Oracle® Fusion Middleware

User's Guide for Technology Adapters

11g Release 1 (11.1.1.6.2)

E10231-10

March 2012

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

ORACLE

Oracle Fusion Middleware User's Guide for Technology Adapters, 11g Release 1 (11.1.1.6.2)

E10231-10

Copyright © 2007, 2012 Oracle and/or its affiliates. All rights reserved.

Primary Author: Bob May

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

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

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

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

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

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

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

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

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Contents iii

Preface xxi

Part I Introduction and Concepts

1 Introduction to Oracle JCA Adapters

1.1 Features of Oracle JCA Adapters 1-1
1.2 Types of Oracle JCA Adapters 1-2
1.2.1 Oracle Technology Adapters 1-3
1.2.1.1 Architecture 1-4
1.2.1.2 Design-Time Components 1-5
1.2.1.3 Run-Time Components 1-7
1.2.1.4 Deployment 1-7
1.2.2 Legacy Adapters 1-8
1.2.2.1 Architecture 1-8
1.2.2.2 Design-Time Components 1-10
1.2.2.3 Run-Time Components 1-12
1.2.2.4 Deployment 1-12
1.2.3 Packaged-Application Adapters 1-12
1.2.3.1 Architecture 1-13
1.2.3.2 Design-Time Components 1-15
1.2.3.3 Run-Time Components 1-16
1.2.3.4 Deployment 1-17
1.2.4 Oracle Adapter for Oracle Applications 1-17

1.3 Types of Oracle JCA Adapters Adapter Services ... 1-17
1.3.1 Request-Response (Outbound Interaction) Service ... 1-18
1.3.2 Event Notification (Inbound Interaction) Service ... 1-18
1.3.3 Metadata Service ... 1-19

2 ADAPTER Life-Cycle Management

2.1 Installing Oracle JCA Adapters ... 2-2
2.2 Starting and Stopping Oracle JCA Adapters ... 2-2
2.3 Defining Adapter Interface by Importing an Existing WSDL ... 2-2
2.3.1 Adapter Configuration Wizard for Oracle MQ Series Adapter, Oracle JMS Adapter and the Oracle AQ Adapter 2-4
2.3.1.1 Example of Use of Callbacks ... 2-4
2.4 Configuring Message Header Properties for Oracle JCA Adapters ... 2-5
2.5 Physically Deploying Oracle JCA Adapters ... 2-6
2.5.1 The RAR Deployment Descriptor File and the weblogic-ra.xml Template File ... 2-6
2.6 Creating an Application Server Connection for Oracle JCA Adapters ... 2-7
2.7 Deploying Oracle JCA Adapter Applications from JDeveloper ... 2-10
2.7.1 Deploying an Application Profile for the SOA Project and the Application ... 2-10
2.8 Manually Deploying an Adapter RAR File that Does Not Have a Jar File Associated With It 2-13
2.8.1 Example of Manual Deployment ... 2-13
2.9 Handling the Deployment Plan When Working on a Remote Oracle SOA Server ... 2-14
2.10 Migrating Repositories from Different Environments ... 2-14
2.11 How Oracle JCA Adapters Ensure No Message Loss ... 2-15
2.11.1 XA Transaction Support ... 2-15
2.11.2 Local Transactions and Global (XA) Transactions ... 2-15
2.11.2.1 Adapter Support of Local Transactions ... 2-16
2.11.2.2 Adapter Support of Global Transactions ... 2-16
2.11.3 Basic Concepts of Transactions and Adapters ... 2-16
2.11.3.1 Asynchronous Transaction Flow ... 2-17
2.11.3.2 Synchronous Transaction Flow ... 2-18
2.11.4 Inbound Transactions ... 2-19
2.11.5 Outbound Transactions ... 2-19
2.12 Composite Availability and Inbound Adapters ... 2-20
2.13 Singleton (Active/Passive) Inbound Endpoint Lifecycle Support Within Adapters ... 2-20
2.13.1 Multiple Activations of the Same Adapter Endpoint ... 2-21
2.13.2 Hot-Standby State ... 2-21
2.14 Correlation Support Within Adapters ... 2-21
2.14.1 CorrelationID of Receive Message Not Matching Invoke: Log Error Message ... 2-22
2.14.1.1 Rejecting Nonmatching Native Correlation IDs ... 2-22
2.15 Setting Payload Size Threshold ... 2-23
2.15.1 Payload Native Size ... 2-23
2.15.1.1 Setting the Payload Threshold ... 2-23
2.15.1.2 Limitations on Payload Size Enforcement ... 2-23
2.16 Streaming Large Payload ... 2-24
2.17 Batching and Debatching Support ... 2-25
2.18 Adding an Adapter Connection Factory ... 2-25

2.18.1 Creating a Data Source ... 2-25
2.18.2 Creating a Connection Pool ... 2-26
2.19 Adding or Updating an Adapter Connection Factory ... 2-28
2.19.1 **Modify the JCA File** ... 2-28
2.19.2 Use a Config Plan ... 2-28
2.19.3 Use the Web Logic Server Console to Create a New Connection ... 2-29
2.20 Recommended Setting for Data Sources Used by Oracle JCA Adapters ... 2-29
2.21 Error Handling ... 2-31
2.21.1 Handling Rejected Messages ... 2-31
2.21.1.1 Configuring Rejection Handlers ... 2-31
2.21.1.2 Checking for Rejected Messages ... 2-32
2.21.2 Inbound Interaction Error Handling ... 2-34
2.21.2.1 Message Error Rejection Handlers ... 2-34
2.21.2.2 Inbound Retryable Errors ... 2-36
2.21.2.3 Inbound Non-Retryable Errors ... 2-39
2.21.3 Outbound Adapter Interaction Error Handling ... 2-40
2.21.3.1 Retryable Errors for Outbound Adapter Error Handling ... 2-40
2.21.3.2 Non-Retryable Errors for Outbound Interaction Handling ... 2-40
2.22 Testing Applications ... 2-42
2.23 Setting the Trace Level of Oracle JCA Adapters ... 2-43
2.23.1 How to Set the Trace Level of Oracle JCA Adapters ... 2-43
2.24 Viewing Adapter Logs ... 2-44
2.25 Creating a Custom Adapter ... 2-45
2.25.1 Configuring a Custom Adapter ... 2-45
2.25.1.1 Custom Adapter Screen Flow ... 2-46
2.25.2 Frequently Asked Questions about Adapters ... 2-53
2.25.2.1 Why are My Applications Timing Out? ... 2-53
2.25.2.2 How do Transactional and Non-Transactional Adapters Differ? ... 2-53
2.25.2.3 What Happened to My Application's Rejected Messages? Can I do Anything With Them? 2-54

3 Adapter Integration with Oracle Application Server Components

3.1 Adapter Integration with Oracle WebLogic Server ... 3-1
3.1.1 Oracle WebLogic Server Overview ... 3-1
3.1.2 Oracle WebLogic Server Integration with Adapters ... 3-3
3.1.2.1 Design Time ... 3-3
3.1.2.2 Run Time ... 3-3
3.2 Adapter Integration with Oracle Fusion Middleware ... 3-3
3.2.1 Oracle BPEL Process Manager Overview ... 3-4
3.2.2 Oracle Mediator Overview ... 3-5
3.2.3 Oracle Fusion Middleware Integration with Adapters ... 3-5
3.2.3.1 Design Time ... 3-5
3.2.3.2 Run Time ... 3-7
3.2.3.3 End-to-End Testing ... 3-7
3.2.3.4 Oracle BPEL PM Integration with Outbound Interaction ... 3-7
3.2.3.5 Oracle BPEL PM Integration with Inbound Interaction ... 3-8
3.2.3.6 Use Case: Integration with Oracle BPEL Process Manager ... 3-10

3.2.4 Oracle SOA Composite Integration with Adapters ... 3-13
3.2.4.1 Oracle SOA Composite Overview ... 3-13
3.2.4.2 Adapters Integration With Oracle SOA Composite ... 3-14
3.3 Monitoring Oracle JCA Adapters ... 3-16

4 Oracle JCA Adapter for Files/FTP

4.1 Introduction to Oracle File and FTP Adapters ... 4-1
4.1.1 Oracle File and FTP Adapters Architecture ... 4-1
4.1.2 Oracle File and FTP Adapters Integration with Oracle BPEL PM ... 4-2
4.1.3 Oracle File and FTP Adapters Integration with Mediator ... 4-4
4.1.4 Oracle File and FTP Adapters Integration with SOA Composite ... 4-5
4.2 Oracle File and FTP Adapters Features ... 4-5
4.2.1 File Formats ... 4-6
4.2.2 FTP Servers ... 4-6
4.2.3 Inbound and Outbound Interactions ... 4-6
4.2.4 File Debatching ... 4-7
4.2.5 File ChunkedRead ... 4-7
4.2.6 File Sorting ... 4-8
4.2.7 Dynamic Outbound Directory and File Name Specification ... 4-8
4.2.8 Security ... 4-8
4.2.9 Nontransactional ... 4-8
4.2.10 Proxy Support ... 4-8
4.2.11 No Payload Support ... 4-9
4.2.12 Large Payload Support ... 4-9
4.2.13 File-Based Triggers ... 4-10
4.2.14 Pre-Processing and Post-Processing of Files ... 4-10
4.2.14.1 Mechanism For Pre-Processing and Post-Processing of Files ... 4-10
4.2.14.2 Configuring a Pipeline ... 4-11
4.2.14.3 Using a Re-Entrant Valve For Processing Zip Files ... 4-19
4.2.14.4 Configuring Batch Notification Handler ... 4-19
4.2.15 Error Handling ... 4-21
4.2.15.1 Sending a Malformed XML File to a Local Filesystem Folder ... 4-22
4.2.16 Threading Model ... 4-22
4.2.16.1 Default Threading Model ... 4-22
4.2.16.2 Modified Threading Model ... 4-23
4.2.17 Performance Tuning ... 4-24
4.2.18 High Availability ... 4-24
4.2.19 Multiple Directories ... 4-24
4.2.20 Append Mode ... 4-25
4.2.21 Recursive Processing of Files Within Directories in Oracle FTP Adapter ... 4-26
4.2.22 Securing Enterprise Information System Credentials ... 4-29
4.3 Oracle File and FTP Adapter Concepts ... 4-37
4.3.1 Oracle File Adapter Read File Concepts ... 4-37
4.3.1.1 Inbound Operation ... 4-37
4.3.1.2 Inbound File Directory Specifications ... 4-38
4.3.1.3 File Matching and Batch Processing ... 4-40
4.3.1.4 File Polling ... 4-45

4.3.1.5 Postprocessing ... 4-47
4.3.1.6 Native Data Translation ... 4-48
4.3.1.7 Inbound Service ... 4-49
4.3.1.8 Inbound Headers ... 4-49
4.3.2 Oracle File Adapter Write File Concepts ... 4-50
4.3.2.1 Outbound Operation ... 4-51
4.3.2.2 Outbound File Directory Creation ... 4-52
4.3.2.3 Native Data Translation ... 4-62
4.3.2.4 Outbound Service Files ... 4-63
4.3.2.5 Outbound Headers ... 4-63
4.3.3 Oracle File Adapter Synchronous Read Concepts ... 4-64
4.3.4 Oracle File Adapter File Listing Concepts ... 4-65
4.3.4.1 Listing Operation ... 4-66
4.3.4.2 File Directory Specifications ... 4-66
4.3.4.3 File Matching ... 4-68
4.3.5 Oracle FTP Adapter Get File Concepts ... 4-70
4.3.6 Oracle FTP Adapter Put File Concepts ... 4-75
4.3.7 Oracle FTP Adapter Synchronous Get File Concepts ... 4-76
4.3.8 Oracle FTP Adapter File Listing Concepts ... 4-77
4.4 Configuring Oracle File and FTP Adapters ... 4-78
4.4.1 Configuring the Credentials for Accessing a Remote FTP Server ... 4-78
4.4.2 Configuring Oracle File and FTP Adapters for High Availability ... 4-78
4.4.2.1 Prerequisites for High Availability ... 4-78
4.4.2.2 High Availability in Inbound Operations ... 4-79
4.4.2.3 High Availability in Outbound Operations ... 4-82
4.4.3 Using Secure FTP with the Oracle FTP Adapter ... 4-84
4.4.3.1 Secure FTP Overview ... 4-84
4.4.3.2 Installing and Configuring FTP Over SSL on Solaris and Linux ... 4-85
4.4.3.3 Installing and Configuring FTP Over SSL on Windows ... 4-89
4.4.4 Using SFTP with Oracle FTP Adapter ... 4-93
4.4.4.1 SFTP Overview ... 4-93
4.4.4.2 Install and Configure OpenSSH for Windows ... 4-94
4.4.4.3 Set Up Oracle FTP Adapter for SFTP ... 4-95
4.4.5 Configuring Oracle FTP Adapter for HTTP Proxy ... 4-100
4.4.5.1 Configuring for Plain FTP Mode ... 4-100
4.4.5.2 Configuring for SFTP Mode ... 4-103
4.5 Oracle File and FTP Adapters Use Cases ... 4-104
4.5.1 Oracle File Adapter XML Debatching ... 4-104
4.5.1.1 Prerequisites ... 4-105
4.5.1.2 Designing the SOA Composite ... 4-105
4.5.1.3 Creating the Inbound Oracle File Adapter Service ... 4-108
4.5.1.4 Creating the Outbound File Adapter Service ... 4-113
4.5.1.5 Wiring Services and Activities ... 4-116
4.5.1.6 Deploying with JDeveloper ... 4-126
4.5.1.7 Monitoring Using Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console) 4-126
4.5.2 Flat Structure for Oracle BPEL PM ... 4-127
4.5.2.1 Prerequisites ... 4-127

4.5.2.2 Designing the SOA Composite ... 4-127
4.5.2.3 Creating the Inbound Oracle File Adapter Service ... 4-128
4.5.2.4 Creating the Outbound Oracle File Adapter Service ... 4-129
4.5.2.5 Wiring Services and Activities ... 4-131
4.5.2.6 Deploying with JDeveloper ... 4-134
4.5.2.7 Monitoring Using Oracle Fusion Middleware Control Console ... 4-134
4.5.3 Flat Structure for Mediator ... 4-135
4.5.3.1 Prerequisites ... 4-135
4.5.3.2 Creating a Mediator Application and Project ... 4-135
4.5.3.3 Importing the Schema Definition (.XSD) Files ... 4-136
4.5.3.4 Creating the Inbound Oracle File Adapter Service ... 4-136
4.5.3.5 Creating the Outbound Oracle FTP Adapter Service ... 4-137
4.5.3.6 Wiring Services ... 4-137
4.5.3.7 Creating the Routing Rule ... 4-138
4.5.3.8 Deploying with JDeveloper ... 4-140
4.5.3.9 Run-Time Task ... 4-140
4.5.4 Oracle File Adapter Scalable DOM ... 4-140
4.5.4.1 Prerequisites ... 4-140
4.5.4.2 Designing the SOA Composite ... 4-140
4.5.4.3 Creating the Inbound Oracle File Adapter Service ... 4-141
4.5.4.4 Creating the Outbound Oracle File Adapter Service ... 4-143
4.5.4.5 Wiring Services and Activities ... 4-144
4.5.4.6 Deploying with JDeveloper ... 4-151
4.5.4.7 Monitoring Using Fusion Middleware Control Console ... 4-151
4.5.5 Oracle File Adapter ChunkedRead ... 4-151
4.5.5.1 Prerequisites ... 4-152
4.5.5.2 Designing the SOA Composite ... 4-152
4.5.5.3 Creating the Inbound Oracle File Adapter Service ... 4-153
4.5.5.4 Creating the Outbound Oracle File Adapter Service ... 4-154
4.5.5.5 Wiring Services and Activities ... 4-157
4.5.5.6 Deploying with JDeveloper ... 4-168
4.5.5.7 Monitoring Using Fusion Middleware Control Console ... 4-168
4.5.6 Oracle File Adapter Read File As Attachments ... 4-169
4.5.6.1 Prerequisites ... 4-169
4.5.6.2 Designing the SOA Composite ... 4-169
4.5.6.3 Creating the Inbound Oracle File Adapter Service ... 4-170
4.5.6.4 Creating the Outbound Oracle File Adapter Service ... 4-172
4.5.6.5 Wiring Services and Activities ... 4-174
4.5.6.6 Deploying with JDeveloper ... 4-181
4.5.6.7 Monitoring Using Fusion Middleware Control Console ... 4-181
4.5.7 Oracle File Adapter File Listing ... 4-181
4.5.7.1 Prerequisites ... 4-182
4.5.7.2 Designing the SOA Composite ... 4-182
4.5.7.3 Creating the Outbound Oracle File Adapter Service ... 4-183
4.5.7.4 Wiring Services and Activities ... 4-184
4.5.7.5 Deploying with JDeveloper ... 4-189
4.5.7.6 Monitoring Using Fusion Middleware Control Console ... 4-189

4.5.8 Oracle File Adapter Complex Structure 4-189
4.5.8.1 Prerequisites 4-190
4.5.8.2 Designing the SOA Composite 4-190
4.5.8.3 Creating the Inbound Oracle File Adapter Service 4-191
4.5.8.4 Creating the Outbound Oracle File Adapter Service 4-192
4.5.8.5 Wiring Services and Activities 4-193
4.5.8.6 Deploying with JDeveloper 4-196
4.5.8.7 Monitoring Using Fusion Middleware Control Console 4-196
4.5.9 Oracle FTP Adapter Debatching 4-197
4.5.9.1 Prerequisites 4-197
4.5.9.2 Designing the SOA Composite 4-197
4.5.9.3 Creating the Inbound Oracle FTP Adapter Service 4-198
4.5.9.4 Creating the Outbound Oracle FTP Adapter Service 4-201
4.5.9.5 Wiring Services and Activities 4-202
4.5.9.6 Deploying with JDeveloper 4-206
4.5.9.7 Monitoring Using Fusion Middleware Control Console 4-206
4.5.10 Oracle FTP Adapter Dynamic Synchronous Read 4-207
4.5.10.1 Prerequisites 4-207
4.5.10.2 Designing the SOA Composite 4-208
4.5.10.3 Creating the Inbound Oracle File Adapter Service 4-209
4.5.10.4 Creating the Outbound Oracle FTP Adapter Service 4-210
4.5.10.5 Wiring Services and Activities 4-212
4.5.10.6 Deploying with JDeveloper 4-218
4.5.10.7 Monitoring Using Fusion Middleware Control Console 4-218
4.5.11 Copying, Moving, and Deleting Files 4-219
4.5.11.1 Moving a File from a Local Directory on the File System to Another Local Directory 4-219
4.5.11.2 Copying a File from a Local Directory on the File System to Another Local Directory 4-222
4.5.11.3 Deleting a File from a Local File System Directory 4-222
4.5.11.4 Using a Large CSV Source File 4-223
4.5.11.5 Moving a File from One Remote Directory to Another Remote Directory on the Same FTP Server 4-224
4.5.11.6 Moving a File from a Local Directory on the File System to a Remote Directory on the FTP Server 4-227
4.5.11.7 Moving a File from a Remote Directory on the FTP Server to a Local Directory on the File System 4-228
4.5.11.8 Moving a File from One FTP Server to another FTP Server 4-229
4.5.12 Creating a Synchronous BPEL Composite using File Adapter 4-229
4.5.12.1 Changing the connection factory JNDI dynamically in Ftp Adapter 4-234
4.5.12.2 Retrieving the Details of the File from an Outbound Write Operation 4-236
4.5.13 Changing the Sequencing Strategy for FILE/Ftp Adapter 4-239
4.5.14 Creating a Synchronous BPEL Composite using the File Adapter 4-241

5 Oracle JCA Adapter for Sockets

5.1 Introduction to Oracle Socket Adapter 5-1
5.1.1 Oracle Socket Adapter Architecture 5-1
5.1.2 Oracle Socket Adapter Integration with Mediator 5-2

5.1.3 Oracle Socket Adapter Integration with Oracle BPEL PM 5-2
5.1.4 Oracle Socket Adapter Integration with SOA Composite 5-4
5.2 Oracle Socket Adapter Features 5-4
5.3 Oracle Socket Adapter Concepts 5-4
5.3.1 Communication Modes 5-4
5.3.1.1 Inbound Synchronous Request/Response 5-5
5.3.1.2 Outbound Synchronous Request/Response 5-5
5.3.1.3 Inbound Receive 5-6
5.3.1.4 Outbound Invoke 5-6
5.3.2 Mechanisms for Defining Protocols 5-6
5.3.2.1 Protocol with Handshake Mechanism Using Style Sheet 5-6
5.3.2.2 Protocol with Handshake Mechanism Using Custom Java Code 5-9
5.3.2.3 Protocol Without Handshake Mechanism 5-11
5.3.3 Character Encoding and Byte Order 5-12
5.3.4 Performance Tuning 5-13
5.3.4.1 Configuring Oracle Socket Adapter Connection Pooling 5-13
5.4 Configuring Oracle Socket Adapter 5-17
5.4.1 Modifying the weblogic-ra.xml File 5-17
5.4.2 Modeling a Handshake 5-19
5.4.2.1 Modeling an Outbound Handshake 5-19
5.4.2.2 Modeling an Inbound Handshake 5-19
5.4.3 Designing an XSL File Using the XSL Mapper Tool 5-20
5.4.3.1 Designing XSL for Inbound Synchronous Request/Reply 5-20
5.4.3.2 Designing XSL for Outbound Synchronous Request/Reply 5-31
5.4.4 Specifying a TCP Port in a Configuration Plan For an Oracle Socket Adapter 5-40
5.5 Oracle Socket Adapter Use Cases 5-41
5.5.1 Oracle Socket Adapter Hello World 5-41
5.5.1.1 Prerequisites 5-41
5.5.1.2 Designing the SOA Composite 5-42
5.5.1.3 Creating the Inbound Oracle Socket Adapter Service 5-45
5.5.1.4 Creating the Outbound Oracle Socket Adapter Service 5-51
5.5.1.5 Wiring Services and Activities 5-54
5.5.1.6 Deploying with JDeveloper 5-67
5.5.1.7 Monitoring Using the Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console) 5-67
5.5.2 Flight Information Display System 5-68
5.5.2.1 Prerequisites 5-68
5.5.2.2 Designing the SOA Composite 5-68
5.5.2.3 Creating the Inbound Oracle Socket Adapter Service 5-70
5.5.2.4 Creating Outbound Oracle Socket Adapter Services 5-74
5.5.2.5 Wiring Services and Activities 5-80
5.5.2.6 Deploying with JDeveloper 5-106
5.5.2.7 Monitoring Using the Fusion Middleware Control Console 5-106

6 Native Format Builder Wizard

6.1 Creating Native Schema Files with the Native Format Builder Wizard 6-1
6.1.1 Supported File Formats 6-2

6.1.1.1 Delimited ... 6-3
6.1.1.2 Fixed Length (Positional) ... 6-4
6.1.1.3 Complex Type ... 6-4
6.1.1.4 DTD ... 6-4
6.1.1.5 COBOL Copybook ... 6-4
6.1.2 Editing Native Schema Files ... 6-8
6.2 Native Schema Constructs ... 6-9
6.2.1 Understanding Native Schema Constructs ... 6-9
6.2.2 Using Native Schema Constructs ... 6-11
6.2.2.1 Defining Fixed-Length Data ... 6-12
6.2.2.2 Defining Terminated Data ... 6-15
6.2.2.3 Defining Surrounded Data ... 6-17
6.2.2.4 Defining Lists ... 6-19
6.2.2.5 Defining Arrays ... 6-20
6.2.2.6 Conditional Processing ... 6-26
6.2.2.7 Defining Dates ... 6-34
6.2.2.8 Using Variables ... 6-36
6.2.2.9 Defining Prefixes and Suffixes ... 6-37
6.2.2.10 Defining Skipping Data ... 6-38
6.2.2.11 Defining fixed and default Values ... 6-39
6.2.2.12 Defining write ... 6-40
6.2.2.13 Defining LookAhead ... 6-41
6.2.2.14 Defining outboundHeader ... 6-43
6.2.2.15 Defining Complex Condition in conditionValue ... 6-44
6.2.2.16 Defining Complex Condition in choiceCondition ... 6-46
6.2.2.17 Defining dataLines ... 6-46
6.2.2.18 Defining Date Formats with Time Zone ... 6-47
6.2.2.19 Implementing Validation During Translation ... 6-49
6.2.2.20 Processing Files with BOM ... 6-51
6.3 Translator XPath Functions ... 6-51
6.3.1 Terminologies ... 6-52
6.3.2 Translator XPath Functions ... 6-52
6.3.2.1 doTranslateFromNative Function ... 6-52
6.3.2.2 doTranslateToNative Function ... 6-57
6.3.2.3 doStreamingTranslate Function ... 6-59
6.3.2.4 Batching Transformation Features ... 6-61
6.4 Use Cases for the Native Format Builder ... 6-63
6.4.1 Defining the Schema for a Delimited File Structure ... 6-63
6.4.1.1 Defining a Asterisk (*) Separated Value File Structure ... 6-75
6.4.2 Defining the Schema for a Fixed Length File Structure ... 6-76
6.4.3 Defining the Schema for a Complex File Structure ... 6-83
6.4.4 Removing or Adding Namespaces to XML with No Namespace ... 6-100
6.4.5 Defining the Choice Condition Schema for a Complex File Structure ... 6-101
6.4.6 Defining Choice Condition With LookAhead for a Complex File Structure ... 6-107
6.4.7 Defining Array Type Schema for a Complex File Structure ... 6-113
6.4.8 Defining the Schema for a DTD File Structure ... 6-118
6.4.9 Defining the Schema for a COBOL Copybook File Structure ... 6-121

