2-35
installing and configuring vsftpd, 2-36
secure FTP overview, 2-34

using secure FTP with the FTP adapter, 2-34
standalone
construct, 6-42
startsWith
construct, 6-43
stored procedures
creating a stored procedure use case, 4-71
design time WSDL and XSD generation, 4-61
invocation at run time, 4-67, 4-69
overloaded procedures, 4-59
run time, 4-67
stored procedures and functions, 4-56
database adapter, 4-54
viewing details, 4-59
stream
construct, 6-42
style
construct, 6-44
surrounded data
native schema, 6-25
surroundedBy
construct, 6-44
synchronous reads
using the file adapter in an invoke activity, 2-17

T

terminated data
native schema, 6-24
terminatedBy
construct, 6-44
third-party database support
database adapter, 4-52
Tibco JMS configuration, 5-16
TopLink mapping workbench project, 4-32
toplink_mappings.xml file, 4-3, 4-36
translation
native data, 2-25
not required, 2-12
of native data, 2-11
requires native schemas, 6-1
translator
converts native data to XML and back, 2-2
troubleshooting and workarounds
AQ adapter, A-18
database adapter, A-1
file adapter, A-16
FTP adapter, A-16
using stored procedures with the database adapter, A-13

U

uniqueMessageSeparater property
error handling, 2-14
uniqueMessageSeparator
construct, 6-42
updating a field in a table
database adapter, 4-29
use cases
creating a stored procedure, 4-71
using the AQ adapter, 3-6
using the database adapter, 4-16
using the File and FTP adapters, 2-39
using the JMS adapter, 5-2
using the Native Format Builder, 6-7
user-defined types, 4-64, 4-66

V

validating
FulfillOrder process, 4-81
value binding
database adapter, 4-67
variable
construct, 6-44
variables
automatically creating, 4-77, 4-78
construct, 6-44
native schema, 6-41
version
construct, 6-42
vsftpd server
installing and configuring, 2-36

W

Web services
SQL operations as, 4-8
WHERE clause
database adapter, 4-26
WSDL file
inbound direction for file and FTP adapters, 2-15
outbound direction for file and FTP adapters, 2-26
selecting for a partner link, 4-76
WSDL files
AQ adapter, 3-14
define the adapter service, 4-38
JMS adapter, 5-10
specifying logical directory paths, 2-6, 2-20
WSIL browser
selecting a service, 4-76

X

XML schema
generated by Adapter Configuration wizard, 4-43
xmlversion
construct, 6-42
XPath expressions
creating, 4-79, 4-81
XSD attributes, 4-63