6.5 Command Line Tool for Testing NXSD Translator 6-134
6.5.1 Prerequisites 6-134
6.5.2 Running the Test Tool 6-134

Part II Message Adapters

7 Oracle JCA Adapter for AQ

7.1 Introduction to the Oracle AQ Adapter 7-1
7.1.1 Oracle AQ Adapter Integration with Oracle BPEL Process Manager 7-1
7.1.2 Oracle AQ Adapter Integration with Oracle Mediator 7-2
7.2 Oracle AQ Adapter Features 7-2
7.2.1 Enqueue-Specific Features (Message Production) 7-3
7.2.2 Dequeue and Enqueue Features 7-4
7.2.3 Supported ADT Payload Types 7-6
7.2.4 Native Format Builder Wizard 7-7
7.2.5 Normalized Message Support 7-7
7.2.6 Is DOM 2 Compliant 7-8
7.2.7 Is Message-Size Aware 7-8
7.2.8 Multiple Receiver Threads 7-8
7.2.9 DequeueTimeout Property 7-9
7.2.10 Control Dequeue Timeout and Multiple Inbound Polling Threads 7-9
7.2.11 Stream Payload Support 7-9
7.2.12 Oracle AQ Adapter Inbound Retries 7-9
7.2.13 Error Handling Support 7-10
7.2.14 Performance Tuning 7-10
7.3 Deployment 7-10
7.4 Oracle AQ Adapter Use Cases 7-12
7.4.1 Generic Use Case 7-12
7.4.1.1 The Adapter Configuration Wizard Walkthrough 7-12
7.4.1.2 Dequeuing and Enqueuing Object and ADT Payloads 7-30
7.4.1.3 Dequeuing One Column of the Object Payload 7-30
7.4.1.4 Configuring the Enqueue/Dequeue Operation Type 7-32
7.4.1.5 Using Correlation ID for Filtering Messages During Dequeue 7-47
7.4.1.6 Enqueuing and Dequeuing from Multisubscriber Queues 7-47
7.4.2 Oracle AQ Adapter ADT Queue 7-48
7.4.2.1 Meeting Prerequisites 7-49
7.4.2.2 Creating an Application and an SOA Project 7-49
7.4.2.3 Creating an Inbound Oracle AQ Adapter 7-50
7.4.2.4 Creating an Outbound Oracle AQ Adapter 7-53
7.4.2.5 Wiring Services and Activities 7-56
7.4.2.6 Configuring Routing Service 7-56
7.4.2.7 Configuring the Data Sources in the Oracle WebLogic Server Administration Console 7-59
7.4.2.8 Deploying with JDeveloper 7-64
7.4.2.9 Monitoring Using the Fusion Middleware Control Console 7-65
7.4.3 Oracle AQ Adapter RAW Queue 7-65
7.4.3.1 Prerequisites 7-65

7.4.3.2 Creating an Application and an SOA Project ... 7-66
7.4.3.3 Creating an Inbound Adapter Service ... 7-66
7.4.3.4 Creating an Outbound Adapter Service ... 7-70
7.4.3.5 Wiring Services and Activities ... 7-72
7.4.3.6 Configuring the Data Sources in the Oracle WebLogic Server Administration Console 7-77
7.4.3.7 Deploying with JDeveloper ... 7-78
7.4.3.8 Monitoring Using the Fusion Middleware Control Console ... 7-78

8 Oracle JCA Adapter for JMS

8.1 Introduction to the Oracle JMS Adapter ... 8-1
8.1.1 Oracle JMS Adapter Integration with Oracle BPEL Process Manager ... 8-1
8.1.2 Oracle JMS Adapter Integration with Oracle Mediator ... 8-1
8.2 Oracle JMS Adapter Features ... 8-2
8.3 Oracle JMS Adapter Concepts ... 8-6
8.3.1 Point-to-Point ... 8-6
8.3.2 Publish/Subscribe ... 8-6
8.3.3 Destination, Connection, Connection Factory, and Session ... 8-7
8.3.4 Structure of a JMS Message ... 8-7
8.3.5 Oracle JMS Adapter Header Properties ... 8-8
8.4 Oracle JMS Adapter Use Cases ... 8-8
8.4.1 Configuring Oracle JMS Adapter ... 8-8
8.4.1.1 Creating an Application and a SOA Project ... 8-8
8.4.1.2 Using the Adapter Configuration Wizard to Configure Oracle JMS Adapter . 8-12
8.4.1.3 Generated Files ... 8-21
8.4.1.4 weblogic-ra.xml file ... 8-22
8.4.1.5 Produce Message Procedure ... 8-24
8.4.2 Configuring Oracle JMS Adapter with TIBCO JMS ... 8-26
8.4.2.1 NonDirect Connection ... 8-26
8.4.2.2 Direct Connection ... 8-28
8.4.3 Configuring Oracle JMS Adapter with IBM WebSphere MQ JMS ... 8-29
8.4.3.1 Non-XA Data Sources ... 8-29
8.4.3.2 XA Data Sources ... 8-30
8.4.4 WLS JMS Text Message ... 8-32
8.4.4.1 Meeting Prerequisites ... 8-32
8.4.4.2 Creating an Application Server Connection ... 8-34
8.4.4.3 Creating an Application and an SOA Project ... 8-34
8.4.4.4 Creating an Inbound Adapter Service ... 8-35
8.4.4.5 Creating an Outbound Adapter Service ... 8-39
8.4.4.6 Wiring Services and Activities ... 8-39
8.4.4.7 Deploying with JDeveloper ... 8-43
8.4.4.8 Monitoring Using the Fusion Middleware Control Console ... 8-43
8.4.5 Accessing Queues and Topics from WLS JMS Server in a Remote Oracle WebLogic Server Domain 8-43
8.4.5.1 JMS Adapter Limitations When a Remote Server is Used ... 8-44
8.4.6 Synchronous/Asynchronous Request Reply Interaction Pattern ... 8-44
8.4.6.1 Synchronous Request Reply Pattern ... 8-44

8.4.6.2 Asynchronous Request Reply Pattern ... 8-45
8.4.7 AQ JMS Text Message ... 8-45
8.4.7.1 Meeting Prerequisites ... 8-45
8.4.7.2 Create an Application Server Connection ... 8-49
8.4.7.3 Creating an Application and an SOA Project ... 8-49
8.4.7.4 Creating an Inbound Adapter Service ... 8-49
8.4.7.5 Creating an Outbound Adapter Service ... 8-50
8.4.7.6 Wiring Services and Activities ... 8-51
8.4.7.7 Deploying with JDeveloper ... 8-53
8.4.7.8 Monitoring Using the Fusion Middleware Control Console ... 8-53
8.4.8 Accessing Queues and Topics Created in 11*g* from the OC4J 10.1.3.4 Server ... 8-53
8.4.9 Configuring the 11G Server to Access Queues Present in 10.1.3.X OC4J ... 8-54
8.4.9.1 Copy Jar Files into the domains Folder of the Web Logic Server ... 8-55
8.4.9.2 Add Connector factory in the weblogic-ra.xml File ... 8-55
8.4.10 Accessing Distributed Destinations (Queues and Topics) on the WebLogic Server JMS 8-56
8.4.10.1 Providing JMS Adapter Access to Distributed Destinations ... 8-56
8.4.10.2 The JMS Adapter with Distributed Queues and Distributed Topics ... 8-57
8.4.10.3 One Copy of a Message Per Application (Default Behavior) ... 8-57
8.4.10.4 One Copy Of a Message Per Adapter Endpoint ... 8-58
8.4.10.5 Configuring Request-Reply in JMS Adapter ... 8-59

9 Oracle JCA Adapter for Database

9.1 Introduction to the Oracle Database Adapter ... 9-1
9.1.1 Functional Overview ... 9-1
9.1.1.1 Oracle Database Adapter Integration with Oracle BPEL PM ... 9-2
9.1.2 Design Overview ... 9-3
9.2 Complete Walkthrough of the Adapter Configuration Wizard ... 9-5
9.2.1 Creating an Application and an SOA Project ... 9-5
9.2.2 Defining an Oracle Database Adapter ... 9-9
9.2.3 Connecting to a Database ... 9-10
9.2.4 Selecting the Operation Type ... 9-12
9.2.5 Selecting and Importing Tables ... 9-15
9.2.6 Defining Primary Keys ... 9-16
9.2.7 Creating Relationships ... 9-17
9.2.7.1 What Happens When Relationships Are Created or Removed ... 9-20
9.2.7.2 Different Types of One-to-One Mappings ... 9-20
9.2.7.3 When Foreign Keys Are Primary Keys ... 9-20
9.2.8 Creating the Attribute Filter ... 9-21
9.2.9 Defining a WHERE Clause ... 9-22
9.2.10 Choosing an After-Read Strategy ... 9-24
9.2.10.1 Delete the Rows That Were Read ... 9-25
9.2.10.2 Update a Field in the Table (Logical Delete) ... 9-25
9.2.10.3 Update a Sequencing Table ... 9-27
9.2.10.4 Update an External Sequencing Table on a Different Database ... 9-28
9.2.10.5 Update a Sequencing File ... 9-29
9.2.11 Specifying Polling Options ... 9-29

9.2.12 Specifying Advanced Options 9-31
9.2.13 Entering the SQL String for the Pure SQL Operation 9-33
9.3 Oracle Database Adapter Features 9-34
9.3.1 Transaction Support 9-34
9.3.1.1 Configuring Oracle Database Adapter for Global Transaction Participation .. 9-35
9.3.1.2 Both Invokes in Same Global Transaction 9-36
9.3.1.3 Failure Must Cause Rollback 9-36
9.3.1.4 Transaction/XA Support 9-36
9.3.2 Pure SQL - XML Type Support 9-37
9.3.3 Row Set Support Using a Strongly or Weakly Typed XSD 9-38
9.3.4 Proxy Authentication Support 9-40
9.3.5 Streaming Large Payload 9-40
9.3.6 Schema Validation 9-41
9.3.7 High Availability 9-41
9.3.8 Scalability 9-41
9.3.8.1 Distributed Polling First Best Practice: SELECT FOR UPDATE (SKIP LOCKED) 9-41
9.3.8.2 Distributed Polling Second Best Practice: Tuning on a Single Node First 9-45
9.3.9 Performance Tuning 9-46
9.3.10 **`detectOmissions`** Feature 9-46
9.3.11 **`OutputCompletedXml`** Feature 9-48
9.3.12 QueryTimeout for Inbound and Outbound Transactions 9-49
9.3.13 Doing Synchronous Post to BPEL (Allow In-Order Delivery) 9-49
9.4 Oracle Database Adapter Concepts 9-49
9.4.1 Relational-to-XML Mapping 9-50
9.4.1.1 Relational Types to XML Schema Types 9-54
9.4.1.2 Mapping Any Relational Schema to Any XML Schema 9-55
9.4.1.3 Querying over Multiple Tables 9-55
9.4.2 SQL Operations as Web Services 9-60
9.4.2.1 DML Operations 9-60
9.4.2.2 Polling Strategies 9-62
9.5 Deployment 9-70
9.5.1 Deployment with Third Party Databases 9-72
9.6 JDBC Driver and Database Connection Configuration 9-73
9.6.1 Creating a Database Connection Using a Native or Bundled Oracle WebLogic Server JDBC Driver 9-74
9.6.2 Creating a Database Connection Using a Third-Party JDBC Driver 9-74
9.6.3 Summary of Third-Party JDBC Driver and Database Connection Information 9-76
9.6.3.1 Using a Microsoft SQL Server 9-76
9.6.3.2 Using a Sybase Database 9-77
9.6.3.3 Using an Informix Database 9-77
9.6.3.4 Using an IBM DB2 Database 9-78
9.6.3.5 Using a MySQL Database 9-78
9.6.4 Location of JDBC Driver JAR Files and Setting the Class Path 9-78
9.7 Stored Procedure and Function Support 9-79
9.7.1 Design Time: Using the Adapter Configuration Wizard 9-79
9.7.1.1 Using Top-Level Standalone APIs 9-79
9.7.1.2 Using Packaged APIs and Overloading 9-90

9.7.2 Supported Third-Party Databases ... 9-92
9.7.2.1 Terms Used ... 9-93
9.7.2.2 Supported Third-Party Databases ... 9-94
9.7.2.3 Creating Database Connections ... 9-100
9.7.3 Design Time: Artifact Generation ... 9-103
9.7.3.1 The WSDL–XSD Relationship ... 9-104
9.7.3.2 JCA File ... 9-105
9.7.3.3 Oracle Data Types ... 9-106
9.7.3.4 Generated XSD Attributes ... 9-106
9.7.3.5 User-Defined Types ... 9-107
9.7.3.6 Complex User-Defined Types ... 9-109
9.7.3.7 Object Type Inheritance ... 9-109
9.7.3.8 Object References ... 9-110
9.7.3.9 Referencing Types in Other Schemas ... 9-110
9.7.3.10 XSD Pruning Optimization ... 9-111
9.7.4 Run Time: Before Stored Procedure Invocation ... 9-112
9.7.4.1 Value Binding ... 9-112
9.7.4.2 Data Type Conversions ... 9-114
9.7.5 Run Time: After Stored Procedure Invocation ... 9-115
9.7.5.1 Data Type Conversions ... 9-115
9.7.5.2 Null Values ... 9-116
9.7.5.3 Function Return Values ... 9-116
9.7.6 Run Time: Common Third-Party Database Functionality ... 9-116
9.7.6.1 Processing ResultSets ... 9-116
9.7.6.2 Returning an INTEGER Status Value ... 9-117
9.7.7 Advanced Topics ... 9-118
9.7.7.1 Row Set Support Using a Strongly Typed XSD ... 9-118
9.7.7.2 Row Set Support Using a Weakly Typed XSD ... 9-127
9.7.7.3 Support for PL/SQL Boolean, PL/SQL Record, and PL/SQL Table Types ... 9-129
9.8 Oracle Database Adapter Use Cases ... 9-133
9.8.1 Use Cases for Oracle Database Adapter ... 9-133
9.8.2 Use Cases for Oracle Database Adapter - Stored Procedures ... 9-135
9.8.2.1 Creating and Configuring a Stored Procedure in JDeveloper BPEL Designer ... 9-136
9.8.2.2 File To StoredProcedure Use Case ... 9-146
9.8.3 DB Adapter/Coherence Integration ... 9-161
9.8.3.1 Inserst/Updates to a Database ... 9-161

A Oracle JCA Adapter Properties

A.1 Oracle File and FTP Adapters Properties ... A-1
A.2 Oracle Socket Adapter Properties ... A-7
A.3 Oracle AQ Adapter Properties ... A-8
A.4 Oracle JMS Adapter Properties ... A-10
A.5 Oracle Database Adapter Properties ... A-13
A.6 Oracle MQ Series Adapter Properties ... A-15
A.7 Generic Oracle JCA Adapter Properties ... A-22

B Oracle JCA Adapter Valves

9.9 A Simple Unzip Valve B-1
9.10 A Simple Decryption Valve That Uses Staging File B-2
9.11 A Valve for Encrypting Outbound Files B-5
9.12 An Unzip Valve for processing Multiple Files B-7

Index

10 Oracle JCA Adapter for MQ Series

10.1 MQ Series Message Queuing Concepts C-1
10.1.1 MQ Series Concepts C-3
10.2 Introduction to Native Oracle MQ Series Adapter C-5
10.2.1 The Need for Oracle MQ Series Adapter C-5
10.2.2 Oracle MQ Series Adapter Integration with Oracle BPEL Process Manager C-6
10.2.3 Oracle MQ Series Adapter Integration with Mediator C-7
10.3 Oracle MQ Series Adapter Features C-7
10.3.1 RFH Version 2 (RFH2) Header C-8
10.3.1.1 Fixed Portion C-8
10.3.1.2 Variable Portion C-9
10.3.2 SSL Enabling C-11
10.3.3 XA Transactions C-11
10.3.3.1 XA Recovery C-12
10.3.4 High Availability C-13
10.3.4.1 Prerequisites for High Availability C-13
10.3.4.2 High Availability in Inbound/Outbound Operations C-13
10.3.5 Scalability C-13
10.3.6 Securing Enterprise Information System Credentials C-14
10.3.7 Fault Policy C-14
10.3.8 Inbound Rejection Handler C-14
10.3.9 Retry Mechanism C-14
10.3.9.1 JCA Inbound Retry Mechanism C-15
10.3.9.2 Message Backout Queue C-15
10.3.10 Performance Tuning C-16
10.4 Oracle MQ Series Adapter Concepts C-16
10.4.1 Messaging Scenarios C-16
10.4.1.1 Enqueue Message C-16
10.4.1.2 Dequeue Message C-20
10.4.1.3 Asynchronous Request-Response (Oracle BPEL PM As Client) C-23
10.4.1.4 Synchronous Request-Response (Oracle BPEL PM As Server) C-29
10.4.1.5 Asynchronous Request-Response (Oracle BPEL PM As Server) C-34
10.4.1.6 Synchronous Request-Response (Mediator As Server) C-41
10.4.1.7 Synchronous Request-Response (Oracle BPEL PM As Client) C-42
10.4.1.8 Synchronous Request-Response (Oracle Mediator as Client) C-44
10.4.1.9 Asynchronous Request-Response (Oracle Mediator As Client) C-44
10.4.1.10 Outbound Dequeue Scenario C-47
10.4.2 Message Properties C-47
10.4.2.1 Messages Types C-47

10.4.2.2 Message Format ... C-48
10.4.2.3 Message Expiry ... C-48
10.4.2.4 Message Priority ... C-48
10.4.2.5 Message Persistence ... C-49
10.4.3 Correlation Schemas ... C-49
10.4.4 Distribution List Support ... C-49
10.4.5 Report Messages ... C-49
10.4.6 Message Delivery Failure Options ... C-50
10.4.7 Message Segmentation ... C-51
10.4.8 Integration with CICS ... C-51
10.4.9 Supported Encodings ... C-57
10.5 Configuring the Oracle MQ Series Adapter ... C-59
10.5.1 Adding com.ibm.mq.jar to the Oracle MQ Series Adapter Classpath ... C-60
10.5.2 Adding JNDI Entry ... C-60
10.5.3 Enabling Binding Mode for Connections ... C-63
10.6 Oracle MQ Series Adapter Use Cases ... C-64
10.6.1 Dequeue Enqueue ... C-64
10.6.1.1 Prerequisites ... C-65
10.6.1.2 Designing the SOA Composite ... C-65
10.6.1.3 Creating an Inbound Adapter Service ... C-69
10.6.1.4 Creating an Outbound Adapter Service ... C-75
10.6.1.5 Wiring Services and Activities ... C-77
10.6.1.6 Deploying with JDeveloper ... C-80
10.6.1.7 Monitoring Using the Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console) C-81
10.6.2 Inbound Synchronous Request-Reply ... C-81
10.6.2.1 Prerequisites ... C-81
10.6.2.2 Designing the SOA Composite ... C-82
10.6.2.3 Creating an Inbound Adapter Service ... C-83
10.6.2.4 Wiring Services and Activities ... C-84
10.6.2.5 Deploying with JDeveloper ... C-86
10.6.2.6 Monitoring Using the Fusion Middleware Control Console ... C-86
10.6.3 Inbound-Outbound Synchronous Request-Reply ... C-87
10.6.3.1 Prerequisites ... C-87
10.6.3.2 Designing the SOA Composite ... C-88
10.6.3.3 Creating an Inbound Adapter Service ... C-89
10.6.3.4 Creating an Outbound Adapter Service ... C-90
10.6.3.5 Wiring Services and Activities ... C-92
10.6.3.6 Deploying with JDeveloper ... C-98
10.6.3.7 Monitoring Using the Fusion Middleware Control Console ... C-98
10.6.4 Asynchronous-Request-Reply ... C-99
10.6.4.1 Prerequisites ... C-99
10.6.4.2 Designing the SOA Composite ... C-99
10.6.4.3 Creating an Inbound Adapter Service ... C-100
10.6.4.4 Creating an Asynchronous Outbound Request Reply Adapter Service Outbound ... C-102
10.6.4.5 Creating Another Outbound Adapter Service ... C-103
10.6.4.6 Wiring Services and Activities ... C-104

10.6.4.7 Deploying with JDeveloper C-109
10.6.4.8 Monitoring Using the Fusion Middleware Control Console C-109
10.6.5 Outbound Dequeue C-110
10.6.5.1 Prerequisites C-110
10.6.5.2 Designing the SOA Composite C-111
10.6.5.3 Creating an Outbound Dequeue Adapter Service C-112
10.6.5.4 Wiring Services and Activities C-113
10.6.5.5 Deploying with JDeveloper C-117
10.6.5.6 Monitoring Using the Fusion Middleware Control Console C-117
10.6.6 Using RFH2 Headers C-118
10.6.6.1 Designing the SOA Composite C-118
10.6.6.2 Creating an Inbound Adapter Service C-120
10.6.6.3 Creating an Outbound Adapter Service C-121
10.6.6.4 Wiring Services and Activities C-122
10.6.6.5 Deploying with JDeveloper C-128
10.6.7 Configuring a Backout Queue C-128
10.6.7.1 Prerequisites C-129
10.6.7.2 Designing the SOA Composite C-129
10.6.7.3 Creating an Inbound Adapter Service C-130
10.6.7.4 Creating an Outbound Adapter Service C-130
10.6.7.5 Wiring Services and Activities C-131
10.6.7.6 Deploying with JDeveloper C-134
10.6.7.7 Monitoring Using the Fusion Middleware Control Console C-134

Preface

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

Oracle Fusion Middleware User's Guide for Technology Adapters is intended for anyone who is interested in using these adapters.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

For more information, see the following documents in the *Oracle Fusion Middleware* 11g Release 1 (11.1.1.4.1) documentation set:

- *Oracle Fusion Middleware Programming Resource Adapters for Oracle WebLogic Server*
- *Oracle Fusion Middleware Adapter for Oracle Applications User's Guide*
- *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*
- *Oracle Fusion Middleware Administrator's Guide*
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*
- *Oracle Fusion Middleware Administrator's Guide for Oracle Service Bus*
- *Oracle® Application Server Installation Guide for Legacy Adapters*

Conventions

The following text conventions are used in this document:

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

What's New in This Guide for Release 11.1.1.6.1

For Release 11.1.1.6.1, this guide has been updated in several ways. The following table lists the sections that have been added or changed. If a feature was not available in the first release of 11.1.1.6.0, the last columns denote which documentation release contains the update.

For a list of known issues (release notes), see the "Known Issues for for Oracle SOA Products and Oracle AIA Foundation Pack" at http://www.oracle.com/technetwork/middleware/docs/soa-aiafp-knownissuesindex-364630.html.

Sections	Changes Made	11.1.1.6.0	11.1.1.6.1
Chapter 1 Introduction			
Section 1.1, "Features of Oracle JCA Adapters"	Added paragraph at beginning of chapter to provide a high level overview summary of Adapters	X	
Chapter 2 Life Cycle			
Example 2–2, "Log Error When CorrelationId of the Receive Does not Match any Earlier Invoke"	Modifed text in this example to clarify how example works.	X	
Table 2–2, " Translation Scenarios Supported for Oracle JCA Adapters"	Changed title to specify the scenarios are "Translation" scenarios and changed a row in the table from "Case" to "Scenarios"	X	
Section 2.19.3, "Use the Web Logic Server Console to Create a New Connection"	Added subsection "Use the Web Logic Server Console to Create a New Connection"	X	
Chapter 4 File ad FTP Adapter			
Section 4.5.12, "Creating a Synchronous BPEL Composite using File Adapter"	Added subsection, "Creating a Synchronous BPEL Composite using File Adapter"	X	
Section 4.5.12, "Creating a Synchronous BPEL Composite using File Adapter"	Added subsection "Changing the Connection Factory JNDI Dynamically in Ftp Adapter"s	X	
Section 4.5.13, "Changing the Sequencing Strategy for FILE/Ftp Adapter"	Added subsection, "Changing the Sequencing Strategy for FILE/Ftp Adapter"	X	

Sections	Changes Made	11.1.1.6.0	11.1.1.6.1
Section 4.5.12.2, "Retrieving the Details of the File from an Outbound Write Operation"	Added subsection, "Retrieving the Details of the File from an Outbound Write Operation"	X	
Section 4.4.3.2.3, "Creating an Oracle Wallet"	New subsection relating secure FTP and Oracle Wallet.	X	
Section 4.4.3.2.4, "Setting Up the Oracle FTP Adapter"	Revised table in section to reference Oracle Wallet and clarified additional definitions.s	X	
Chapter 6 Native Format Builder Wizard			
Changes to text for clarification		X	
Section 6.5, "Command Line Tool for Testing NXSD Translator"	Added section to provide information on Command Line Tool for Testing NXSD Translators	X	
Chapter 8 Oracle JCA Adapter for JMS			
Revised Chapter for clarity.			
Section 8.4.10.5, "Configuring Request-Reply in JMS Adapter"	Added this section	X	
Chapter 9 Oracle JCA Adapter for Database			
Modifications to wording for clarity.		X	
Section 9.3.1, "Transaction Support"	Added specifics about samples related to this subsection, specifically Transaction Support.		
Section 9.8.3, "Database Adapter/Coherence Integration"	Added section on Coherence Integration with Database Adapter.	X	
Appendix A Oracle JCA Adapter Properties			
Added additional summary paragraph at beginning of Appendix to clarify purpose of Appendix.		X	
Section A–12, " JCA Properties for Oracle JMS Adapter: Normalized Properties"	Added more text to the description of `jca.jms.JMSProperty.`*`name`*	X	

Part I

Introduction and Concepts

Part I contains the following chapters:

- Chapter 1, "Introduction to Oracle JCA Adapters"
- Chapter 2, "ADAPTER Life-Cycle Management"
- Chapter 3, "Adapter Integration with Oracle Application Server Components"
- Chapter 4, "Oracle JCA Adapter for Files/FTP"
- Chapter 5, "Oracle JCA Adapter for Sockets"
- Chapter 6, "Native Format Builder Wizard"

1

Introduction to Oracle JCA Adapters

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

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

For more information on Open Service Bus-specific adapter configuration, see the *Oracle® Fusion Middleware Developer's Guide for Oracle Service Bus*, JCA Transport section:

`http://docs.oracle.com/cd/E23943_01/dev.1111/e15866/jca.htm#OSBDV924`

This chapter includes the following sections:

- Section 1.1, "Features of Oracle JCA Adapters"
- Section 1.2, "Types of Oracle JCA Adapters"
- Section 1.3, "Types of Oracle JCA Adapters Adapter Services"

1.1 Features of Oracle JCA Adapters

Oracle JCA Adapters provide the following benefits:

- Provide a connectivity platform for integrating complex business processes: Adapters integrate mainframe and legacy applications with enterprise resource planning (ERP), customer relationship management (CRM), databases, and messaging systems. Oracle provides adapters to connect various packaged applications, such as SAP and Siebel, and databases. In addition, adapters integrate middleware messaging systems, such as MQSeries and Oracle Advanced Queuing, and legacy applications, such as CICS and Tuxedo, to provide a complete solution.

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

  For more information, see *Oracle Fusion Middleware Programming Resource Adapters for Oracle WebLogic Server*.

  In addition, adapters can be deployed on the JBoss and WebSphere platforms.
- Provide easy-to-use design-time tools: Adapters use design-time tools that provide a graphical user interface (GUI) to configure and administer adapters for fast implementation and deployment. In addition, the tools let you to browse, download, and configure back-end schemas.
- Support seamless integration with Oracle Application Server components: Adapters integrate with Oracle Fusion Middleware. Adapters integrate with the JCA Binding Component of the Oracle Fusion Middleware platform, thereby seamlessly integrating with other service engines and binding components.

1.2 Types of Oracle JCA Adapters

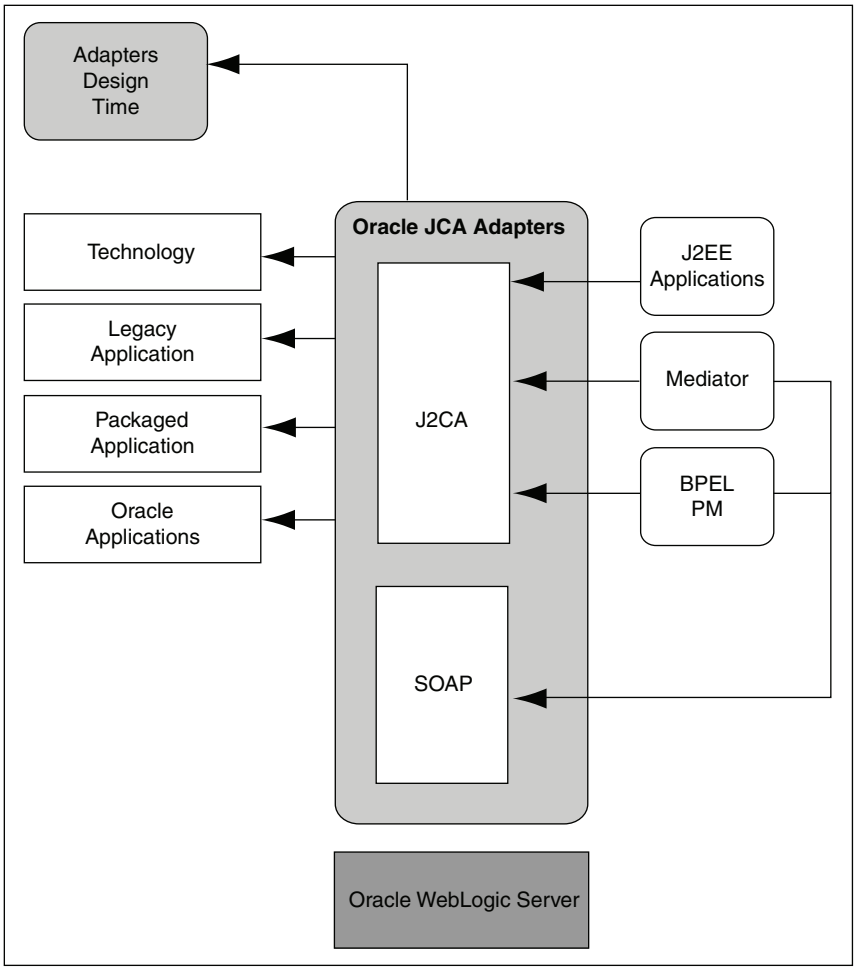
Oracle JCA Adapters include:

- Section 1.2.1, "Oracle Technology Adapters"
- Section 1.2.2, "Legacy Adapters"

- Section 1.2.3, "Packaged-Application Adapters"
- Section 1.2.4, "Oracle Adapter for Oracle Applications"

Figure 1–1 illustrates the different types of adapters.

Figure 1–1 Types of Oracle JCA Adapters



1.2.1 Oracle Technology Adapters

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

These adapters include:

- Oracle JCA Adapter for Files/FTP
- Oracle JCA Adapter for Sockets
- Oracle JCA Adapter for Oracle Streams Advanced Queuing (Oracle JCA Adapter for AQ)
- Oracle JCA Adapter for JMS
- Oracle JCA Adapter for Database

- Oracle JCA Adapter for MQ Series

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

This section includes the following topics:

- Section 1.2.1.1, "Architecture"
- Section 1.2.1.2, "Design-Time Components"
- Section 1.2.1.3, "Run-Time Components"
- Section 1.2.1.4, "Deployment"

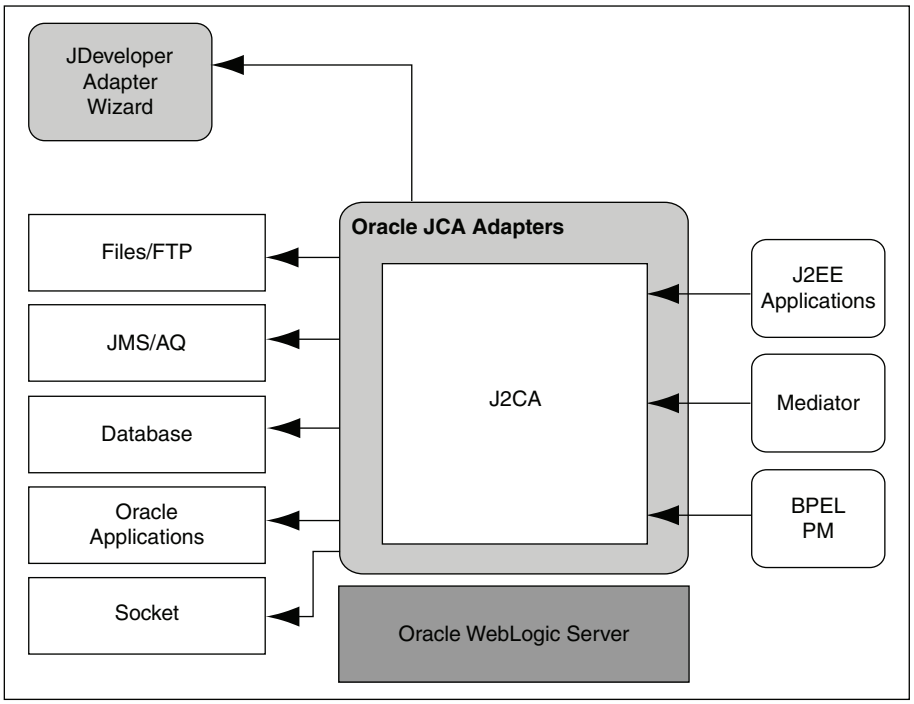
For more information, see:

- the remaining chapters in this book
- *Oracle Fusion Middleware Programming Resource Adapters for Oracle WebLogic Server*
- *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*
- *Oracle Fusion Middleware Administrator's Guide*
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*

1.2.1.1 Architecture

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

Figure 1–2 Oracle Technology Adapters Architecture



1.2.1.2 Design-Time Components

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

For more information about integration of Oracle technology adapters with Oracle Fusion Middleware, see Section 3.2, "Adapter Integration with Oracle Fusion Middleware."

Example 1–1 Generating WSDL and Binding Configuration Files for Oracle JCA Adapter for Database

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

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

Figure 1–3 Browsing for Required Tables

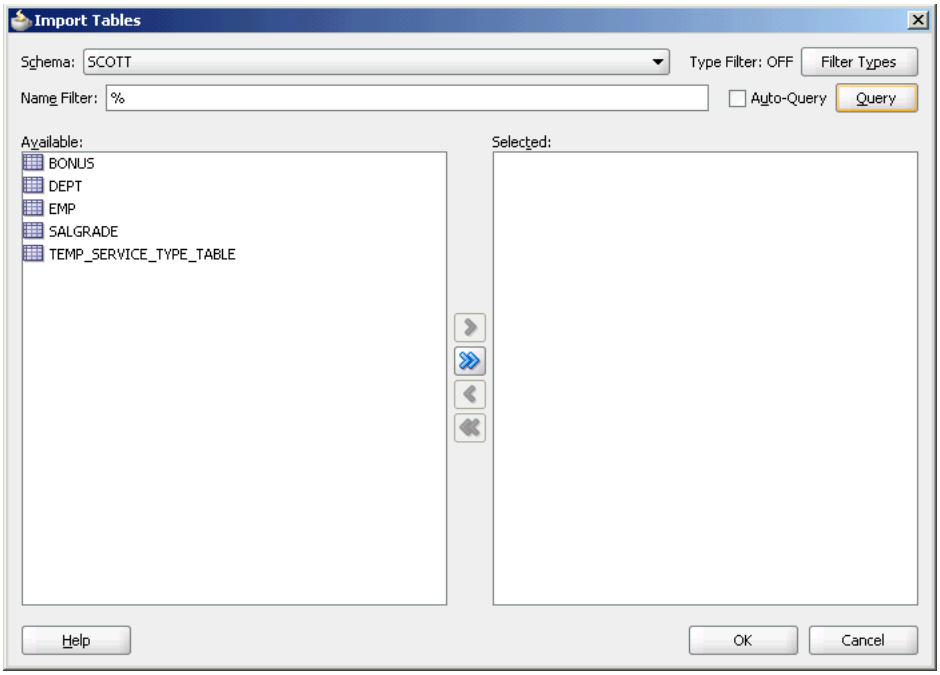
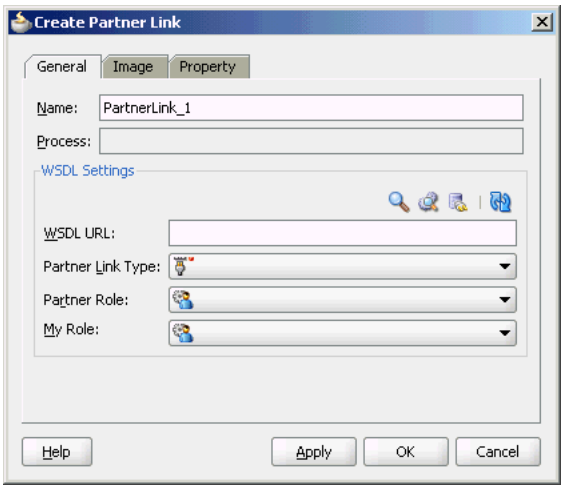


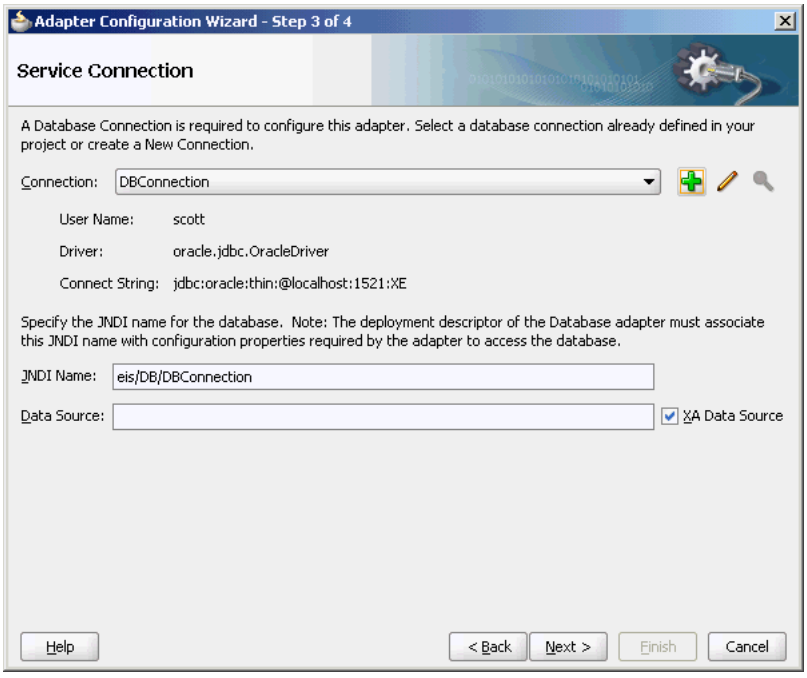
Figure 1–4 shows how to specify the WSDL settings for Oracle JCA Adapter for Database.

Figure 1–4 Specifying WSDL Settings



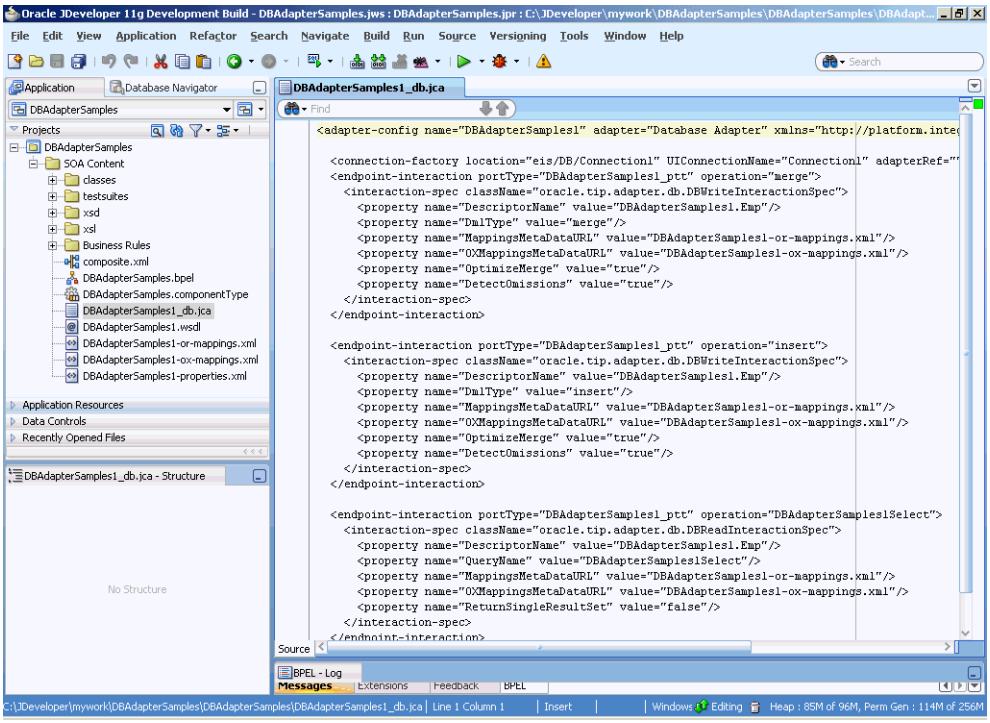
Next, you must establish a database connection, select an operation type, and select the required tables. The run-time connection parameters are specified in the `weblogic-ra.xml` file and linked to a Java Naming and Directory Interface (JNDI) name, which is specified during design time. Figure 1–5 shows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

Figure 1–5 Creating a New Database Connection



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

Figure 1–6 Structure of a JCA File



1.2.1.3 Run-Time Components

The run-time component of Oracle technology adapters is the J2CA 1.5 resource adapter for the specific back-end application. Oracle technology adapters are deployed in the J2CA container of the Oracle WebLogic Server. Oracle Fusion Middleware integrates with these J2CA 1.5 adapters through the JCA Binding Component, which converts Web service messages into J2CA interactions and back.

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

For more information about integration with Oracle Fusion Middleware, see Chapter 3, "Adapter Integration with Oracle Application Server Components".

1.2.1.4 Deployment

Oracle technology adapters are deployed as J2CA 1.5 resource adapters within the same Oracle WebServer container as that of Oracle Fusion Middleware during installation. Although Oracle technology adapters are physically deployed as J2CA 1.5 resource adapters, their logical deployment involves creating the connection factory entries for the J2CA 1.5 resource adapter by editing the `weblogic-ra.xml` file and using JDeveloper during design time. By using JDeveloper, you specify the JNDI name, which acts as a placeholder for the connection used when your composite is deployed to the Oracle WebLogic Server. This placeholder enables you to use different databases for development and later production. However, for the logical deployment changes (that is, only if you are creating a new outbound connection) to take effect, the WebLogic Server container process should be updated. However, if you are updating any outbound connection property for an existing JNDI, then you must restart the Oracle WebLogic Server. To avoid a server restart when you update an outbound connection property for an existing JNDI, refer to Section 2.19, "Adding or Updating an Adapter Connection Factory."

1.2.2 Legacy Adapters

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

These adapters include:

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

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

This section includes the following topics:

- Section 1.2.2.1, "Architecture"
- Section 1.2.2.2, "Design-Time Components"
- Section 1.2.2.3, "Run-Time Components"
- Section 1.2.2.4, "Deployment"

For more information, see:

- *Oracle Fusion Middleware Programming Resource Adapters for Oracle WebLogic Server*
- *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*
- *Oracle Fusion Middleware Administrator's Guide*
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*

1.2.2.1 Architecture

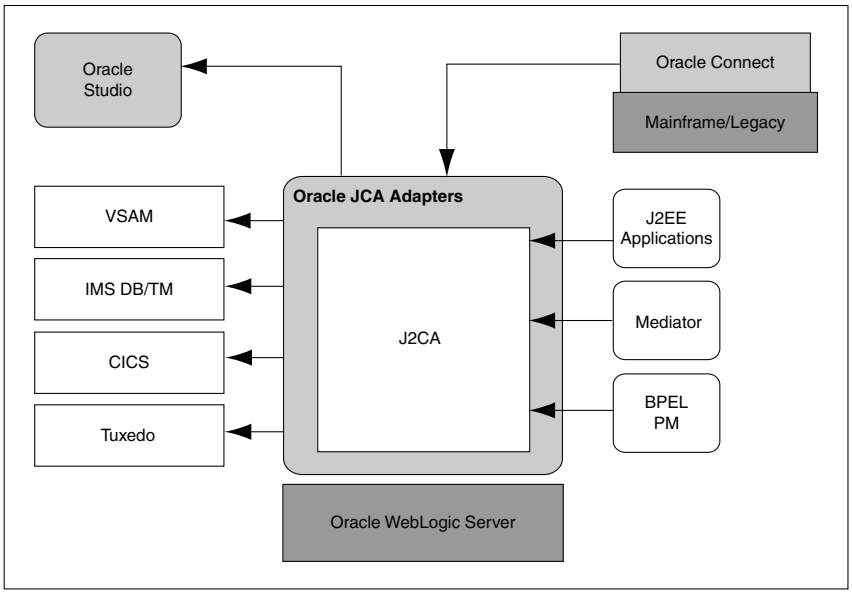
Legacy adapters include the following components in the architecture

- Section 1.2.2.1.1, "Oracle Connect"
- Section 1.2.2.1.2, "Oracle Studio"
- Section 1.2.2.1.3, "J2CA Adapter"

Figure 1–7 illustrates the architecture of legacy adapters.

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

Figure 1–7 Legacy Adapter Architecture



1.2.2.1.1 Oracle Connect

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

- Server Processes
- Native Adapters
- Daemon
- Repository

Server Processes

Oracle Connect consists of multiple servers to process client requests.

Native Adapters

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

Daemon

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

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

notified when the connection ends and the server process is either killed or being used by another client.

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

Repository

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

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

1.2.2.1.2 Oracle Studio

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

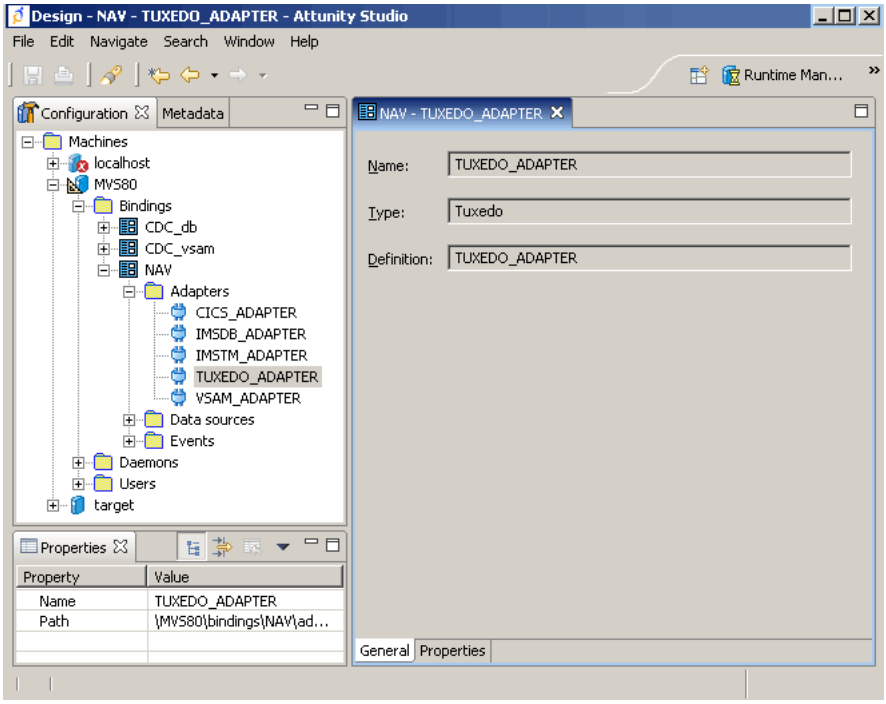
1.2.2.1.3 J2CA Adapter

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

1.2.2.2 Design-Time Components

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

Figure 1–8 Oracle Studio



Example 1–2 Configuring OracleAS Adapter for Tuxedo

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

Figure 1–9 Configuring OracleAS Adapter for Tuxedo

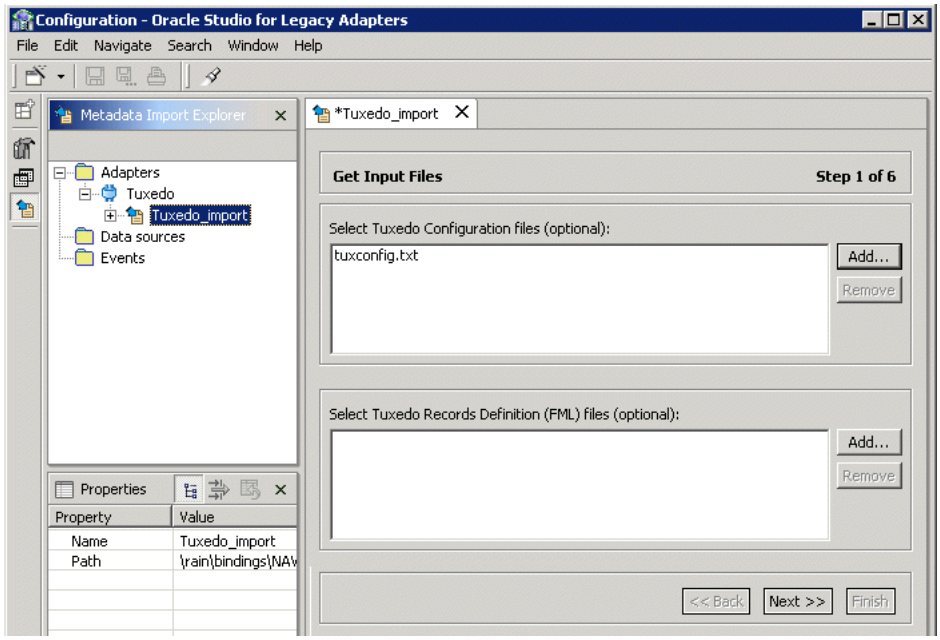
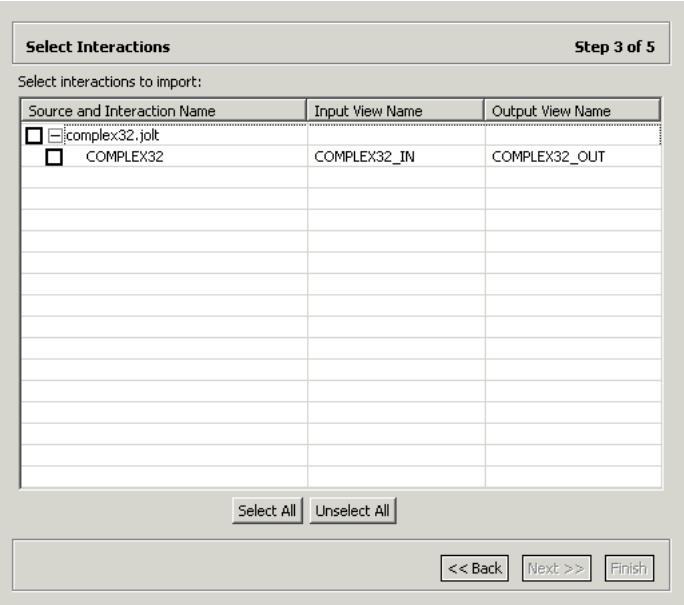


Figure 1–10 Selecting the Types of Interactions for OracleAS Adapter for Tuxedo



1.2.2.3 Run-Time Components

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

For more information, see Section 3.2, "Adapter Integration with Oracle Fusion Middleware."

1.2.2.4 Deployment

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

1.2.3 Packaged-Application Adapters

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

These adapters include:

- OracleAS Adapter for PeopleSoft
- OracleAS Adapter for SAP R/3
- OracleAS Adapter for Siebel
- OracleAS Adapter for J.D. Edwards

Packaged-application adapters are available as part of the OracleAS Adapters CD.

This section includes the following topics:

- Section 1.2.3.1, "Architecture"
- Section 1.2.3.2, "Design-Time Components"

- Section 1.2.3.3, "Run-Time Components"
- Section 1.2.3.4, "Deployment"

For more information, see:

- *Oracle Fusion Middleware Programming Resource Adapters for Oracle WebLogic Server*
- *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*
- *Oracle Fusion Middleware Administrator's Guide*
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*

1.2.3.1 Architecture

Packaged-application adapters can be deployed as J2EE Connector Architecture (J2CA) 1.5 resource adapters or as Web service servlets within the Oracle WebLogic Server container. Packaged-application adapters support the Web Service Definition Language (WSDL) and Simple Object Access Protocol (SOAP) interface, in addition to a J2CA interface. J2CA and Web service deployments of packaged-application adapters should have a repository project. In J2CA deployment, the resource adapter points to a repository project that can contain multiple back-end connection objects. The deployment descriptor, `weblogic-ra.xml`, points to the J2CA repository project and the connection name to access within the J2CA repository project. In the WSDL deployment, the WSDL repository project consists of a set of WSDL files that describe the adapter metadata.

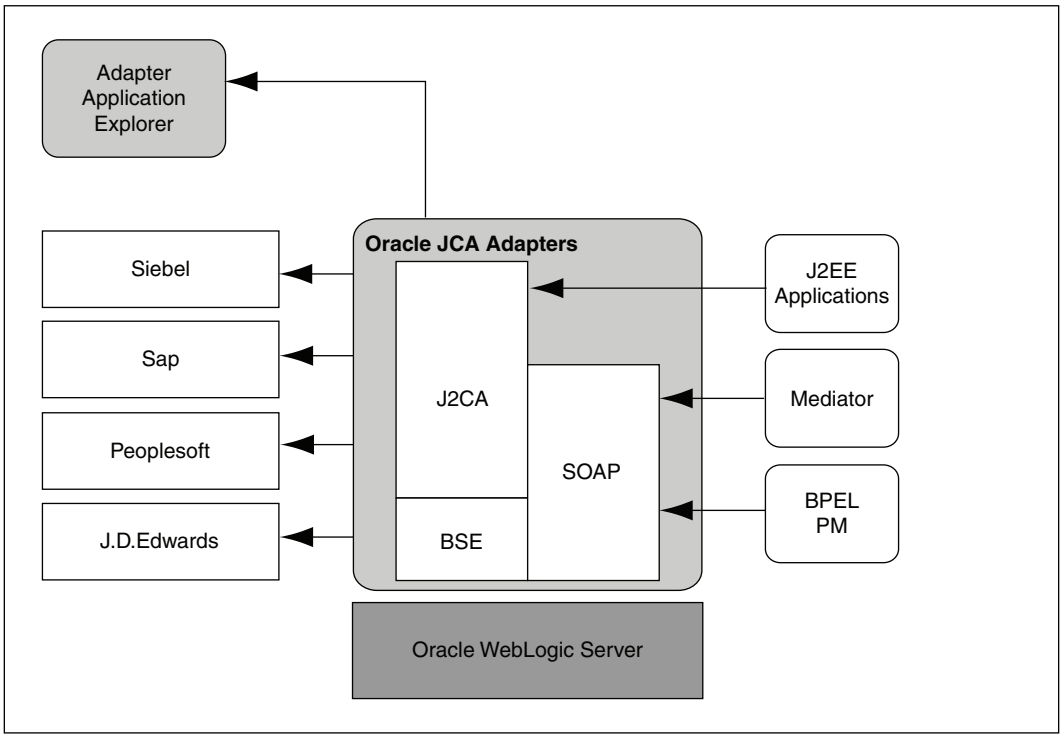
Note: Only the following packaged-application adapters support WSDL and SOAP extensions in this release:

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

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

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

Figure 1–11 Packaged-Application Adapters Architecture



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

This section includes the following topics:

- Section 1.2.3.1.1, "Application Explorer"
- Section 1.2.3.1.2, "BSE"
- Section 1.2.3.1.3, "J2CA 1.5 Resource Adapter"

1.2.3.1.1 Application Explorer

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

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

1.2.3.1.2 BSE

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

1.2.3.1.3 J2CA 1.5 Resource Adapter

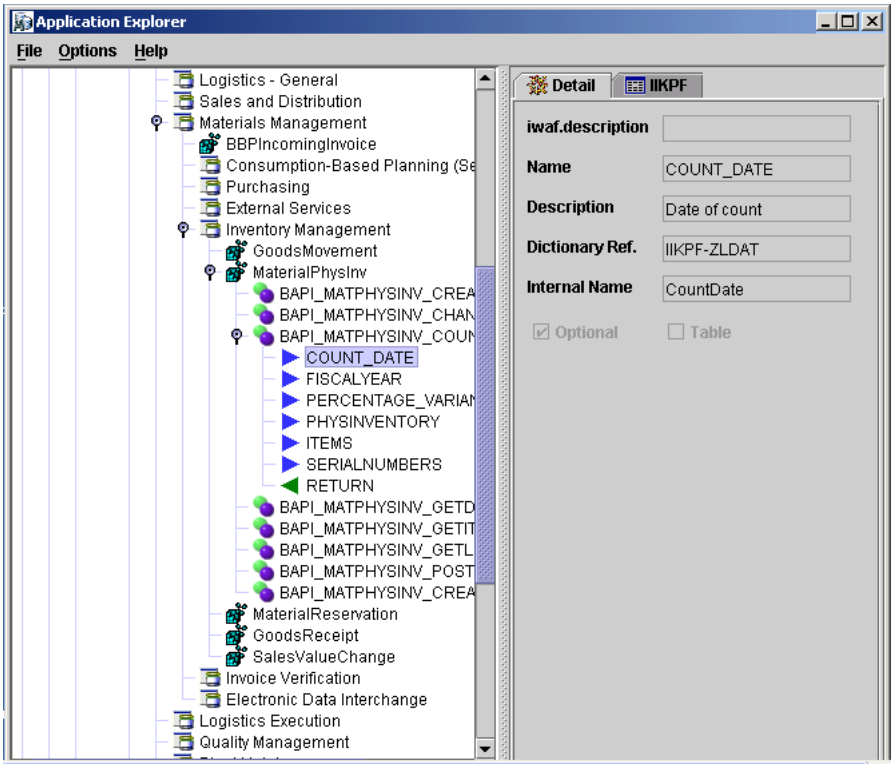
The J2CA 1.5 resource adapter consists of a Channel component for receiving back-end events.

1.2.3.2 Design-Time Components

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

Figure 1–12 shows the Application Explorer.

Figure 1–12 Application Explorer



Example 1–3 Generating XML Request Schema for OracleAS Adapter for SAP

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

Figure 1–13 Selecting OracleAS Adapter for SAP

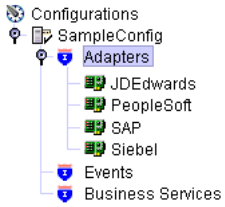
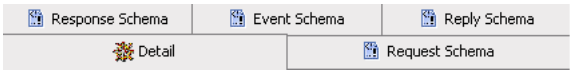


Figure 1–14 Defining a Target to OracleAS Adapter for SAP

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

Figure 1–15 Viewing the XML Schema



1.2.3.3 Run-Time Components

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

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

For more information about integrating with Oracle Fusion Middleware, see Section 3.2, "Adapter Integration with Oracle Fusion Middleware".

1.2.3.4 Deployment

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

You can integrate any Web service client with the BSE servlet. The BSE exposes the underlying back-end functionality as Web services, which can be either WSDL or SOAP. Oracle BPEL PM can integrate with the BSE layer, as well, through WSDL and SOAP binding.

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

1.2.4 Oracle Adapter for Oracle Applications

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

Note: Adapter for Oracle Applications is also informally known as Oracle E-Business Suite Adapter

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

The architecture of the Oracle Adapter for Oracle Applications is similar to Oracle technology adapters.

For more information, see:

- *Oracle Fusion Middleware Adapter for Oracle Applications User's Guide*
- *Oracle Fusion Middleware Programming Resource Adapters for Oracle WebLogic Server*
- *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*
- *Oracle Fusion Middleware Administrator's Guide*
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*

1.3 Types of Oracle JCA Adapters Adapter Services

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

- Section 1.3.1, "Request-Response (Outbound Interaction) Service"
- Section 1.3.2, "Event Notification (Inbound Interaction) Service"
- Section 1.3.3, "Metadata Service"

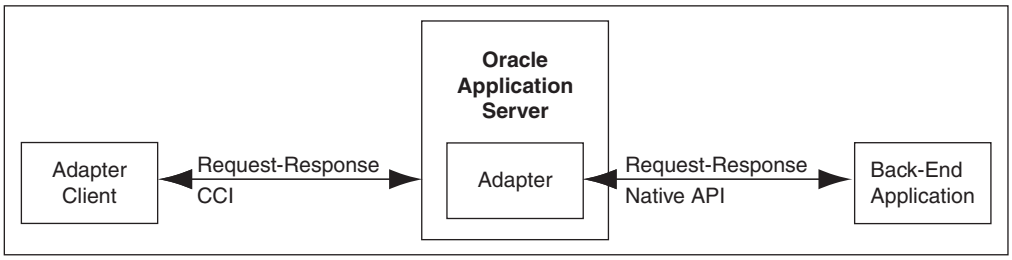
1.3.1 Request-Response (Outbound Interaction) Service

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

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

Figure 1–16 illustrates the request-response service.

Figure 1–16 Request-Response Service



1.3.2 Event Notification (Inbound Interaction) Service

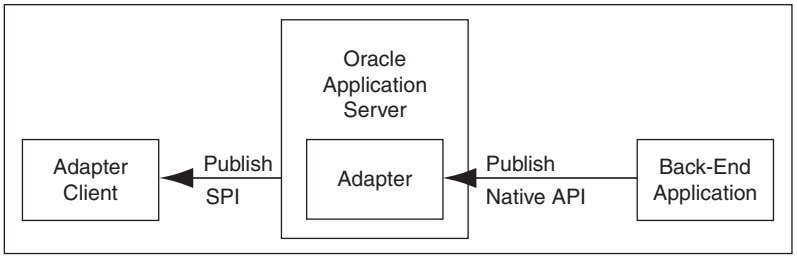
Adapters support the event-notification service, which is an asynchronous communication paradigm. In J2CA terminology, this type of service is also known as inbound interaction.

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

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

Figure 1–17 illustrates the event-notification service.

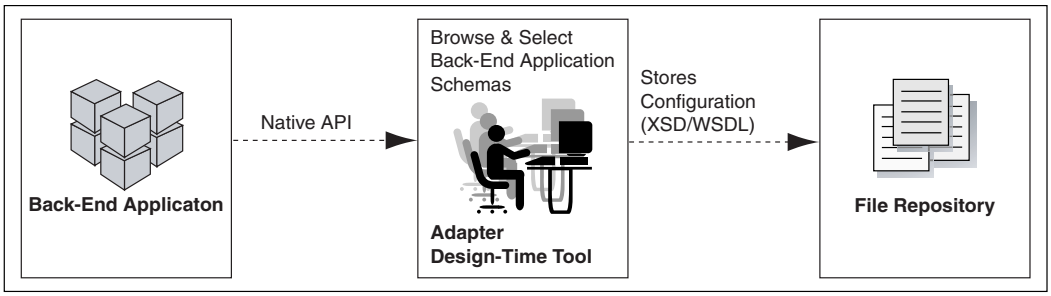
Figure 1–17 Event-Notification Service



1.3.3 Metadata Service

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

Figure 1–18 Metadata Service



2

ADAPTER Life-Cycle Management

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

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

This chapter includes the following sections:

- Section 2.1, "Installing Oracle JCA Adapters"
- Section 2.2, "Starting and Stopping Oracle JCA Adapters"
- Section 2.3, "Defining Adapter Interface by Importing an Existing WSDL"
- Section 2.4, "Configuring Message Header Properties for Oracle JCA Adapters"
- Section 2.5, "Physically Deploying Oracle JCA Adapters"
- Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters"
- Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper"
- Section 2.8, "Manually Deploying an Adapter RAR File that Does Not Have a Jar File Associated With It"
- Section 2.9, "Handling the Deployment Plan When Working on a Remote Oracle SOA Server"
- Section 2.10, "Migrating Repositories from Different Environments"
- Section 2.11, "How Oracle JCA Adapters Ensure No Message Loss"
- Section 2.12, "Composite Availability and Inbound Adapters"
- Section 2.13, "Singleton (Active/Passive) Inbound Endpoint Lifecycle Support Within Adapters"
- Section 2.14, "Correlation Support Within Adapters"
- Section 2.15, "Setting Payload Size Threshold"
- Section 2.16, "Streaming Large Payload"
- Section 2.17, "Batching and Debatching Support"
- Section 2.18, "Adding an Adapter Connection Factory"
- Section 2.19, "Adding or Updating an Adapter Connection Factory"

- Section 2.20, "Recommended Setting for Data Sources Used by Oracle JCA Adapters"
- Section 2.21, "Error Handling"
- Section 2.22, "Testing Applications"
- Section 2.23, "Setting the Trace Level of Oracle JCA Adapters"
- Section 2.24, "Viewing Adapter Logs"
- Section 2.25, "Creating a Custom Adapter"

2.1 Installing Oracle JCA Adapters

Oracle Technology Adapters and Oracle Adapter for Oracle Applications are available as part of the Oracle Fusion Middleware install. These adapters support both Oracle Containers for Java EE and middle tier deployments. For more information, see the *Oracle Fusion Middleware Installation Planning Guide*.

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

Note: Before installing any adapter, "System Requirements and Supported Platforms for Oracle Fusion Middleware 11gR1" document on the following page:
`http://www.oracle.com/technetwork/middleware/ias/downloads/ fusion-certification-100350.html`

2.2 Starting and Stopping Oracle JCA Adapters

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

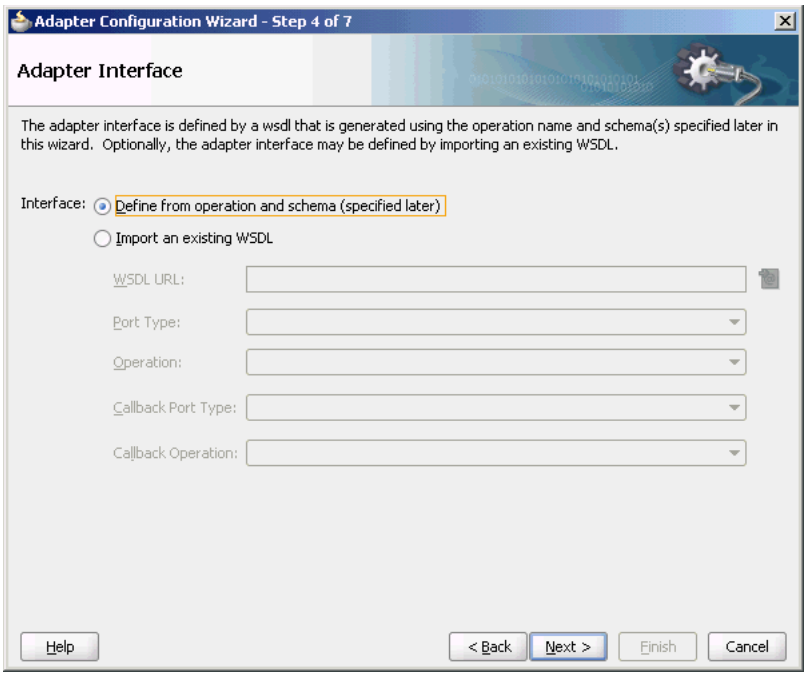
Once you have started an adapter, you can stop the adapter by shutting down the Oracle Containers for Java EE or by stopping the J2EE application within Oracle Fusion Middleware. In this release, the JCA Binding Component acts as a part of the JCA 1.5 container.

2.3 Defining Adapter Interface by Importing an Existing WSDL

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

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

Figure 2–1 The Adapter Configuration Wizard Adapter Interface Page



This figure shows the Adapter Configuration Wizard - Adapter Interface page where you define the adapter interface by either using a WSDL that is generated using the operation name and schemas that you specify later in the wizard or by choosing an existing WSDL.

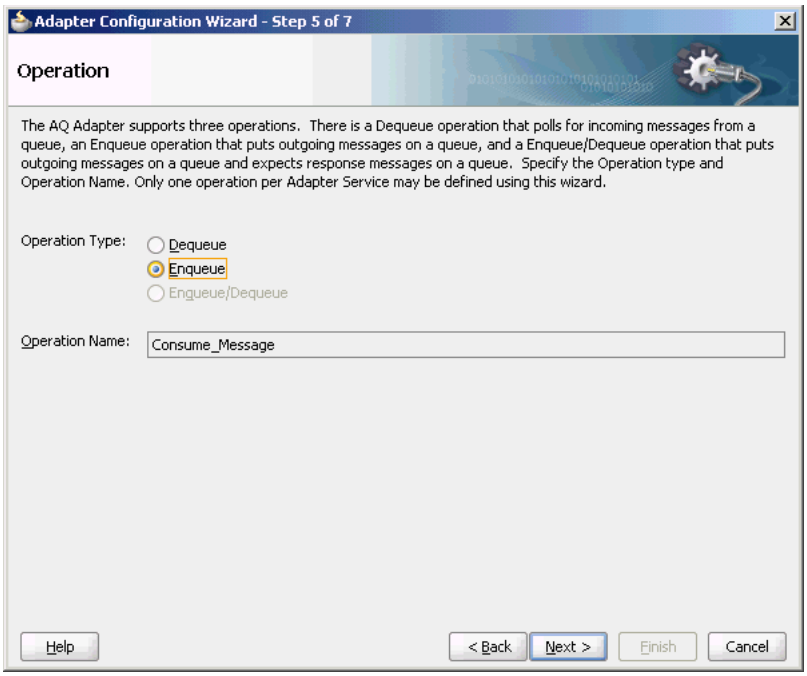
****

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

- Oracle File Adapter
- Oracle FTP Adapter
- Oracle Socket Adapter
- Oracle AQ Adapter
- Oracle JMS Adapter
- Oracle MQ Series Adapter

If you select the option of defining the adapter interface by importing an existing WSDL, then some functionalities on subsequent wizard pages are disabled. For example, since the WSDL defines the operation name and the message schema, the subsequent operation name and schema element fields are automatically filled in and you cannot modify it, as shown in Figure 2–2. However, if you do not choose to use an existing WSDL, then the adapter wizards behaves exactly as before.

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



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

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

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

2.3.1.1 Example of Use of Callbacks

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

If a callback is *not* selected, only the Put Message into MQ and Get Message from MQ options are enabled.

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

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

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

2.4 Configuring Message Header Properties for Oracle JCA Adapters

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

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

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

Table 2–1 Oracle JCA Adapters Property Types

Property Type	Description	Restrictions
Activation specification and interaction specification	Activation specification properties operate as services and interaction specification properties operate as references in a SOA composite application.	Do *not* add or remove these properties. You can only change their values. These properties require the adapter endpoint to be recycled. These types of properties are also dependent upon other properties. If you attempt to add properties, you have no way of knowing which dependent properties must also be added.
Endpoint	These are tuning-related properties that are not exposed through the activation or interaction specification properties, such as specifying time outs, thresholds, maximum intervals, and so on.	There are no restrictions on adding, removing, or changing endpoint properties. The adapter is notified when these properties are added, removed, or changed, but it does not require redeployment. You cannot add or remove `jca.retry.*` endpoint properties without redeploying the composite. However, you can change these properties by using the Fusion Middleware Control Console without redeploying the composite.

For more information, see Appendix A, "Oracle JCA Adapter Properties".

2.5 Physically Deploying Oracle JCA Adapters

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

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

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

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

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

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

For more information, see:

- Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters"

- Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper"
- Section 2.8, "Manually Deploying an Adapter RAR File that Does Not Have a Jar File Associated With It"
- Section 2.9, "Handling the Deployment Plan When Working on a Remote Oracle SOA Server"
- Section 2.10, "Migrating Repositories from Different Environments"
- Section 2.18, "Adding an Adapter Connection Factory"
- Section 2.20, "Recommended Setting for Data Sources Used by Oracle JCA Adapters"
- *Oracle Fusion Middleware Installation Guide for Oracle WebLogic Server*.

2.6 Creating an Application Server Connection for Oracle JCA Adapters

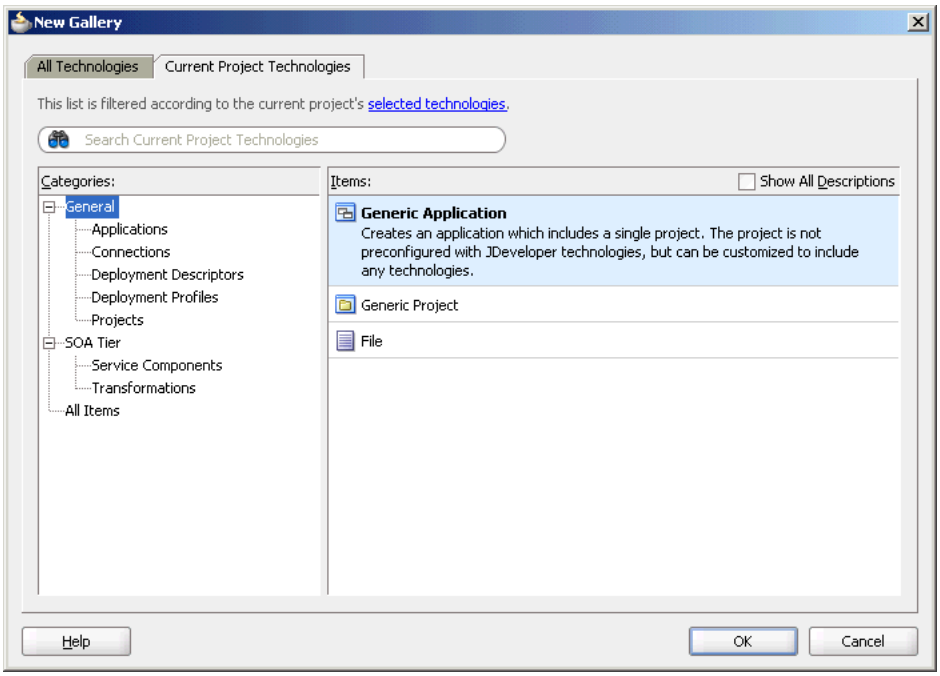
You must establish connectivity between the design-time environment and the server to which you want to deploy. To establish such connectivity, you must create an application server connection.

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

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

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

Figure 2–3 The New Gallery Page



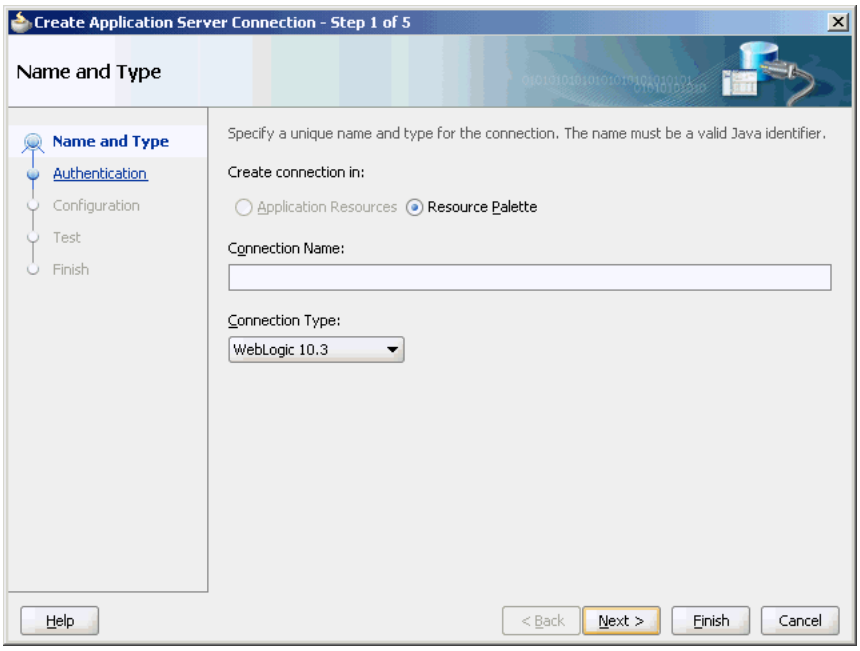
2. In the **All Technologies** tab, under **General** categories, select **Connections**.

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

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

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

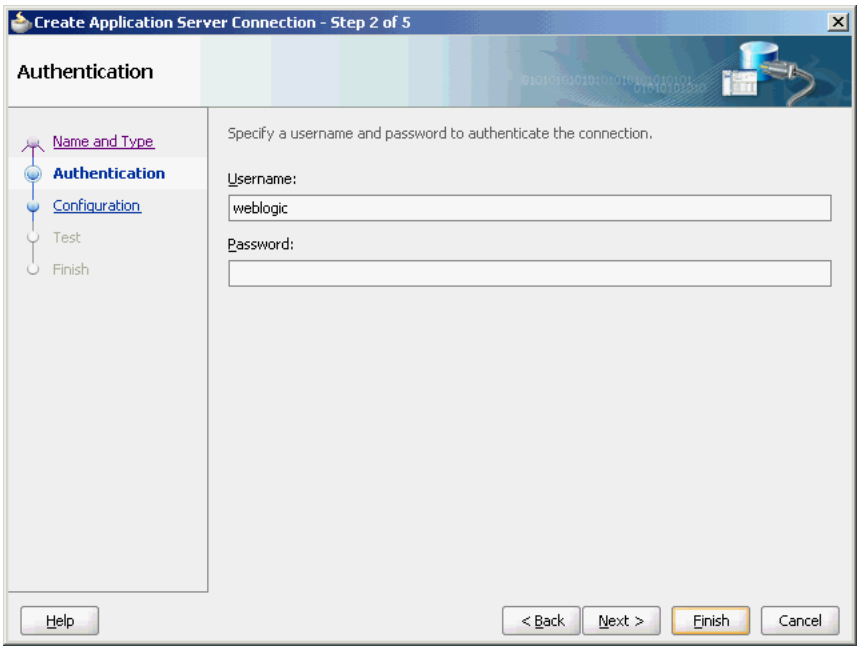
Figure 2–4 The Create Application Server Connection Name and Type Page



4. Enter a connection name in the **Connection Name** field. For example, `AppsServerConnection1`.
5. Select **WebLogic 10.3.** for Connection Type and click **Next**.

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

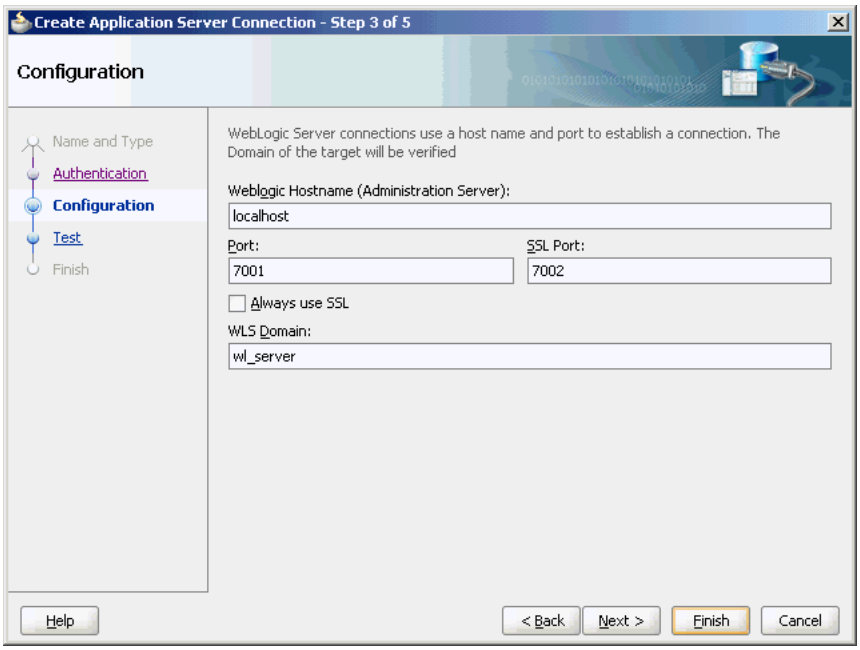
Figure 2–5 The Create Application Server Connection Authentication Page



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

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

Figure 2–6 The Create Application Server Connection Configuration Page

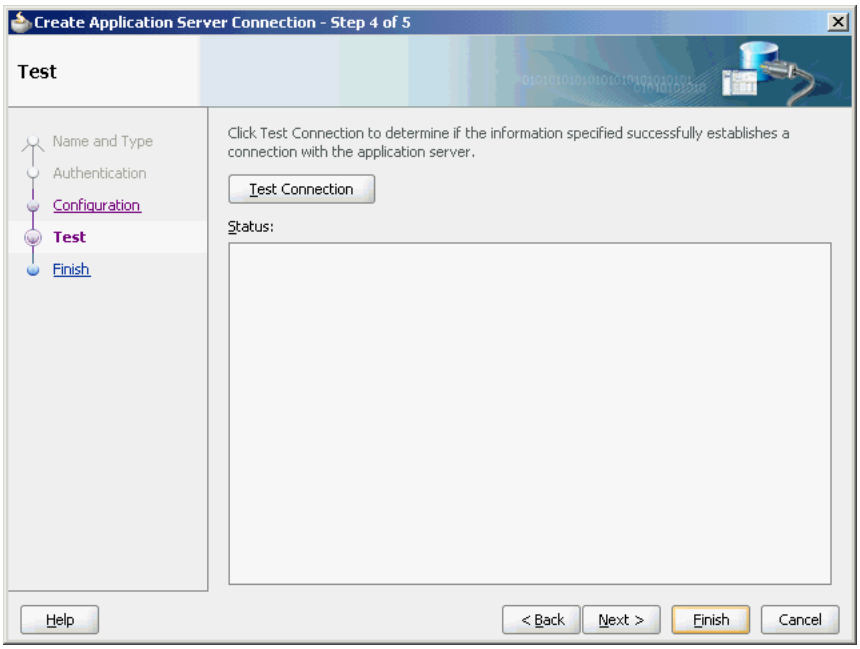


7. Enter the host name, the port details, and the domain server name in the Configuration page.

8. Click **Next**.

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

Figure 2–7 The Create Application Server Connection Test Page



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

    You have created a server connection.

2.7 Deploying Oracle JCA Adapter Applications from JDeveloper

You deploy an SOA composite application from JDeveloper.

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

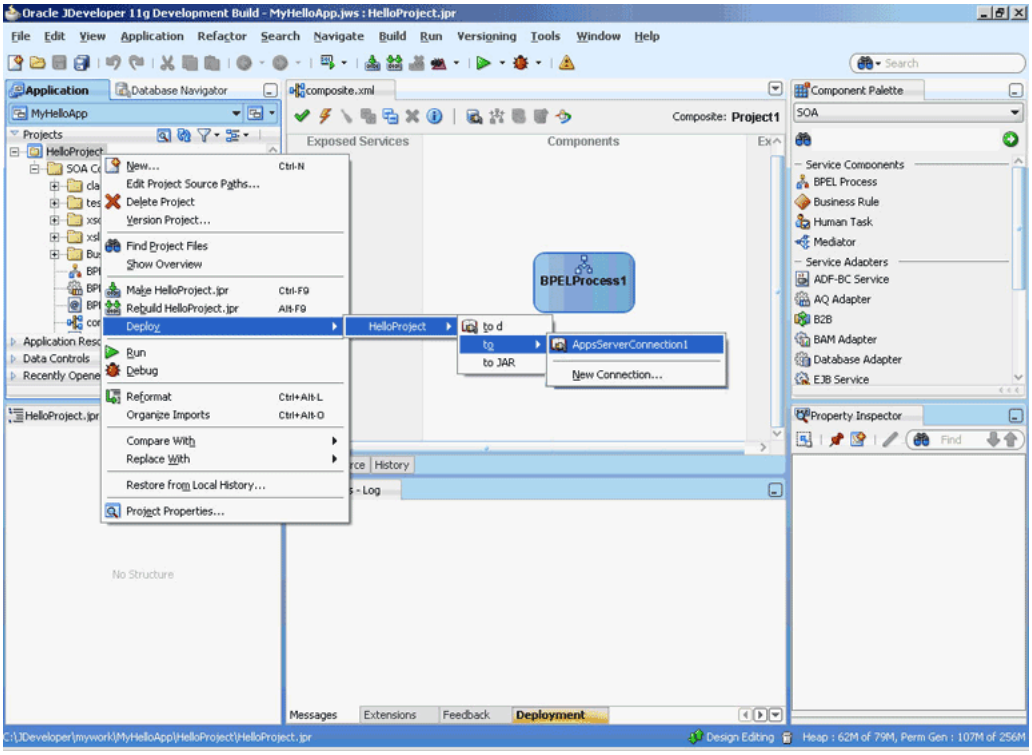
2.7.1 Deploying an Application Profile for the SOA Project and the Application

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

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

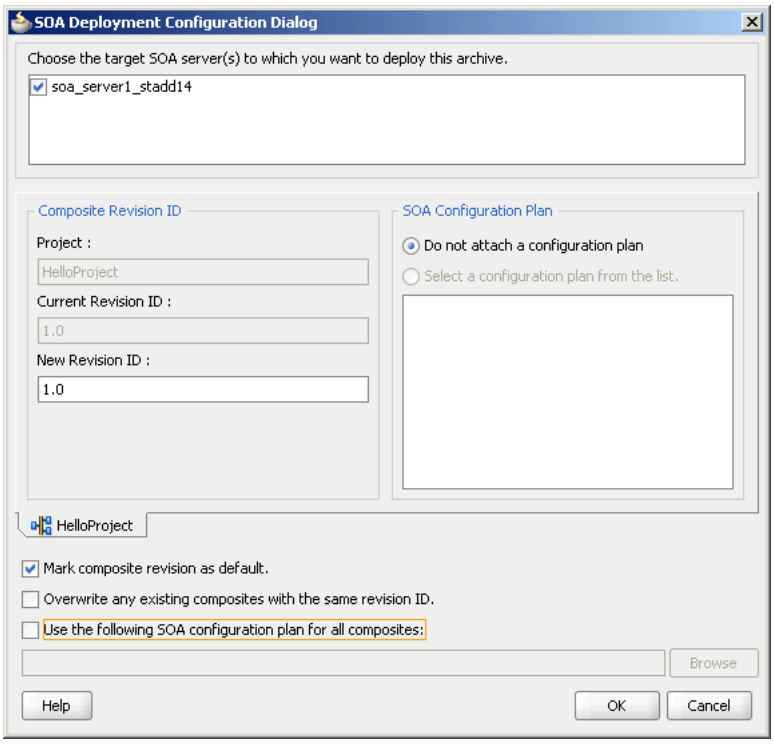
   The SOA Deployment Configuration dialog is displayed.

Figure 2–8 Application Profile Deployment



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

Figure 2–9 The SOA Deployment Configuration Dialog



3. Click **OK**.

The Authorization request dialog is displayed.

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

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

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

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

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

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

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

2.8.1 Example of Manual Deployment

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

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

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

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

The Home page of the Oracle WebLogic Server Administration Console is displayed.

3. Select **Deployments** in the Domain Structure pane.

   The Oracle WebLogic Server Administration Console Summary of Deployments page is displayed.

4. Click **Install**.

   The Install Application Assistant page is displayed.

5. Enter the path of the application directory or file in the **Path** field, and then click **Next**.

6. Select the servers to which you want to deploy this application, and then click **Next**.

   The Optional Settings page is displayed.

7. Modify these settings or accept the defaults, and then click **Next**.

   The Review your choices and click Finish page is displayed.

8. Click **Finish** to complete the deployment.

9. After you deploy the RAR file, under **Summary of Deployments**, click the name of the RAR file that you deployed.

   The Settings page is displayed.

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

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

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

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

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

2.10 Migrating Repositories from Different Environments

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

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

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

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

The `weblogic-ra.xml` identifies the end point as a development EIS or testing EIS or production EIS. For example, consider that when running through the Database

Adapter Service Wizard, you specify `eis/DB/custStore` as the JNDI name for the `createCustomer` service.

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

To summarize:

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

2.11 How Oracle JCA Adapters Ensure No Message Loss

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

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

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

This section describes:

- Section 2.11.2, "Local Transactions and Global (XA) Transactions"
- Section 2.11.3, "Basic Concepts of Transactions and Adapters"
- Section 2.11.5, "Outbound Transactions"

For more information, see:

- Section 2.12, "Composite Availability and Inbound Adapters"
- Section 3.1.1, "Oracle WebLogic Server Overview"
- Section 5.9.1.2 "Oracle JCA Adapters Reliability and Transactional Behavior" in the *Oracle Fusion Middleware High Availability Guide*.

2.11.1 XA Transaction Support

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

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

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

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

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

2.11.2 Local Transactions and Global (XA) Transactions

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

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

2.11.2.1 Adapter Support of Local Transactions

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

2.11.2.2 Adapter Support of Global Transactions

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

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

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

2.11.2.2.1 Global Transactions, Retries and Rollbacks and Fault Policies

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

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

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

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

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

2.11.3 Basic Concepts of Transactions and Adapters

For additional information on topics related to retryability, see Section 2.21.1, "Handling Rejected Messages," and following sections.

- **Polling**: All Oracle JCA Adapters and legacy adapters, support a pull, or polling, model for connecting to the back-end application for receiving events, that is, periodically querying the EIS endpoint for available messages and data.The exception to this is the Oracle Socket Adapter, which uses a different set of logistics, where the socket adapter can either connect to the EIS endpoint as the other adapters do using a client socket (polling), or, alternatively, create a server socket and then wait for incoming requests (push.) With polling, connection-related issues are recoverable and the inbound adapters keep retrying until the adapters are able to establish connection with the EIS. The adapter endpoints attempts to recover a lost connection for the duration of the active life of the composite. During this time they also update the log with diagnostics pinpointing the issue with connection
- **Local retry**: These are typically transient connectivity errors. where retries can be tried again and data is not compromised by a retry. However, non-successive local retries can change transaction state. Examples of retryable errors include temporary permission errors or resource constraint errors, or both. If a transaction can be retried, this does not necessarily mean a rollback.
- **Global retry**: A transaction that is rolled back to the beginning of the composite, for example, to a BPEL Receive where BPEL is part of the composite, which is at the beginning of the BPEL flow within a composite application. The transaction can be retried as indefinitely, or as many times as `jca.count.retry` indicates. Prior to the retry, a rollback can occur. An example could be where there is a BPEL fault in a synchronous process, or where there is a partial update to a database with master and child records and a temporary database fault occurs, and the toplink mapping logic decides a retry is acceptable. In other words, a global retry can occur if data is not tainted and it can be considered an explicit retry, where a rollback is needed.
- **Not-retriable**: A transaction that is not retried. With not-retriable conditions, there is no change to existing state. No-retry conditions derive from binding faults. Not-retriable situations typically occur where database integrity is an issue. Hence, not-retriable transactions are rolled back, when rejected; they are typically related to database constraint issues. Errors such as "Data already exists" (for example, Primary Key Errors) are not retriable as are message correlation ID errors. A list of errors that are not retriable is provided later in this chapter.
- **Inbound transaction**: A transaction initiated by an inbound adapter. For example, a transaction entering the SOA system from a JMS system.

- **Outbound transaction**: A transaction outbound from the SOA system (and hence from an adapter). For example, a transaction that is made against a database outside the SOA system.
- **JTA transaction**: Every step of a process is executed within the context of a JTA transaction. A JTA transaction ensures that one or more operations execute as an atomic unit of work. See the section on XA above.
- **Asynchronous transaction**: A composite transaction composed of sub-transactions. However, these sub-transactions are consecutive and serialized, that is, some sub-transactions may have been committed while others may be still executing or have not yet executed. Asynchronous transactions are guaranteed to be propagated once and only once. When an update at the source is committed, the transaction commits and expects that the update is propagated to the target appropriately.
- **Synchronous transaction**: These are transactions that execute in one thread from one endpoint to another, without intermediate processes, and which are not serialized.

2.11.3.1 Asynchronous Transaction Flow

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

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

2.11.3.1.1 Example using JMS, BPEL, DB Adapter and a Database

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

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

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

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

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

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

2.11.3.2 Synchronous Transaction Flow

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

- Message delivery
- Composite execution

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

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

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

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

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

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

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

2.11.4 Inbound Transactions

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

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

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

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

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

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

2.11.5 Outbound Transactions

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

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

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

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

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

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

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

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

2.12 Composite Availability and Inbound Adapters

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

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

For more information, see:

- Section 2.13, "Singleton (Active/Passive) Inbound Endpoint Lifecycle Support Within Adapters"
- File and FTP adapters: Section 4.2.18, "High Availability"
- Database adapter: Section 9.3.7, "High Availability"
- MQ adapter: Section 10.3.4, "High Availability"
- Section 2.11, "How Oracle JCA Adapters Ensure No Message Loss"
- Section 5.9 "Oracle JCA Adapters and High Availability Concepts" in the *Oracle Fusion Middleware High Availability Guide*.

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

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

To enable this fail over feature for a given inbound adapter endpoint, you must add the `singleton` JCA service binding property in the `composite.xml` within the `<binding.jca>` element and set it to a value of `true` as Example 2–1 shows.

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

Example 2–1 singleton Property in composite.xml

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

2.13.1 Multiple Activations of the Same Adapter Endpoint

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

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

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

2.13.2 Hot-Standby State

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

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

For more information, see Section 2.12, "Composite Availability and Inbound Adapters".

2.14 Correlation Support Within Adapters

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

For example, the following composite defines such a correlation:

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

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

The correlation between the request and the response is done transparently by the JCA binding run-time.

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

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

2.14.1 CorrelationID of Receive Message Not Matching Invoke: Log Error Message

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

In this case, although the message is persisted in the database, the `SEVERE` log message can occur, as Example 2–2 shows:

Example 2–2 Log Error When CorrelationId of the Receive Does not Match any Earlier Invoke

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

2.14.1.1 Rejecting Nonmatching Native Correlation IDs

You can explicitly alter the adapter framework behavior so that it *rejects* nonmatching native correlation IDs by adding the `rejectUncorrelatedMessages` service binding property to the `composite.xml` file as shown in Example 2–3.

Example 2–3 Setting the rejectUncorrelatedMessages Property

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

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

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

For more information, see:

- Section 2.21, "Error Handling"
- Section 8.2, "Oracle JMS Adapter Features"
- Section 7.2.5, "Normalized Message Support"
- Appendix A, "Oracle JCA Adapter Properties".

2.15 Setting Payload Size Threshold

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

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

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

2.15.1 Payload Native Size

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

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

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

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

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

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

2.15.1.1 Setting the Payload Threshold

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

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

2.15.1.2 Limitations on Payload Size Enforcement

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

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

Table 2–2 Translation Scenarios Supported for Oracle JCA Adapters

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

2.15.1.2.1 Changing Global Payload Size to a Finite Value

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

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

2.16 Streaming Large Payload

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

- Oracle File Adapter

  For more information, see Section 4.5.4, "Oracle File Adapter Scalable DOM".

- Oracle AQ Adapter

  For more information, see Section 7.2.11, "Stream Payload Support".

- Oracle JMS Adapter

  For more information, see "Supports Streaming Large Payload" on page 8-5.

- Oracle Database Adapter

  For more information, see Section 9.3.5, "Streaming Large Payload".

The other adapters do not have explicit support for both.

2.17 Batching and Debatching Support

The batching and debatching functionality is supported for these adapters:

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

Oracle JCA Adapter for File and Oracle JCA Adapter for FTP consist of a `Reader` to debatch a single large file into several batches. You must specify the batch size during the design-time configuration. In addition, the adapter includes a Writer to batch a set of messages into a single file. For more information, see Section 4.2.4, "File Debatching".

Oracle JCA Adapter for Databases consists of a Publish component to poll a set of tables to detect events. This component can raise events to the BPEL process one record at a time or multiple records at a time. For more information, see Section 9.4.2.2, "Polling Strategies".

2.18 Adding an Adapter Connection Factory

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

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

For more information about creating a connection factory, see *Oracle Fusion Middleware Installation Guide for Oracle WebLogic Server*.

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

This section includes the following topics:

- Section 2.18.1, "Creating a Data Source"
- Section 2.18.2, "Creating a Connection Pool"

2.18.1 Creating a Data Source

To create a data source:

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

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

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

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

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

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

2.18.2 Creating a Connection Pool

To create a connection pool:

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

2. Use the required credentials to open the Home page of the Oracle WebLogic Server Administration Console.

   The Home page of the Oracle WebLogic Server Administration Console is displayed.

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

   The Summary of Deployments page is displayed.

4. Click the Database adapter name from the Deployments list.

   The Settings for DbAdapter page is displayed.

5. Click **Configuration** tab, and then click **Outbound Connection Pools** tab.

   The Outbound Connection Pool Configuration Table is displayed.

6. Click **New**.

7. Select **javax.resource.cci.ConnectionFactory**, and then click **Next**.

   The Create a New Outbound Connection page is displayed.

8. In the **JNDI Name:** field, enter `eis/DB/soademoDatabase`.

   Note: The JNDI value that you enter in this step is different from the same value that you entered in Step 5 in Section 2.18.1, "Creating a Data Source." The JNDI name specified in this step must match the value you enter in your database connection you create when building your application later.

9. Click **Finish**.

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

   The configuration changes that you made must be stored in a new deployment plan. You do this in the next step.

10. In the **Path** field, select or enter the path of a deployment plan file. The path must end with ".xml".

    Note: If the Adminserver is running on computer A and the Oracle SOA server is running on computer B, then you *must copy* the deployment plan file to computer B before you activate changes made on the Oracle SOA server. If you try to activate changes *without* copying the deployment plan to the Oracle SOA Server computer, a `NullPointerExceptwion` is thrown.

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

12. Click **Save**.

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

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

    The Summary of Deployments is displayed.

14. Perform the following steps:

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

       The Update Application Assistant page is displayed.

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

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

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

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

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

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

2.19 Adding or Updating an Adapter Connection Factory

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

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

2.19.1 Modify the JCA File

Follow these steps:

1. Create a new JNDI for a JCA adapter connection factory. For more information about creating a connection factory, see *Oracle Fusion Middleware Installation Guide for Oracle WebLogic Server*.
2. Modify the JCA file of the deployed composite to point to the new JNDI.

The composite takes the properties from the newly-created JNDI.

2.19.2 Use a Config Plan

1. Create a new JNDI for a JCA adapter connection factory.
2. Create a Config plan for the composite.

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

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

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

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

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

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

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

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

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

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

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

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

2.19.3 Use the Web Logic Server Console to Create a New Connection

You can use the Web Logic Console to create connection factories for use with JMS. Refer to Section 8.4.1.4.1, "Creating a New Connection by Using the Oracle WebLogic Server Administration Console"

2.20 Recommended Setting for Data Sources Used by Oracle JCA Adapters

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

The following are the recommended settings for multi data sources:

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

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

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

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

XA Data Sources	Non-XA Data Sources
The driver used is `oracle.jdbc.xa.client.OracleXADataSource`.	The driver used is `oracle.jdbc.OracleDriver`.
The JDBC URL should be in the following format: `jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=host-vip)(PORT=1521))(CONNECT_DATA=(SERVICE_NAME=service_name)(INSTANCE_NAME=inst1)))`	Same as that of XA data source.

You must set the following property
`<property>`
`<name>oracle.net.CONNECT_TIMEOUT</name>`
`<value>10000</value>`
`</property>`
	Same as that of XA data source.

The value of `initial-capacity` must be 0	Same as that of XA data source.
The value of `connection-creation-retry-frequency-seconds` must be 10	Same as that of XA data source.
The value of `test-frequency-seconds` must be 300.	Same as that of XA data source.
The value of `test-connections-on-reserve` must be `TRUE`.	Same as that of XA data source.
The value of `test-table-name` must be `SQL SELECT 1 FROM DUAL`	Same as that of XA data source.
The value of `seconds-to-trust-an-idle-pool-connection` must be 0	Same as that of XA data source.
The value of `global-transactions-protocol` must be `TwoPhaseCommit`	The value for `global-transactions-protocol` must be `None`.
The value of `keep-xa-conn-till-tx-complete` must be `TRUE`.	NA
The value of `xa-retry-duration-seconds` must be 300.	NA
The value of `xa-retry-interval-seconds` must be 60.	NA

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

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

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

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

2.21 Error Handling

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

This section includes the following topics:

- Section 2.21.2, "Inbound Interaction Error Handling"
- Section 2.21.3, "Outbound Adapter Interaction Error Handling"
- Section 2.21.1.2.3, "Handling Message Errors: A Sample Scenario"

2.21.1 Handling Rejected Messages

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

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

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

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

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

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

This section includes the following topics:

- Section 2.21.1.1, "Configuring Rejection Handlers"
- Section 2.21.1.2, "Checking for Rejected Messages"

For more information, see Section 2.14, "Correlation Support Within Adapters".

2.21.1.1 Configuring Rejection Handlers

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

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

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

2.21.1.1.1 Creating Fault Policies

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

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

   The following is an example of a fault policy:

```
<?xml version="1.0" encoding="UTF-8"?>
  <faultPolicies>
    <faultPolicy version="2.0.1" id="RejectedMessages">
      <Conditions> <!-- All the fault conditions are defined here -->
        <faultName xmlns:rjm="http://schemas.oracle.com/sca/rejectedmessages"
name="rjm:<SERVICE_NAME>"> <!-- local part of fault name should be the service
name-->
          <condition>
           <action ref="writeToFile"/> <!-- action to be taken, refer to
Actions section for the details of the action -->
          </condition>
        </faultName>
      </Conditions>
      <Actions> <!-- All the actions are defined here -->
          <Action id="writeToFile">
            <fileAction>
              <location>/tmp/rej_msgs</location>
              <fileName>emp_%ID%_%TIMESTAMP%.xml</fileName>
            </fileAction>
          </Action>
      </Actions>
    </faultPolicy>
  </faultPolicies>
```

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

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

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

Note: If you do not configure rejection handlers as mentioned in Section 2.21.1.1, "Configuring Rejection Handlers", a default file-based rejection handler starts processing and the rejected messages is directed to `<domain_home>/rejmsgs/<wls_server_name>/<composite_name>`.

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

2.21.1.2 Checking for Rejected Messages

Rejected messages are stored in the `rejected_message` table.

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

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

2.21.1.2.1 Checking from the Database

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

```
select * from rejected_message
```

2.21.1.2.2 Checking from the Fusion Middleware Control Console

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

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

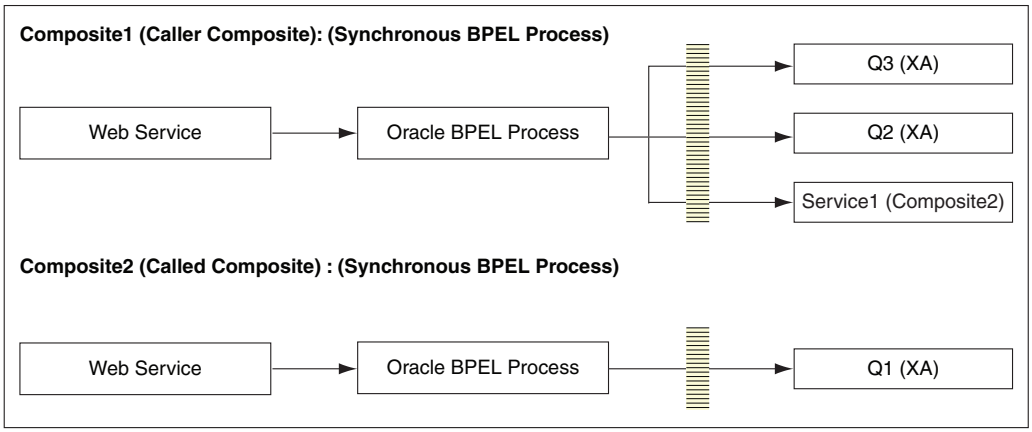
- Section 28.2, "Monitoring Recent Faults and Rejected Messages for an Inbound Adapter" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.
- Section 28.3, "Monitoring Faults and Rejected Messages for an Inbound Adapter" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*

2.21.1.2.3 Handling Message Errors: A Sample Scenario

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

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

Figure 2–10 Sample Scenario: Handling Message Errors



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

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

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

To have the transaction roll back *all the queues even if only one fails, and for the other two have messages successfully delivered to them*, you must make the change in the `composite.xml` file of the called composite (Composite2) as Example 2–4 shows:

Example 2–4 Changes in composite.xml of Composite2

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

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

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

2.21.2 Inbound Interaction Error Handling

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

2.21.2.1 Message Error Rejection Handlers

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

2.21.2.1.1 Available Rejection Handlers for Message Errors

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

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

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

2.21.2.1.2 Web Service Handler

A rejected message can be handled by calling a Web Service. If you choose to use a Web Service to handle these errors, you should implement a predefined WSDL interface implemented by the target service, SOAP bindings for the Web service invocaiton, and native payloads passed as WebService-attachments, as shown in the following example:

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

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

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

2.21.2.1.3 Custom Java Handler

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

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

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

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

2.21.2.1.4 JMS Queue

You can enqueue a rejected message to a JMS queue as a JMS message with the appropriate context and payload, as shown in the following two examples.

The first example uses a standalone database:

```
<Action id="ora-queue">
  <enqueue uri="QueueURI"/> <!-- QueueURI format  -
jdbc:oracle:thin:@<host>:<port>:<sid>#<un>/<pw>#queue -->
</Action>
```

The second example is used with an Oracle RAC database:

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

2.21.2.1.5 File

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

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

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

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

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

2.21.2.2 Inbound Retryable Errors

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

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

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

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

2.21.2.2.1 Configuring Inbound Adapters to Handle Retryable Errors

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

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

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

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

- `jca.retry.count`

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

- `jca.retry.interval`

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

- `jca.retry.backoff`

  Specifies the retry interval growth factor (positive integer.)

- `jca.retry.maxInterval`

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

2.21.2.2.2 Specifying Inbound Retry Properties in the composite.xml File

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

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

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

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

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

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

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

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

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

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

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

2.21.2.2.4 Global Property Modification using the MBeans Browser

To modify the *global property* using the MBeans browser (Adapter Mbean) of the Oracle Enterprise Manager, you must use the following procedure:

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

   The Fusion Middleware Control Console displays its home page.

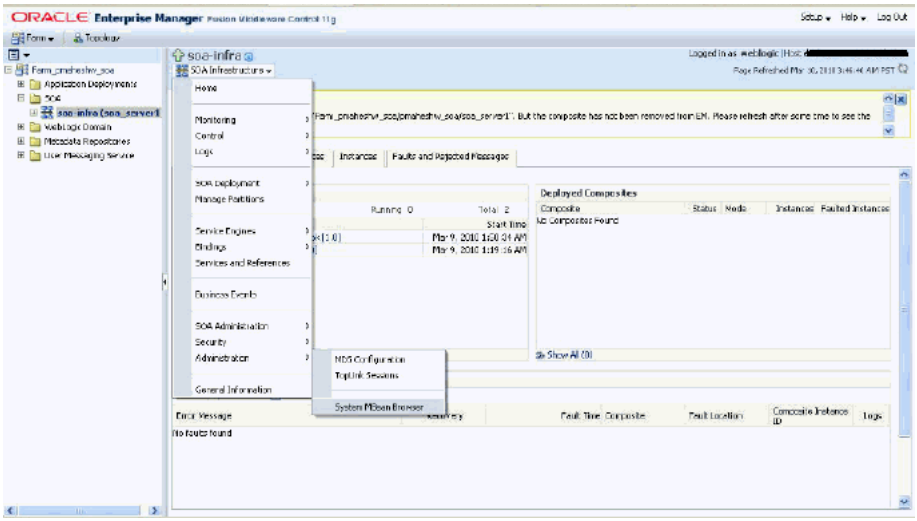
2. Right-click **soa-infra** from the SOA Folder in the navigator in the left pane.

   The soa-infra page is displayed.

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

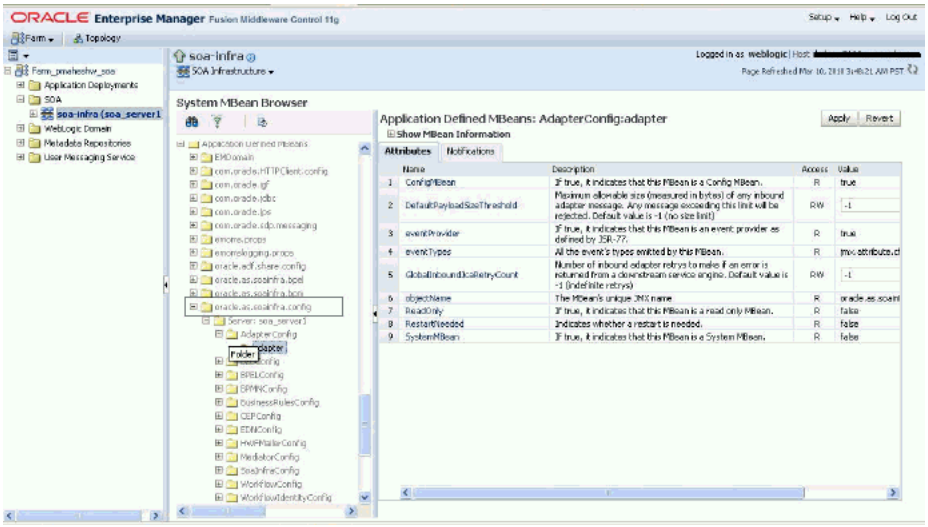
   The System Mbean Browser page is displayed.

Figure 2–11 The soa-infra Page



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

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



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

2.21.2.3 Inbound Non-Retryable Errors

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

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

2.21.2.3.1 Examples of Non-Retryable Errors

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

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

Non-retryable errors do not resolve themselves until after the operation is retried. For example, messages can be sent from a file to an inbound file adapter through a Mediator. The Mediator, in turn, has sequential routing to an outbound Database Adapter that inserts data to a database table. The DB adapter might encounter a unique *constraint error* as it is performing the insert operation. This unique constraint error is:

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

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

2.21.3 Outbound Adapter Interaction Error Handling

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

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

2.21.3.1 Retryable Errors for Outbound Adapter Error Handling

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

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

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

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

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

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

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

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

```
<reference name="Outbound">
  <interface.wsdl interface="http://xmlns...#wsdl.interface(Outbound_PortType)"/>
  <binding.jca config="Outbound_jms.jca">
    <property name="jca.retry.count">5</property>
```

```
    <property name="jca.retry.interval">1</property>
    <property name="jca.retry.backoff">2</property>
    <property name="jca.retry.maxInterval">6</property>
    <property name="jca.retry.maxPeriod">30</property>
  </binding.jca>
</reference>
```

2.21.3.2 Non-Retryable Errors for Outbound Interaction Handling

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

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

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

All time measurements are specified in seconds.

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicyBindings version="2.0.1"
xmlns="http://schemas.oracle.com/bpel/faultpolicy"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <reference faultPolicy="ConnectionFaults">
      <name>writeMessageToQueue</name>
    </reference>
</faultPolicyBindings>
```

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

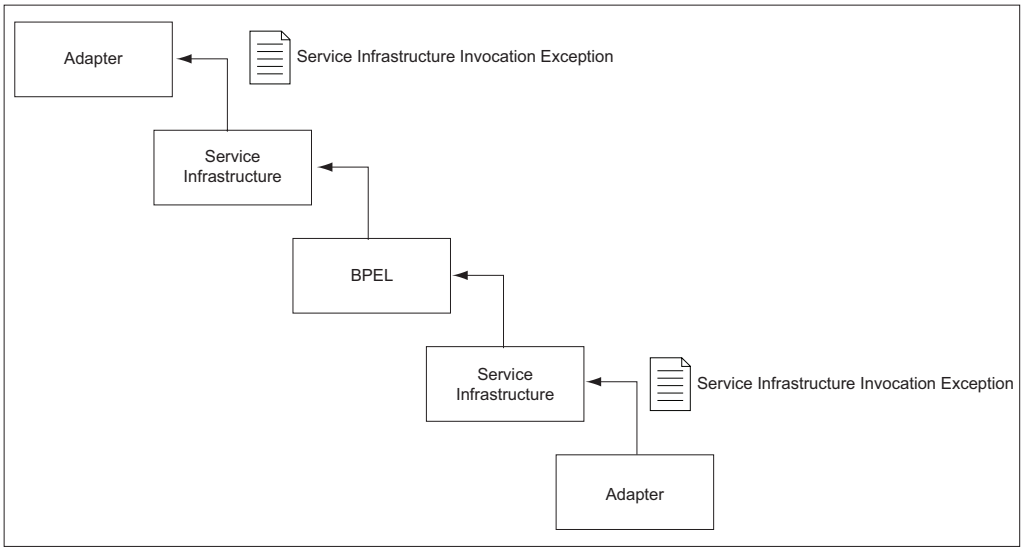
2.21.3.2.1 Fault Propagation

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

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

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

Figure 2–13 Fault Propagation



2.21.3.2.2 Two Cases When the Fault Policy Mechanism Does Not Work

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

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

2.21.3.2.3 Outbound Adapters in XA Mode

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

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

2.21.3.2.4 Outbound Adapter in Mediator Sequential Routing

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

2.22 Testing Applications

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

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

For more information about testing applications, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

2.23 Setting the Trace Level of Oracle JCA Adapters

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

- Oracle JCA Adapters and Oracle Adapter for Oracle Applications: set the log level to, for example, `TRACE:32` in the logger `oracle.soa.adapter`.

  For more information about setting trace levels for adapters, see *Oracle Fusion Middleware Administrator's Guide*.
- Packaged-application adapters: For outbound interactions, set the `Loglevel` property for packaged-application adapters in the `weblogic-ra.xml` file.
- Legacy adapters: you can use Oracle Studio to set the trace level for Oracle Connect, and the mainframe server.

2.23.1 How to Set the Trace Level of Oracle JCA Adapters

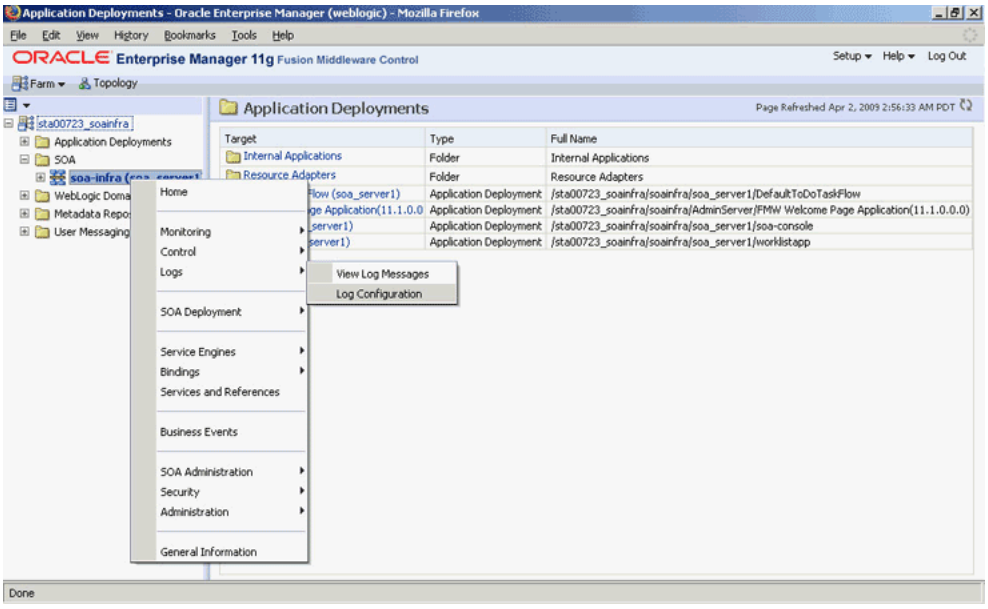
To set the trace level by using the Fusion Middleware Control Console:

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

   The Fusion Middleware Control Console home page is displayed.
2. Right-click soa-infra from the SOA Folder in the Navigator in the left pane.

   The console displays a menu.
3. Select Logs, and the Log Configuration, as shown in Figure 2–14.

Figure 2–14 Navigating to the Log Configuration Page

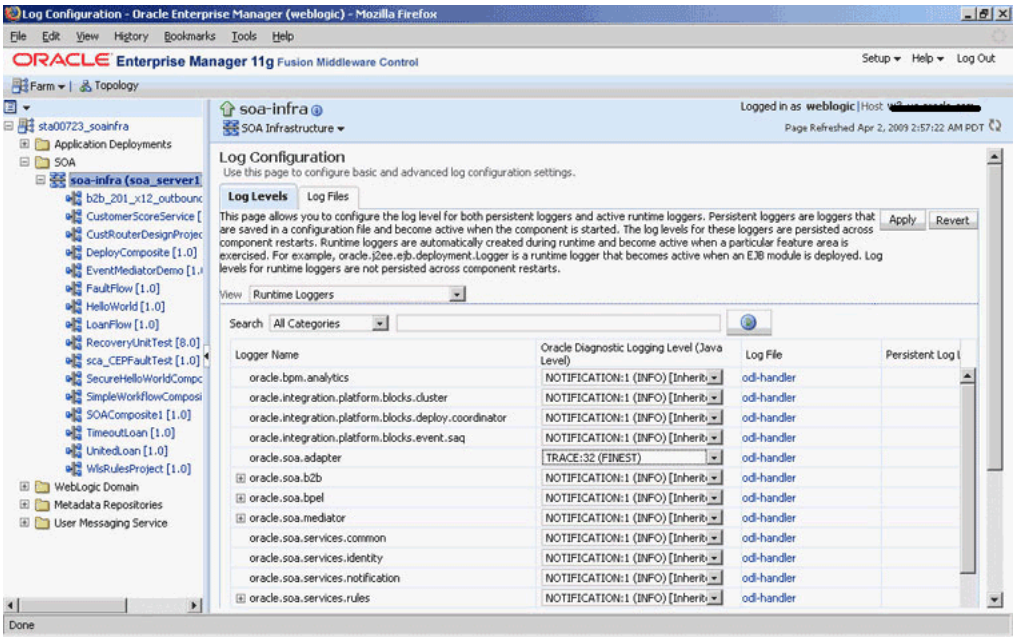


The Log Configuration page is displayed.

4. Locate **oracle.soa.adapter** in the Logger Name list, and change the log level in the Oracle Diagnostic Logging Level (Java Level) field. In this example, select **Trace:32 (FINEST)** from the list, as shown in Figure 2–15.

> **Note:** To ensure that log levels persist across component restarts, select Loggers With Persistent Log Level State from the View list. By default, the log level is set for run-time loggers. Run-time loggers do not persist across component restarts.

Figure 2–15 The Log Configuration Page



For more information, see Section 2.24, "Viewing Adapter Logs".

2.24 Viewing Adapter Logs

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

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

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

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

  For more information about searching and viewing log files, see *Oracle Fusion Middleware Administrator's Guide*.

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

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

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

For more information, see Section 2.23, "Setting the Trace Level of Oracle JCA Adapters".

2.25 Creating a Custom Adapter

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

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

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

2.25.1 Configuring a Custom Adapter

When you select SOA as an installable option with JDev, by default the Custom Adapter is not available. To ensure that the Custom Adapter is available, edit the<JDEV_HOME>\jdeveloper\integration\seed\soa\configuration\soa-config.xml file, search for "custom", and uncomment its <adapterType> element. The JDEV Component Palette displays the Custom Adapter for the SOA Diagram.

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

The properties within an activation-spec are properties that are specific to an inbound adapter. The properties within an interaction-spec are the properties specific to an outbound adapter. The property values are the default values shown by the Custom Adapter. See the screenshots below for examples.

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

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

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

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

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

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

   <endpoint-activation>
    <activation-spec className="oracle.tip.adapter.custom.inbound.
CustomActivationSpec" displayResourceKey="CustomActivationSpec">
      <property name="UseHeaders" value="false"/>
      <property name="PhysicalDirectory" value="x"/>
      <property name="Recursive" value="true"/>
```

```
      <property name="DeleteFile" value="true"/>
      <property name="IncludeFiles" value="x"/>
      <property name="PollingFrequency" value="60"/>
      <property name="MinimumAge" value="0"/>
    </activation-spec>
  </endpoint-activation>

</adapter-config>
```

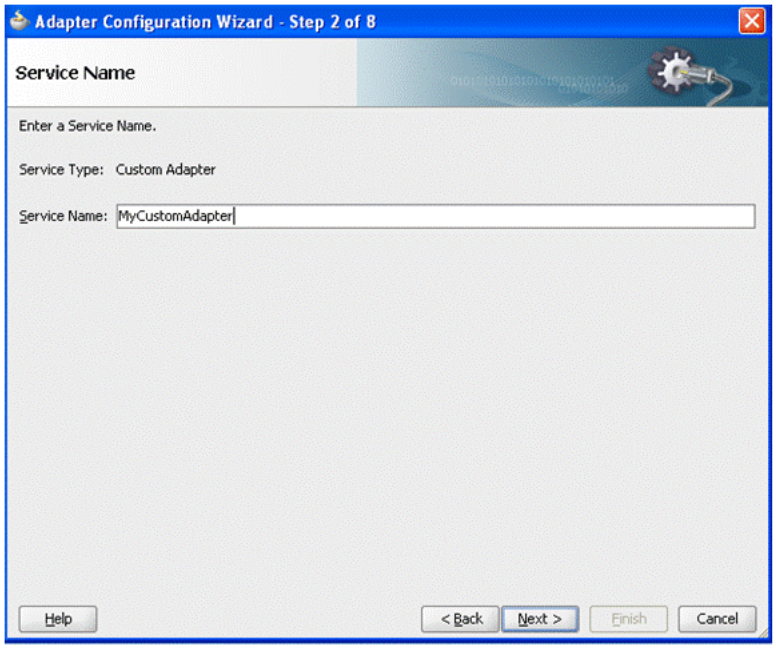
2.25.1.1 Custom Adapter Screen Flow

When you drag-and-drop the Custom JCA Adapter icon to the Exposed Service or External Reference swimlane within JDev, the IDE displays the Adapter Configuration Welcome Page. You can then select Next to begin the Custom Adapter Configuration Wizard workflow.

Figure 2–16 Adapter Configuration Wizard Welcome Screen

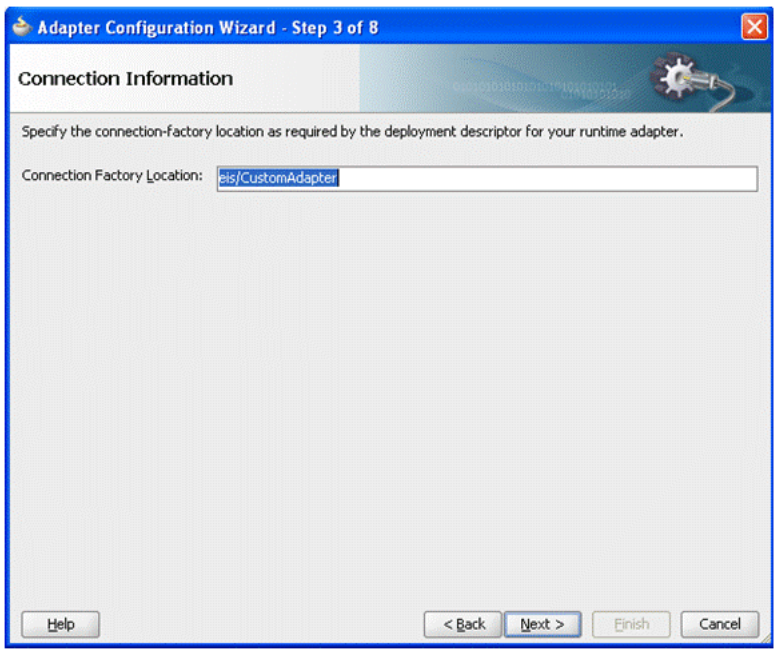
The next screen displays the service type and name, similar to the way it occurs with the Configuration Wizards of other adapters. This screen enables you to provide the name of a Service that makes sense in the Adapter you are configuring.

Figure 2–17 Adapter Configuration Wizard Service Name Screen



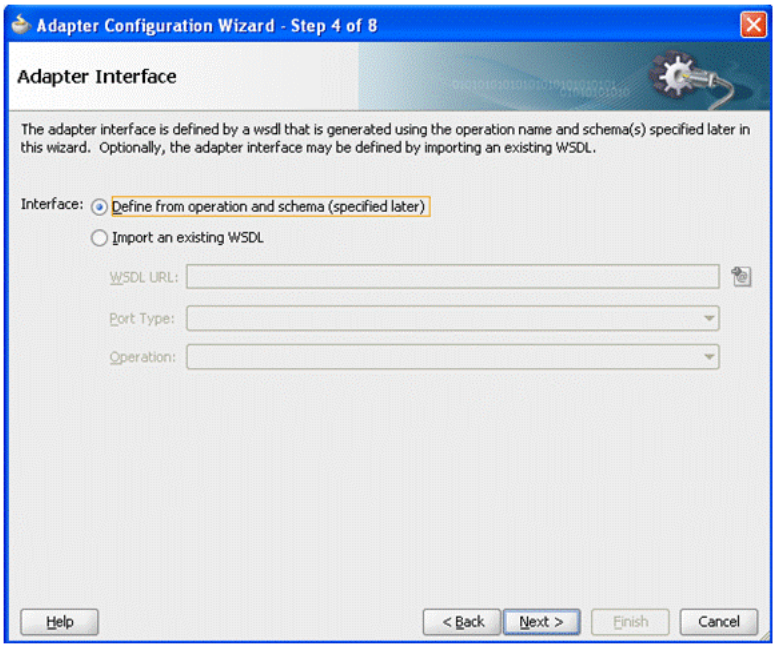
If the `config.xml` file contains a `<connection-factory>` entry (as required by the custom run-time adapter), the Wizard displays the Connection Information page displaying the default Connection Factory Location. If the `config.xml` does not contain a `<connection-factory>` entry (not required by the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

Figure 2–18 Adapter Configuration Wizard Connection Information Screen



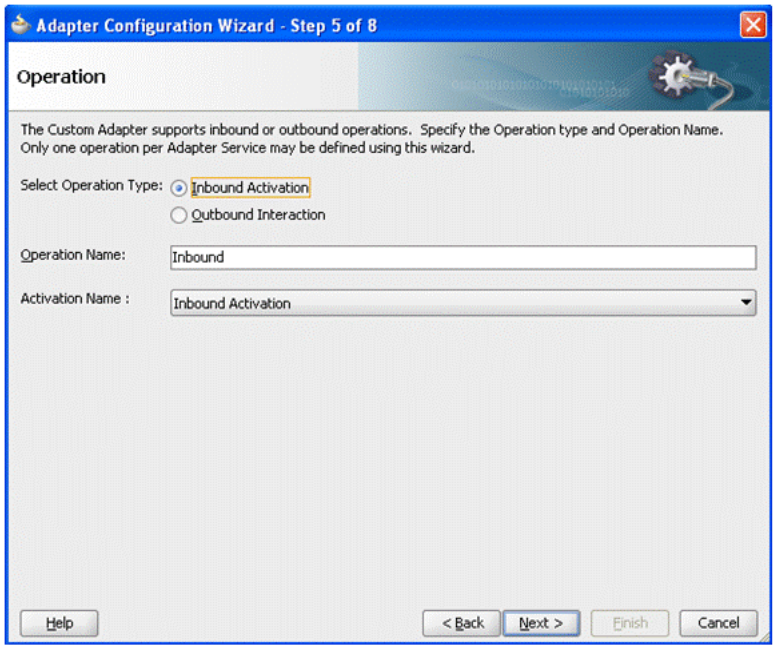
The next screen is the Adapter Interface Screen, which displays information in a similar manner to the configuration wizard for other Adapters. This screen provides you a way to either define a new WSDL from an operation and schema you provide later, or import an existing WSDL, using the WSDL name, port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

Figure 2–19 Custom Adapter Wizard Adapter Interface Screen



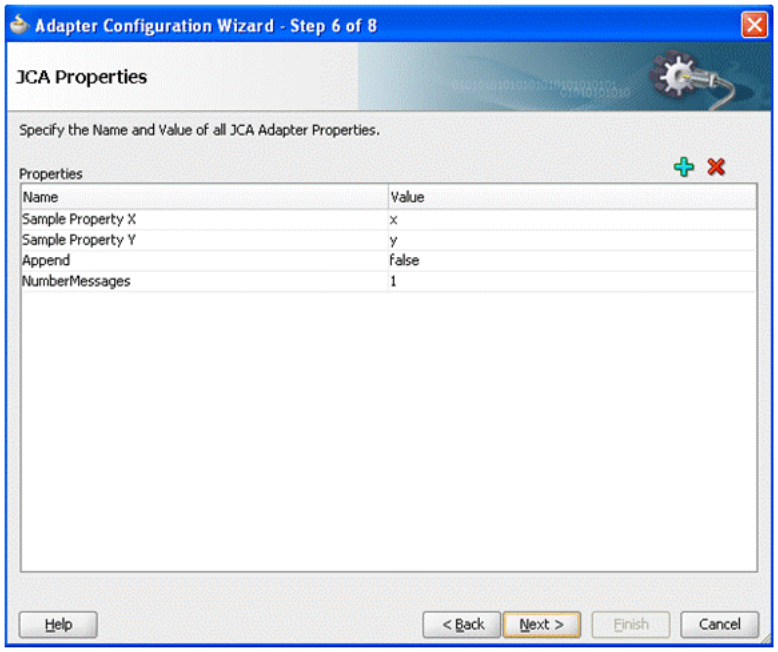
The next screen enables the user to choose the type of interaction: Inbound or Outbound. If Outbound Interaction is selected, the Wizard provides a list of Interaction Class names (or translated display names as seen in this example) from which to choose. You earlier provided these names in the `customAdapter-config.xml`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

Figure 2–20 Custom Adapter Configuration Wizard Operation Screen (Inbound Choice)



The following screen enables you to specify the name and value of JCA properties. Depending on the Class Name chosen, the screen displays the properties associated with that class in the `customAdapter-config.xml` file. You can use this screen to change any of the default values and to add or delete properties.

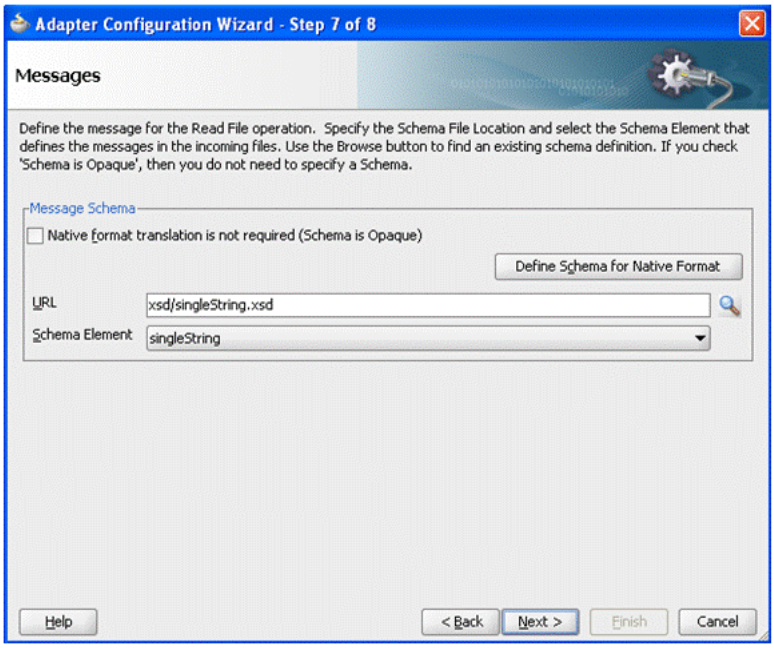
Figure 2–21 Custom Adapter Configuration Wizard JCA Properties Screen



The next screen is the Custom Adapter Wizard Messages Screen, which behaves in a way similar to that of other Adapter Configuration Wizards, enabling you to define

the message for the Read File operation, by either specifying a Schema or by declaring that the schema is opaque.

Figure 2–22 Customer Adapter Configuration Wizard Messages Screen



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

2.25.2 Frequently Asked Questions about Adapters

Following are some frequently asked questions about adapters.

2.25.2.1 Why are My Applications Timing Out?

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

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

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

2.25.2.2 How do Transactional and Non-Transactional Adapters Differ?

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

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

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

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

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

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

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

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

2.25.2.3 What Happened to My Application's Rejected Messages? Can Onedo Anything With Them?

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

3

Adapter Integration with Oracle Application Server Components

This chapter discusses how to integrate adapters with Oracle WebLogic Server and Oracle Fusion Middleware.

Oracle Application Server adapters can be integrated with various components of Oracle WebLogic Server and Oracle Fusion Middleware.

This chapter includes the following topics:

- Section 3.1, "Adapter Integration with Oracle WebLogic Server"
- Section 3.2, "Adapter Integration with Oracle Fusion Middleware"
- Section 3.3, "Monitoring Oracle JCA Adapters"

3.1 Adapter Integration with Oracle WebLogic Server

Oracle JCA Adapters are based on the J2CA 1.5 specification and are deployed to the Oracle WebLogic Server. The resource adapter is used within the address space of the Oracle Fusion Middleware. This section provides an overview of the Oracle WebLogic Server and design-time and run-time integration with an adapter.

This section includes the following topics:

- Section 3.1.1, "Oracle WebLogic Server Overview"
- Section 3.1.2, "Oracle WebLogic Server Integration with Adapters"

3.1.1 Oracle WebLogic Server Overview

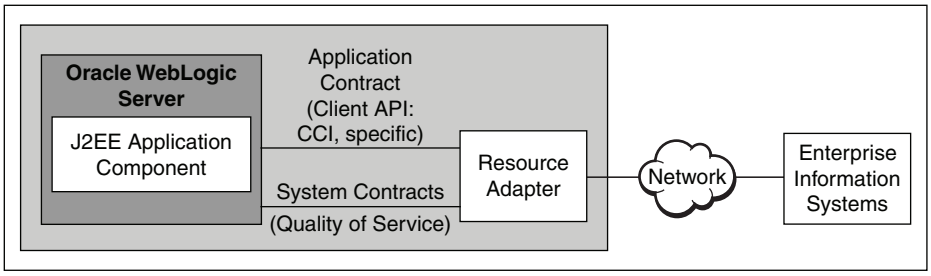
Oracle WebLogic Server is the core J2EE run-time component of Oracle Application Server. Oracle WebLogic Server is a scalable, enterprise-ready Java Platform, Enterprise Edition (Java EE) application server. The WebLogic Server infrastructure supports the deployment of many types of distributed applications. It is an ideal foundation for building applications based on Service Oriented Architecture (SOA).

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

On the other hand, the contract between the Oracle WebLogic Server container and the resource adapter is defined by the service provider interface (SPI). Contracts define a

standard between Oracle WebLogic Server and adapters. The system handles these contracts automatically and hides them from the application developer. Figure 3–1 illustrates the CCI and SPI contracts:

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



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

- Connection management: Enables application components to connect to a back-end application and leverage any connection pooling support of the Oracle WebLogic Server container. This leads to a scalable and efficient environment that can support a large number of components requiring access to a back-end application. For more information, see Section 2.18, "Adding an Adapter Connection Factory".
- Transaction management: Enables an application server to use a transaction manager to manage transactions across multiple resource managers. Most of the adapters support only local transactions (single-phase commit) and not XA transactions (two phase commit). For more information, see Section 2.11, "How Oracle JCA Adapters Ensure No Message Loss".

  The following adapters support XA transactions:

  – Oracle MQ Series Adapter
  – Oracle JMS Adapter
  – Oracle AQ Adapter
  – Oracle Database Adapter
  – Oracle EBS Adapter

  The following adapters do not support XA transactions:

  – Oracle File Adapter
  – Oracle FTP Adapter
  – Oracle Socket Adapter

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

3.1.2 Oracle WebLogic Server Integration with Adapters

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

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

3.1.2.1 Design Time

Use the adapter design-time tool to generate XML Schema Definition (XSD) files for the adapter request-response service. The Oracle WebLogic Server clients use these XSD files during run time for calling the J2CA outbound interaction. Packaged-application adapters use OracleAS Adapter Application Explorer (Application Explorer), Legacy adapters use OracleAS Studio, and technology adapters use Oracle JDeveloper (JDeveloper).

For more information, see Section 3.2.3.1, "Design Time".

3.1.2.2 Run Time

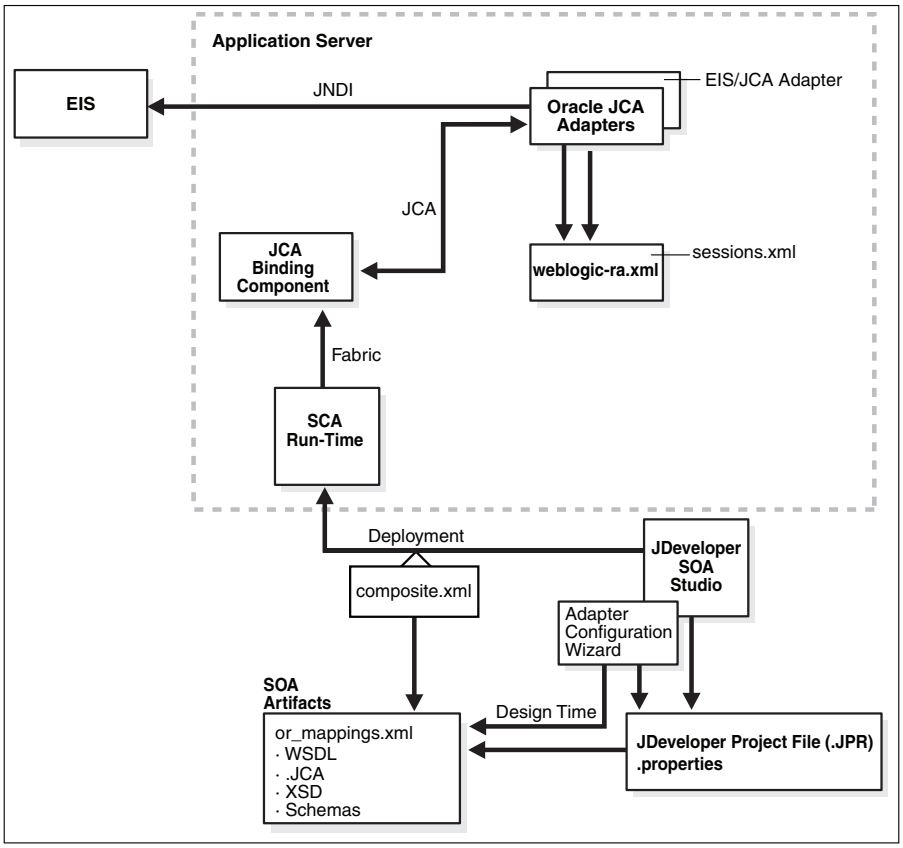
Oracle JCA Adapters are based on the J2CA 1.5 specification but are deployed as the J2CA 1.5 resource adapter within the Oracle WebLogic Server container in this release. The J2CA 1.5 specification addresses the life-cycle management, message-inflow (for Adapter Event publish), and work management contracts.

3.2 Adapter Integration with Oracle Fusion Middleware

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

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

Figure 3–2 Oracle Adapter Architecture in Oracle Fusion Middleware



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

Oracle technology adapters gather and publish statistics for every inbound and outbound message they process. For more information, see Section 3.3, "Monitoring Oracle JCA Adapters".

This section includes the follows topics:

- Section 3.2.1, "Oracle BPEL Process Manager Overview"
- Section 3.2.2, "Oracle Mediator Overview"
- Section 3.2.3, "Oracle Fusion Middleware Integration with Adapters"
- Section 3.2.4, "Oracle SOA Composite Integration with Adapters"

3.2.1 Oracle BPEL Process Manager Overview

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

3.2.2 Oracle Mediator Overview

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

3.2.3 Oracle Fusion Middleware Integration with Adapters

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

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

This section describes:

- Section 3.2.3.1, "Design Time"
- Section 3.2.3.2, "Run Time"
- Section 3.2.3.3, "End-to-End Testing"
- Section 3.2.3.4, "Oracle BPEL PM Integration with Outbound Interaction"
- Section 3.2.3.5, "Oracle BPEL PM Integration with Inbound Interaction"
- Section 3.2.3.6, "Use Case: Integration with Oracle BPEL Process Manager"

3.2.3.1 Design Time

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

Adapter	Tool
Oracle Technology Adapters	Oracle JDeveloper
Legacy Adapters	Oracle Studio
Packaged-Application Adapters	Application Explorer
Oracle Adapter for Oracle Applications	Oracle JDeveloper

WSDL files are created for both request-response and event-notification services of an adapter. The J2CA extension contains J2CA elements that are required by the JCA Binding Component during run time to convert web service messages to J2CA Interactions and back. The J2CA WSDL extension elements contain the metadata for

the JCA Binding Component to call any request-response service and activate any inbound J2CA 1.5 endpoint to receive inbound events. The J2CA extension elements for the request-response service contains the JNDI location and `InteractionSpec` details for calling an outbound interaction. The J2CA extension elements for the event-notification service contains the resource adapter class name and `ActivationSpec` parameters for publishing an adapter event through the J2CA inbound interaction.

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

Figure 3–3 Design Time Configuration of Technology Adapters

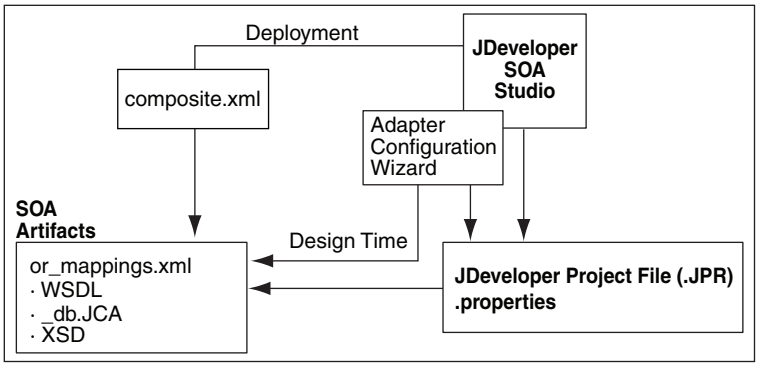
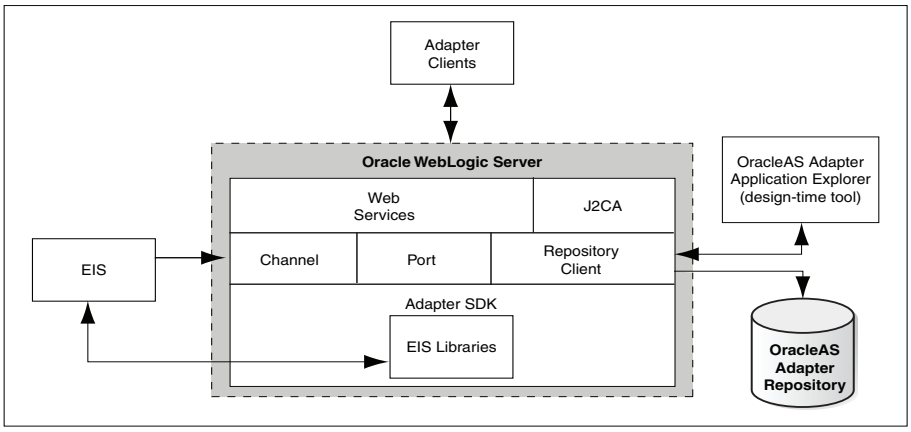


Figure 3–4 illustrates the design-time tool for configuring packaged-application adapters. In this figure, the design-time tools are used to expose adapter metadata as WSDL files. The WSDL files are consumed by BPEL Process Manager during run time.

Figure 3–4 Configuring Packaged-Application Adapters



3.2.3.2 Run Time

Oracle Application Server adapters are based on the J2CA 1.5 specification, and BPEL is deployed on the 11*g* run-time on the Oracle WebLogic Server. The JCA Binding Component acts as a glue layer that integrates the standard J2CA 1.5 resource adapter with the Oracle BPEL Process Manager and Oracle Mediator during run time. The JCA Binding Component acts as a pseudo J2CA 1.5 container.

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

3.2.3.3 End-to-End Testing

You could also wrap up your custom adapter as a web Service, and expose this to BPEL Process Manager. This is a loose coupling strategy and does not need an Adapter SDK. Both these approaches (JCA/web service) are suitable for outbound invoke operations referred to as reference. Only the JCA 1.5 integration allows the Oracle BPEL PM to receive inbound events (from EIS to J2EE/Oracle BPEL PM). The Oracle BPEL PM acts as a pseudo JCA 1.5 container and implements the JCA 1.5-specific System Contracts.

You can use any custom design tool for the configuration of the adapter, but a WSDL file must be generated at the end of the design-time phase for consumption by the Oracle BPEL PM design-time (JDeveloper). The WSDL file for the JCA interactions have a JCA extension. The Adapter is a JCA 1.5 resource adapter deployed in the same Oracle WebLogic Server container as that of the Oracle BPEL PM product. The JCA 1.5 Resource Adapter and the Oracle BPEL PM instance must be deployed in the same Oracle WebLogic Server container.

The JCA Binding Component is the glue layer that integrates the standard JCA 1.5 Resource Adapter seamlessly with the Oracle BPEL PM product at run time. The JCA Binding Component has a JCA Provider for wrapping the JCA interactions as web Services and performs the translation between web Service messages to JCA interaction messages based on the WSDL files generated at design time.

3.2.3.4 Oracle BPEL PM Integration with Outbound Interaction

The following is a snippet of the `composite.xml` file for an outbound invoke (referred to as reference in the 11*g* release):

```
<reference name="insert" ui:wsdlLocation="insert.wsdl">
    <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/db/DBRetriesApplication/XARollba
ck/insert%2F#wsdl.interface(insert_ptt)"/>
    <binding.jca config="insert_db.jca"/>
</reference>
```

The following list summarizes the process of BPEL Process Manager integration with the outbound interaction:

- During design time, adapter services are exposed as WSDL files and consumed during configuration of the `PartnerLink` activity of the BPEL process.
- The `.jca` file contains the JNDI address of the resource adapter, `InteractionSpec` class name, `InteractionSpec` parameters.
- During run time, the Invoke activity of the BPEL Process Manager is used to call the `PartnerLink` activity, which is a J2CA Resource Adapter outbound interaction.
- The components are wired into a composite application.
- The JCA Binding Component translates the event to a web service response for consumption by the Oracle BPEL PM instance.
- The outbound JCA adapter communicates with the EIS through CCI interaction.

> **Note:** The outbound interaction with Oracle Mediator is the same as that of Oracle BPEL PM.

3.2.3.5 Oracle BPEL PM Integration with Inbound Interaction

BPEL Process Manager receives events from the J2CA 1.5 resource adapter through the JCA Binding Component, which is the pseudo J2CA 1.5 container and implements the message inflow contracts for receiving events from the adapter. The J2CA inbound interaction is captured in a WSDL file during design time. The J2CA inbound WSDL binding section contains the J2CA 1.5 `ActivationSpec` parameter. The `ActivationSpec` parameter captures the inbound connectivity and inbound interaction details (according to J2CA 1.5 specification). The J2CA Inbound WSDL Service section contains the J2CA 1.5 `ResourceAdapter` class name. In addition, the Service section can optionally contain a JNDI location.

The following list summarizes the process of BPEL Process Manager integration with the inbound interaction:

- The `ResourceAdapter` class name and the `ActivationSpec` parameter are captured in the WSDL extension section of the J2CA inbound interaction WSDL during design time and made available to BPEL Process Manager and the JCA Binding Component during run time.
- An instance of the J2CA 1.5 `ResourceAdapter` class is created, and the `Start` method of the J2CA `ResourceAdapter` class is called.
- Each inbound interaction operation referenced by the BPEL Process Manager processes results in invoking the `EndPointActivation` method of the J2CA 1.5 `ResourceAdapter` instance. The JCA Binding Component creates the `ActivationSpec` class (Java bean) based on the `ActivationSpec` details present in the WSDL extension section of the J2CA inbound interaction and activates the endpoint of the J2CA 1.5 resource adapter.
- The JCA Binding Component `MessageEndpoint` implementation implements the `javax.resource.cci.MessageListener` interface. The J2CA 1.5 resource adapter calls the `onMessage()` method in this `MessageEndpoint` when it receives a back-end application event. The J2CA 1.5 resource adapter creates an instance of the `MessageEndpoint` implementation through `MessageEndpointFactory` provided to the resource adapter during `endpointActivation`.
- The JCA Binding Component receives the event through the `MessageListener` class and forwards it to the Receive activity of the BPEL Process Manager instance.
- When the BPEL process is stopped, all associated inbound end points are deactivated through the `endPointDeactivation` method implemented by the resource adapter.

In the case of J2CA adapters, particularly the JDBC based ones, such as Oracle Database Adapter and Oracle AQ Adapter, there are two kinds of connection management at play:

- for inbound (endpoint) activations (BPEL Receive)
- for outbound interactions (BPEL Invoke).

In the case of inbound activations, the J2CA adapter is fully in charge of connection creation and recovery. The JCA Binding Component can only be requested to lookup and provide a J2CA `ConnectionFactory` handle to the adapter through its `ActivationSpec`. This is possible only if it implements a certain interface, which it can use to create connections, thereby going through the Application Server connection manager. Whenever a managed (JDBC) connection goes bad, the adapter must close the J2CA connection handle (and subsequently the managed connection if

`destroy()` is called by the Application Server), enter a temporary recovery loop, and then try to reestablish a new connection.

In the case of outbound interactions (J2CA), each port caches tuples of the following:

- `ConnectionFactory`
- `ConnectionSpec`
- `Connection`
- `Interaction`
- `InteractionSpec`

As the BPEL engine typically invokes the port concurrently with any number of threads, the size of the cache reflect the highest concurrency level at any given time. The cache can be tuned to automatically expire unused tuples after a configured idle period (interactions and connection handles are then closed). The cache greatly improves performance in high load environments, for example, Retek (8 million transactions every hour).

If just one JCA adapter interaction using the cache throws a `ResourceException`, then all members of the cache are closed and released immediately (purged), so new interactions have to re-create (fresh) members to the cache. The BPEL engine has a feature known as *PartnerLink* retry which can be configured for each invoke. Thus, any JCA adapter invoke or interaction which throws a `ResourceException` exception marked as `Retryable` make the engine retry the Invoke (Database update) which then repopulate the port cache (if the Database has become available again: typically immediately the case with Oracle RAC).For non-transactional adapters (`adapterMetadata.supportsLocalTransactionDemarcation() == false`), such as File adapter, the J2CA connection cache contains only one member. Thus all threads coming through multiplex over the same CCI Connection handle.

The JCA connection cache can be enabled or configured explicitly by using the following `bpel.xml` partnerlink properties:

```
<property name="useJCAConnectionPool">true</property>
```

Generally, this property is derived from the declared transactional support of the adapter. For example, the File adapter does not use this connection pool because it is multi thread safe, but that can be overridden through the following property:

```
<property name="maxSizeJCAConnectionPool">500</property>
```

If the property mentioned in the preceding example is not specified, then the size of the connection pool is assumed to be unbounded. This applies for each partnerlink.

```
<property name="lruConnectionMaxIdleAge">50000</property>
```

The maximum age of idle connections in the pool is important because some type of connections hold on to expensive external resources, for example DB shadow processes which is measured in ms, as shown in the following example:

```
property name="lruConnectionCheckInterval">10000</property>
```

Finally, the property mentioned in the preceding example determines how frequently the connection pool should be scanned for idle connections, also measured in ms.

The following is a code snippet of the `composite.xml` file for an inbound polling receive operation (referred to as service in the 11*g* release):

```
<service name="poll" ui:wsdlLocation="poll.wsdl">
    <interface.wsdl
```

```
interface="http://xmlns.oracle.com/pcbpel/adapter/db/DBRetriesApplication/
XARollback/poll%2F#wsdl.interface(poll_ptt)"
(http://xmlns.oracle.com/pcbpel/adapter/db/DBRetriesApplication/XARollback/poll%2F
#wsdl.interface%28poll_ptt%29)/>
    <binding.jca config="poll_db.jca"/>
</service>
```

Note how the `composite.xml` file links the WSDL interface (`the interface.wsdl` file), the name of the component which is handling the request (the `binding.jca` file), and the binding information required to invoke a particular call (`the config` file). Hence the JCA Binding Component is registered in SCA as the implementation of the `binding.jca` file (others include `binding.ejb` and `binding.java`), while in the 10.1.3 release it was registered as a WSIF provider.

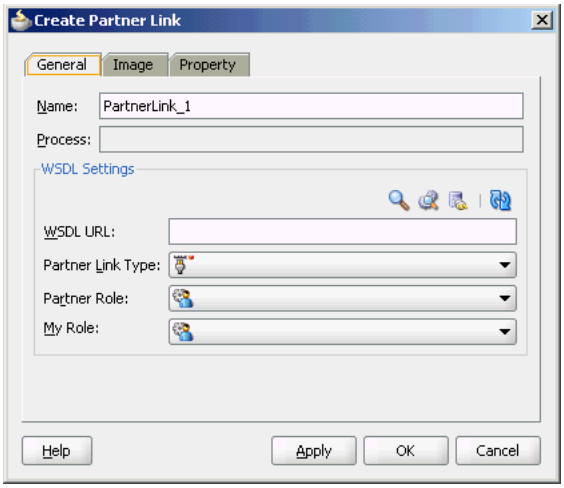
In the current release the `<binding.jca>` element is in the `composite.xml` file, which explicitly indicates that the JCA Binding Component is handling the invoke activity. Whereas in the 10.1.3 release you had to look at the concrete binding in the WSDL to see whether it was an adapter invoke or not, as shown in the following example:

```
<binding name="invokeService_binding" type="tns:invokeService_ptt">
<jca:binding />
<operation name="merge">
<jca:operation>
```

3.2.3.6 Use Case: Integration with Oracle BPEL Process Manager

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

Figure 3–5 Partner Link dialog box

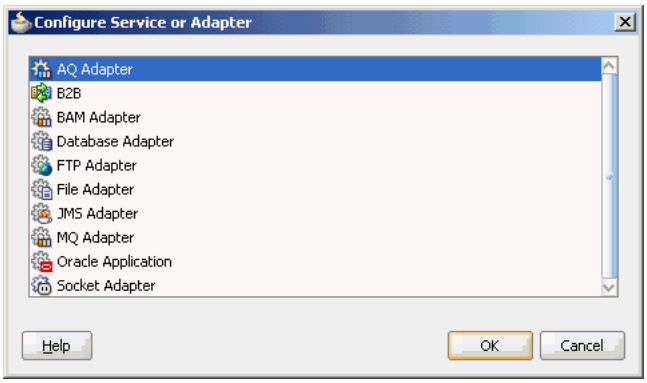


Click the **Define Service** icon, shown in Figure 3–6, to access the Configure Service or Adapter dialog.

Figure 3–6 Defining an Adapter

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

Figure 3–7 Adapter Types



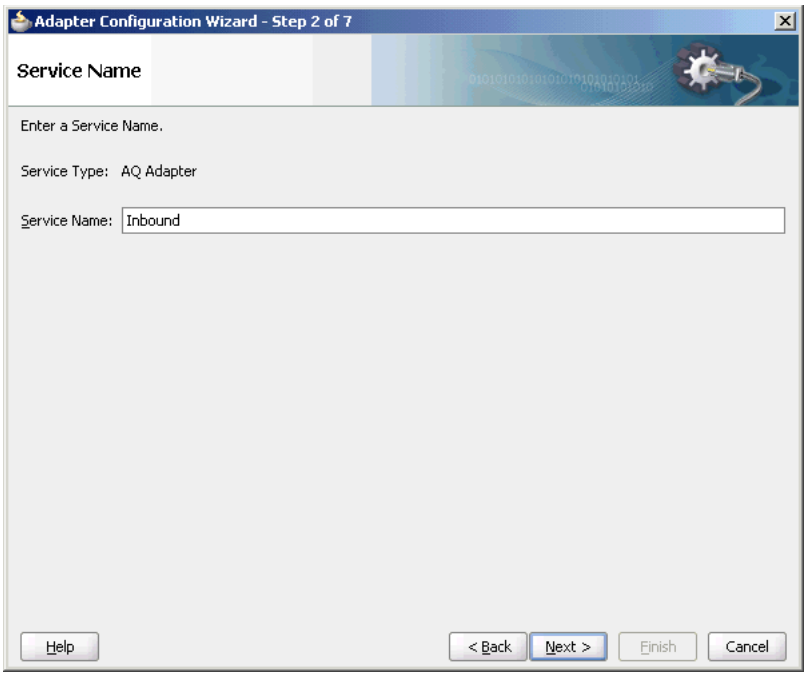
When you select an adapter type (Oracle AQ Adapter in this example), and then click **OK**, the Adapter Configuration Wizard - Welcome page appears, as shown in Figure 3–8.

Figure 3–8 The Adapter Configuration Wizard- Welcome Page

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

For this example, AQ Adapter is selected, as shown in Figure 3–7. When the wizard completes, a WSDL file by this service name appears in the Application Navigator for the BPEL process (for this example, named `DequeueDemo.wsdl`). This file includes the adapter configuration settings you specify with this wizard. Other configuration files (such as header properties and files specific to the adapter) are also created and displayed in the Application Navigator.

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



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

3.2.4 Oracle SOA Composite Integration with Adapters

Oracle JCA Adapters can be integrated with Oracle SOA Suite.

This section includes the following:

- Section 3.2.4.1, "Oracle SOA Composite Overview"
- Section 3.2.4.2, "Adapters Integration With Oracle SOA Composite"

3.2.4.1 Oracle SOA Composite Overview

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

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

3.2.4.2 Adapters Integration With Oracle SOA Composite

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

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

An Oracle SOA composite typically comprises the following parts:

- **Binding Components**

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

  – *Service* Binding Components

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

  – *Reference* Binding Components

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

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

- **Service Infrastructure**

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

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

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

  For more information about adapter integration with service engines, see Section 3.2, "Adapter Integration with Oracle Fusion Middleware."

- **UDDI and MDS**

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

- **SOA Archive: Composite**

  The deployment unit that describes the composite application.

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

The following is a sample `composite.xml` file:

```
Composite.xml (JCA Bindings)<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SOA Modeler version 1.0 at [2/23/09 3:02 PM]. -->
<composite name="MediatorFlatStructure"
   revision="1.0"
```

```
   label="2009-02-23_15-02-00_374"
   mode="active"
   state="on"
   xmlns="http://xmlns.oracle.com/sca/1.0"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
   xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy"
   xmlns:ui="http://xmlns.oracle.com/soa/designer/">
    <import
namespace="http://xmlns.oracle.com/pcbpel/adapter/file/SOA-FlatStructure/MediatorF
latStructure/MedFlatIn%2F" location="MedFlatIn.wsdl" importType="wsdl"/>
    <import
namespace="http://xmlns.oracle.com/pcbpel/adapter/file/SOA-FlatStructure/MediatorF
latStructure/MedFlatOut%2F" location="MedFlatOut.wsdl" importType="wsdl"/>
    <service name="MedFlatIn" ui:wsdlLocation="MedFlatIn.wsdl">
        <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/file/SOA-FlatStructure/MediatorF
latStructure/MedFlatIn%2F#wsdl.interface(Read_ptt)"/>
        <binding.jca config="MedFlatIn_file.jca"/>
    </service>
    <component name="MediatorFlat">
        <implementation.mediator src="MediatorFlat.mplan"/>
    </component>
    <reference name="MedFlatOut" ui:wsdlLocation="MedFlatOut.wsdl">
        <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/file/SOA-FlatStructure/MediatorF
latStructure/MedFlatOut%2F#wsdl.interface(Write_ptt)"/>
        <binding.jca config="MedFlatOut_file.jca"/>
    </reference>
    <wire>
        <source.uri>MedFlatIn</source.uri>
        <target.uri>MediatorFlat/MediatorFlat</target.uri>
    </wire>
    <wire>
        <source.uri>MediatorFlat/MedFlatOut</source.uri>
        <target.uri>MedFlatOut</target.uri>
    </wire>
</composite>
```

For more information about Oracle SOA composite and its integration with various service engines, see Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.

3.3 Monitoring Oracle JCA Adapters

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

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

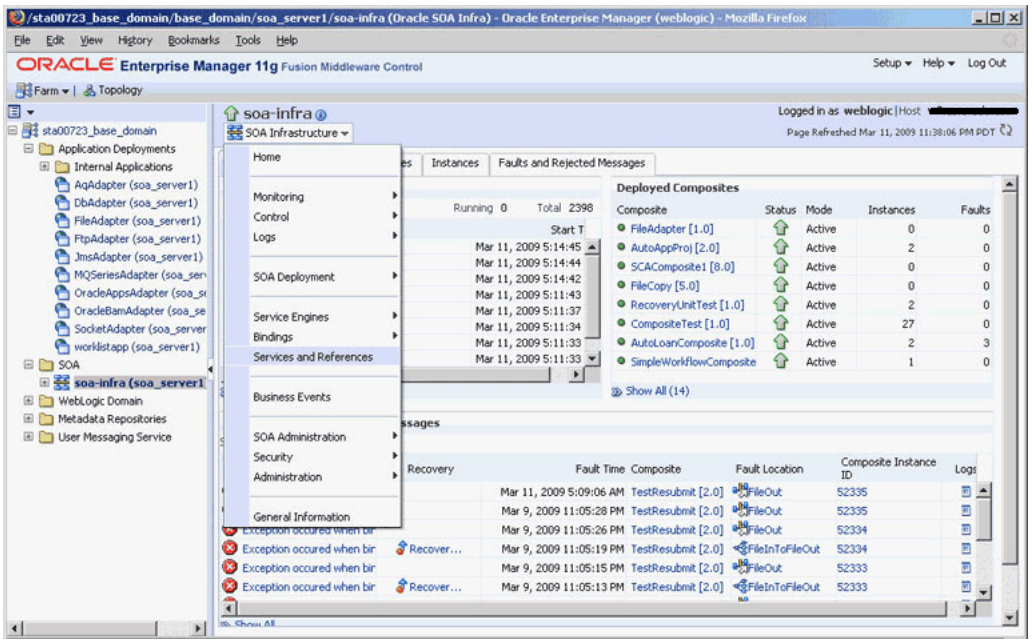
  - Parsing Result
- Adapter Postprocessing

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

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

   The soa-infra page is displayed.
3. From the SOA Infrastructure menu in the soa-infra page, click **Services and References**, as shown in Figure 3–10.

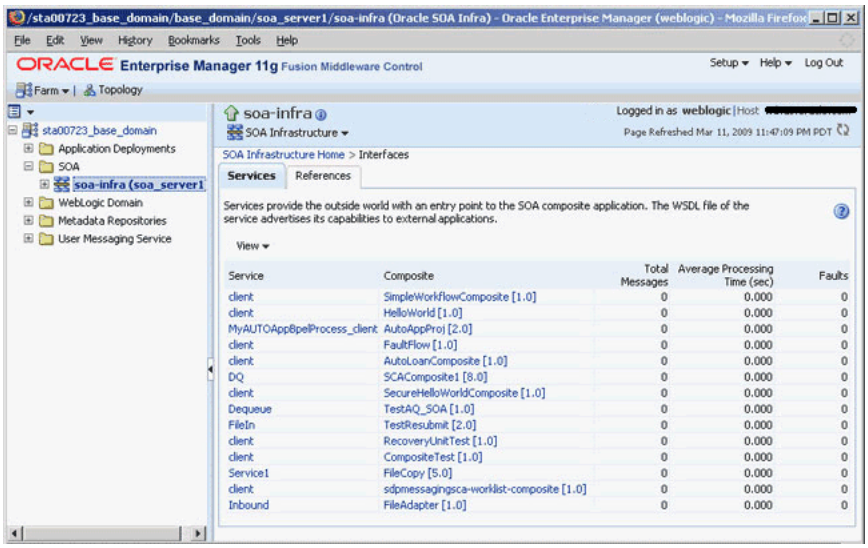
Figure 3–10 Viewing the Adapter Statistics in the Fusion Middleware Control Console



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

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

Figure 3–11 The SOA Infrastructure Home > Interfaces Page



4

Oracle JCA Adapter for Files/FTP

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

This chapter includes the following sections:

- Section 4.1, "Introduction to Oracle File and FTP Adapters"
- Section 4.2, "Oracle File and FTP Adapters Features"
- Section 4.3, "Oracle File and FTP Adapter Concepts"
- Section 4.4, "Configuring Oracle File and FTP Adapters"
- Section 4.5, "Oracle File and FTP Adapters Use Cases"

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

4.1 Introduction to Oracle File and FTP Adapters

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

This section includes the following topics:

- Section 4.1.1, "Oracle File and FTP Adapters Architecture"
- Section 4.1.2, "Oracle File and FTP Adapters Integration with Oracle BPEL PM"
- Section 4.1.3, "Oracle File and FTP Adapters Integration with Mediator"

4.1.1 Oracle File and FTP Adapters Architecture

The Oracle File and FTP Adapters are based on JCA 1.5 architecture. JCA provides a standard architecture for integrating heterogeneous enterprise information systems (EIS). The JCA Binding Component of the Oracle File and FTP Adapters expose the underlying JCA interactions as services (`WSDL` with JCA binding) for Oracle BPEL PM integration. For details about Oracle JCA Adapter architecture, see Chapter 1, "Introduction to Oracle JCA Adapters."

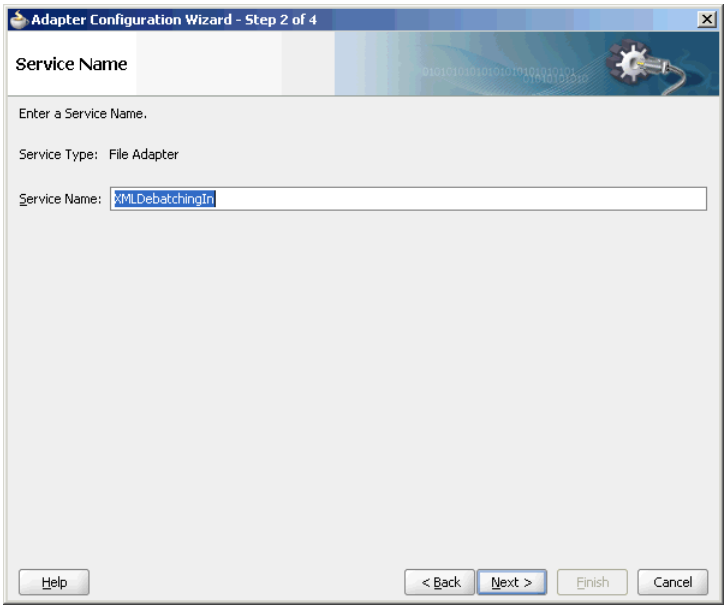
4.1.2 Oracle File and FTP Adapters Integration with Oracle BPEL PM

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

Figure 4–1 The Adapter Configuration Wizard - Welcome Page

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

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



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

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

Table 4–1 Supported Operations for Oracle BPEL Process Manager

Operation	Section
Oracle File Adapter	-
■ Read File (inbound operation)	Section 4.3.1, "Oracle File Adapter Read File Concepts"
■ Write File (outbound operation)	Section 4.3.2, "Oracle File Adapter Write File Concepts"
■ Synchronous Read File (outbound operation)	Section 4.3.3, "Oracle File Adapter Synchronous Read Concepts"
■ List Files (outbound operation)	Section 4.3.4, "Oracle File Adapter File Listing Concepts"
Oracle FTP Adapter	-
■ Get File (inbound operation)	Section 4.3.5, "Oracle FTP Adapter Get File Concepts"
■ Put File (outbound operation)	Section 4.3.6, "Oracle FTP Adapter Put File Concepts"
■ Synchronous Get File (outbound operation)	Section 4.3.7, "Oracle FTP Adapter Synchronous Get File Concepts"
■ List Files (outbound operation)	Section 4.3.8, "Oracle FTP Adapter File Listing Concepts"

For more information about Oracle JCA Adapter integration with Oracle BPEL PM, see Chapter 1, "Introduction to Oracle JCA Adapters."

4.1.3 Oracle File and FTP Adapters Integration with Mediator

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

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

The Operation Type page of the Adapter Configuration Wizard prompts you to select an operation to perform. Based on your selection, different Adapter Configuration Wizard pages appear and prompt you for configuration information. Table 4–2 lists the available operations and provides references to sections that describe the configuration information you must provide. For more information about Adapters and Mediator, see Chapter 1, "Introduction to Oracle JCA Adapters."

Table 4–2 Supported Operations for Oracle Mediator

Operation	Section
Oracle File Adapter	-
■ Read File (inbound operation)	Section 4.3.1, "Oracle File Adapter Read File Concepts"
■ Write File (outbound operation)	Section 4.3.2, "Oracle File Adapter Write File Concepts"
■ Synchronous Read File (outbound operation)	Section 4.3.3, "Oracle File Adapter Synchronous Read Concepts"
■ List Files (outbound operation)	Section 4.3.4, "Oracle File Adapter File Listing Concepts"
Oracle FTP Adapter	-
■ Get File (inbound operation)	Section 4.3.5, "Oracle FTP Adapter Get File Concepts"
■ Put File (outbound operation)	Section 4.3.6, "Oracle FTP Adapter Put File Concepts"
■ Synchronous Get File (outbound operation)	Section 4.3.7, "Oracle FTP Adapter Synchronous Get File Concepts"
■ List Files (outbound operation)	Section 4.3.8, "Oracle FTP Adapter File Listing Concepts"

4.1.4 Oracle File and FTP Adapters Integration with SOA Composite

A composite is an assembly of services, service components (Oracle BPEL PM and Mediator), wires, and references designed and deployed in a single application. The composite processes the information described in the messages. The details of the composite are stored in the `composite.xml` file. For more information about integration of the Oracle File and FTP Adapters with SOA composite, see Section 3.2.4, "Oracle SOA Composite Integration with Adapters."

4.2 Oracle File and FTP Adapters Features

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

- File Formats
- FTP Servers
- Inbound and Outbound Interactions
- File Debatching
- File ChunkedRead
- File Sorting
- Dynamic Outbound Directory and File Name Specification
- Security
- Nontransactional
- Proxy Support
- No Payload Support

- Large Payload Support
- File-Based Triggers
- Pre-Processing and Post-Processing of Files
- Error Handling
- Threading Model
- Performance Tuning
- High Availability
- Multiple Directories
- Append Mode
- Recursive Processing of Files Within Directories in Oracle FTP Adapter
- Securing Enterprise Information System Credentials

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

4.2.1 File Formats

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

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

4.2.2 FTP Servers

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

Note: Oracle FTP Adapter supports SFTP server version 4 or later.

4.2.3 Inbound and Outbound Interactions

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

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

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

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

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

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

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

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

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

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

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

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

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

- Section 4.3.1, "Oracle File Adapter Read File Concepts"
- Section 4.3.2, "Oracle File Adapter Write File Concepts"

- Section 4.3.5, "Oracle FTP Adapter Get File Concepts"
- Section 4.3.6, "Oracle FTP Adapter Put File Concepts"

4.2.4 File Debatching

When a file contains multiple messages, you can choose to publish messages in a specific number of batches. This is referred to as debatching. During debatching, the file reader, on restart, proceeds from where it left off in the previous run, thereby avoiding duplicate messages. File debatching is supported for files in XML and native formats. You *must not* manually change the value of the publish size parameter in `JCA` files. You must use the Adapter Configuration Wizard to modify this parameter.

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

```
<service ...
  <binding.jca ...
    <property name="batchNotificationHandler">com.acme.batchHandler</property>
```

where `com.acme.batchHandler` must implement

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

4.2.5 File ChunkedRead

This is a feature of Oracle File and FTP Adapters that uses an invoke activity within a while loop to process the target file. This feature enables you to process arbitrarily large files.

If an invalid payload is provided, then ChunkedRead scenarios do not throw an exception. When a translation exception (bad record violating the NXSD specification) is encountered, the return header is populated with the translation exception message that includes details such as line and column where the error occurred. All translation errors do not result in a fault. These errors are manifested as a value in the return header. You must check the `jca.file.IsMessageRejected` and `jca.file.RejectionReason` header values to ascertain whether an exception has occurred. Additionally, you can also check the `jca.file.NoDataFound` header value.

4.2.6 File Sorting

When files must be processed by Oracle File and FTP Adapters in a particular order, you must configure the sorting parameters. For example, you can configure the sorting

parameters for Oracle File and FTP Adapters to process files in ascending or descending order by time stamps.

You must meet the following prerequisites for sorting scenarios of Oracle File and FTP Adapters:

- Use a synchronous operation
- Add the following property to the inbound JCA file:

```
<property name="ListSorter"
value="oracle.tip.adapter.file.inbound.listing.TimestampSorterAscending"/>
<property name="SingleThreadModel" value="true"/>
```

4.2.7 Dynamic Outbound Directory and File Name Specification

The Oracle File and FTP Adapters enable you to dynamically specify the logical or physical name of the outbound file or outbound directory. For information about how to specify dynamic outbound directory, see Section 4.3.2.2, "Outbound File Directory Creation."

4.2.8 Security

The Oracle FTP Adapter supports FTP over SSL (FTPS) and Secure FTP (SFTP) to enable secure file transfer over a network.

For more information, see Section 4.4.3, "Using Secure FTP with the Oracle FTP Adapter" and Section 4.4.4, "Using SFTP with Oracle FTP Adapter."

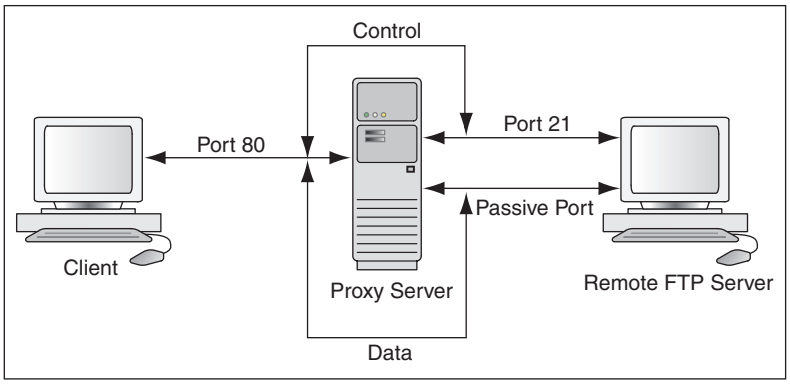
4.2.9 Nontransactional

The Oracle File Adapter picks up a file from an inbound directory, processes the file, and sends the processed file to an output directory. However, during this process if a failover occurs in the Oracle RAC back end or in an SOA managed server, then the file is processed twice because of the nontransactional nature of Oracle File Adapter. As a result, there can be duplicate files in the output directory.

4.2.10 Proxy Support

You can use the proxy support feature of the Oracle FTP Adapter to transfer and retrieve data to and from the FTP servers that are located outside a firewall or can only be accessed through a proxy server. A proxy server enables the hosts in an intranet to indirectly connect to hosts on the Internet. Figure 4–3 shows how a proxy server creates connections to simulate a direct connection between the client and the remote FTP server.

Figure 4–3 Remote FTP Server Communication Through a Proxy Server



To use the HTTP proxy feature, your proxy server must support FTP traffic through HTTP Connection. In addition, only passive data connections are supported with this feature. For information about how to configure the Oracle FTP Adapter, see Section 4.4.5, "Configuring Oracle FTP Adapter for HTTP Proxy."

4.2.11 No Payload Support

For Oracle BPEL PM and Mediator, the Oracle File and FTP Adapters provide support for publishing only file metadata such as file name, directory, file size, and last modified time to a BPEL process or Mediator and excludes the payload. The process can use this metadata for subsequent processing. For example, the process can call another reference and pass the file and directory name for further processing.You can use the Oracle File and FTP Adapters as a notification service to notify a process whenever a new file appears in the inbound directory. To use this feature, select the **Do not read file content** check box in the JDeveloper wizard while configuring the "Read operation."

4.2.12 Large Payload Support

For Oracle BPEL PM and Mediator, the Oracle File Adapter provides support for transferring large files as attachments. To use this feature, select the **Read File As Attachment** check box in the JDeveloper wizard while configuring the "Read operation." This option opaquely transfers a large amount of data from one place to another as attachments. For example, you can transfer large MS Word documents, images, and PDFs without processing their content within the composite application. For information about how to pass large payloads as attachments, see Section 4.5.6, "Oracle File Adapter Read File As Attachments."

Note: You must not pass large payloads as opaque schemas.

4.2.13 File-Based Triggers

You can use the Oracle File and FTP Adapters, which provide support for file-based triggers, to control inbound adapter endpoint activation. For information about how to use file-based triggers, see Section 4.3.1.4, "File Polling."

4.2.14 Pre-Processing and Post-Processing of Files

The process modeler may encounter situations where files must be pre-processed before they are picked up for processing or post-processed before the files are written

out to the destination folder. For example, the files that the Oracle File and FTP adapters receive may be compressed or encrypted and the adapter must decompress or decrypt the files before processing. In this case you must use a custom code to decompress or decrypt the files before processing. The Oracle File and FTP Adapters supports the use of custom code that can be plugged in for pre-processing or post-processing of files.

The implementation of the pre-processing and post-processing of files is restricted to the following communication modes of the Oracle File and FTP Adapters:

- Read File or Get File
- Write File or Put File
- Synchronous Read File
- Chunked Read

This section contains the following topics:

- Section 4.2.14.1, "Mechanism For Pre-Processing and Post-Processing of Files"
- Section 4.2.14.2, "Configuring a Pipeline"
- Section 4.2.14.3, "Using a Re-Entrant Valve For Processing Zip Files"
- Section 4.2.14.4, "Configuring Batch Notification Handler"

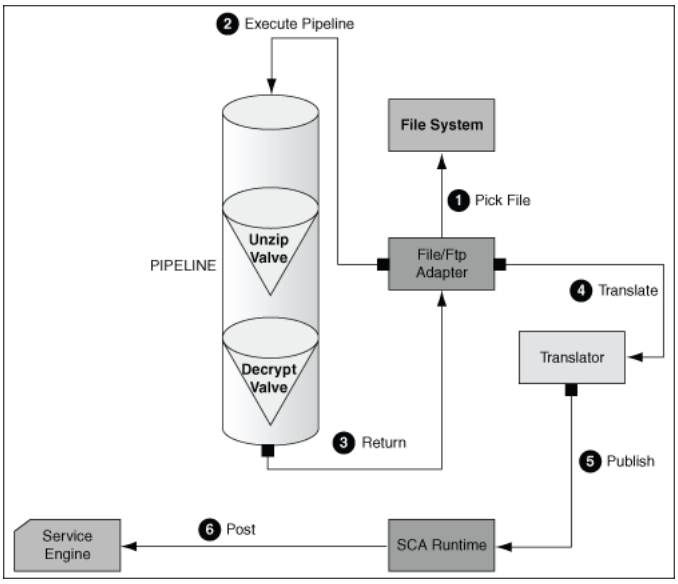
4.2.14.1 Mechanism For Pre-Processing and Post-Processing of Files

The mechanism for pre-processing and post-processing of files is configured as pipelines and valves. This section describes the concept of pipelines and valves.

A **pipeline** consists of a series of custom-defined valves. A pipeline loads a stream from the file system, subjects the stream to processing by an ordered sequence of valves, and after the processing returns the modified stream to the adapter.

A **valve** is the primary component of execution in a processing pipeline. A valve processes the content it receives and forwards the processed content to the next valve. For example, in a scenario where the Oracle File and FTP Adapters receive files that are encrypted and zipped, you can configure a pipeline with an unzip valve followed by a decryption valve. The unzip valve extracts the file content before forwarding it to the decryption valve, which decrypts the content and the final content is made available to the Oracle File or FTP Adapter as shown in Figure 4–4.

Figure 4–4 A Sample Pre-Processing Pipeline



4.2.14.2 Configuring a Pipeline

Configuring the mechanism for pre-processing and post-processing of files requires defining a pipeline and configuring it in the corresponding `JCA` file.

To configure a pipeline, you must perform the following steps:

- Step 1, "Implementing and Extending Valves"
- Step 2, "Compiling the Valves"
- Step 3, "Creating a Pipeline"
- Step 4, "Adding the Pipeline to the SOA Project Directory"
- Step 5, "Registering the Pipeline"

Step 1 Implementing and Extending Valves

All valves must implement `Valve` or `StagedValve` interface.

> **Tip:** You can extend either the `AbstractValve` or the `AbstractStagedValve` class based on business requirement rather than implementing a valve from the beginning.

Example 4–1 is a sample valve interface.

Example 4–1 The Valve Interface

```
package oracle.tip.pc.services.pipeline;


import java.io.IOException;
/** <p>
 *  Valve component is resposible for processing the input stream
 *  and returning a modified input stream.
 *  The <code>execute()</code> method of the valve gets invoked
 *  by the caller (on behalf) of the pipeline. This method must
 *  return the input stream wrapped within an InputStreamContext.
 *  The Valve is also responsible for error handling specifically
```

```
 *
 *  The Valve can be marked as reentrant in which case the caller
 *  must call the <code>execute()</code> multiple times and each
 *  invocation must return a new input stream. This is useful, if
 *  you are writing an UnzipValve since each iteration of the valve
 *  must return the input stream for a different zipped entry.
 *  <b> You must note that only the first Valve in the pipeline can
 *  be reentrant </b>
 *
 *  The Valve has another flavor <code>StagedValve</code> and if
 *  the valve implements StagedValve, then the valve must store
 *  intermediate content in a staging file and return it whenever
 *  required.
 * </p>
 */
public interface Valve
{
    /**
     * Set the Pipeline instance. This parameter can be
     * used to get a reference to the PipelineContext instance.
     * @param pipeline
     */
    public void setPipeline(Pipeline pipeline);

    /** Returns the Pipeline instance.
       * @return
       */
      public Pipeline getPipeline();

      /** Returns true if the valve has more input streams to return
       *  For example, if the input stream is from a zipped file, then
       *  each invocation of <code>execute()</code> returns a different
       *  input stream once for each zipped entry. The caller calls
       *  <code>hasNext()</code> to check if more entries are available
       * @return true/false
       */
      public boolean hasNext();

      /** Set to true if the caller can call the valve multiple times
       *  e.g. in case of ZippedInputStreams
       * @param reentrant
       */
      public void setReentrant(boolean reentrant);

      /** Returns true if the valve is reentrant.
     * @return
     */
    public boolean isReentrant();

    /** The method is called by pipeline to return the modified input stream
     * @param in
     * @return InputStreamContext that wraps the input stream along with required
metadata
     * @throws PipelineException
     */
    public InputStreamContext execute(InputStreamContext in) throws
PipelineException, IOException;

      /**
       * This method is called by the pipeline after the caller publishes the
```

```
       * message to the SCA container.
       * In the case of a zipped file, this method gets called repeatedly, once
       * for each entry in the zip file.
       * This should be used by the Valve to do additional tasks such as
       * delete the staging file that has been processed in a reentrant
scenario.
       * @param in The original InputStreamContext returned from
<code>execute()</code>
       */
      public void finalize(InputStreamContext in);

    /**Cleans up intermediate staging files, input streams
     * @throws PipelineException, IOException
     */
    public void cleanup() throws PipelineException, IOException;
 }
```

The `StagedValve` stores intermediate content in staging files. Example 4–2 shows the `StagedValve` interface extending the `Valve` interface.

Example 4–2 The StagedValve Interface Extending the Valve Interface

```
package oracle.tip.pc.services.pipeline;

import java.io.File;

/**
 * A special valve that stages the modified
 * input stream in a staging file.
 * If such a <code>Valve</code> exists, then
 * it must return the staging file containing
 * the intermediate data.
 */
public interface StagedValve extends Valve {

      /**
       * @return staging file where the valve stores its intermediate results
       */
      public File getStagingFile();
}
```

Example 4–3 is a sample of an `AbstractValve` class implementing the `Valve` interface.

Example 4–3 The AbstractValve Class Implementing the Valve Interface

```
package oracle.tip.pc.services.pipeline;

import java.io.IOException;

/**
 * A bare bone implementation of Valve. The user should extend from
 * AbstractValve rather than implementing a Valve from scratch
 *
 */
public abstract class AbstractValve implements Valve {

      /**
       * The pipeline instance is stored as a member
       */
```

```
      private Pipeline pipeline = null;

      /**
       * If reentrant is set to true, then the Valve must adhere to the
following:
       * i) It must the first valve in the pipeline ii) Must implement hasNext
       * method and return true if more input streams are available A reentrant
       * valve will be called by the pipeline more than once and each time the
       * valve must return a different input stream, for example Zipped entries
       * within a zip file
       */
      private boolean reentrant = false;

      /*
       * Save the pipeline instance.
       *
       * @see
oracle.tip.pc.services.pipeline.Valve#setPipeline(oracle.tip.pc.services.pipeline
.Pipeline)
       */
      public void setPipeline(Pipeline pipeline) {
            this.pipeline = pipeline;
      }

      /*
       * Return the pipeline instance (non-Javadoc)
       *
       * @see oracle.tip.pc.services.pipeline.Valve#getPipeline()
       */
      public Pipeline getPipeline() {
            return this.pipeline;
      }

      /*
       * Return true if the valve is reentrant (non-Javadoc)
       *
       * @see oracle.tip.pc.services.pipeline.Valve#isReentrant()
       */
      public boolean isReentrant() {
            return this.reentrant;
      }

      /*
       * If set to true, the valve is reentrant (non-Javadoc)
       *
       * @see oracle.tip.pc.services.pipeline.Valve#setReentrant(boolean)
       */
      public void setReentrant(boolean reentrant) {
            this.reentrant = reentrant;
      }

      /*
       * By default, set to false For valves that can return more than one
       * inputstreams to callers, this parameter must return true/false
depending
       * on the availability of input streams (non-Javadoc)
       *
       * @see oracle.tip.pc.services.pipeline.Valve#hasNext()
       */
      public boolean hasNext() {
```

```
            return false;
      }

      /*
       * Implemented by concrete valve (non-Javadoc)
       *
       * @see oracle.tip.pc.services.pipeline.Valve#execute(InputStreamContext)
       */
      public abstract InputStreamContext execute(InputStreamContext in)
                  throws PipelineException, IOException;

      /*
       * Implemented by concrete valve (non-Javadoc)
       *
       * @see
oracle.tip.pc.services.pipeline.Valve#finalize(oracle.tip.pc.services.pipeline.In
putStreamContext)
       */
      public abstract void finalize(InputStreamContext in);

      /*
       * Implemented by concrete valve (non-Javadoc)
       *
       * @see oracle.tip.pc.services.pipeline.Valve#cleanup()
       */
      public abstract void cleanup() throws PipelineException, IOException;

}
```

Example 4–4 shows the `AbstractStagedValve` class extending the `AbstractValve` class.

Example 4–4 The AbstractStagedValve Class Extending the AbstractValve Class

```
package oracle.tip.pc.services.pipeline;

import java.io.File;
import java.io.IOException;

public abstract class AbstractStagedValve extends AbstractValve implements
            StagedValve {

      public abstract File getStagingFile();

      public abstract void cleanup() throws IOException, PipelineException;

      public abstract InputStreamContext execute(InputStreamContext in)
                  throws IOException, PipelineException;

}
```

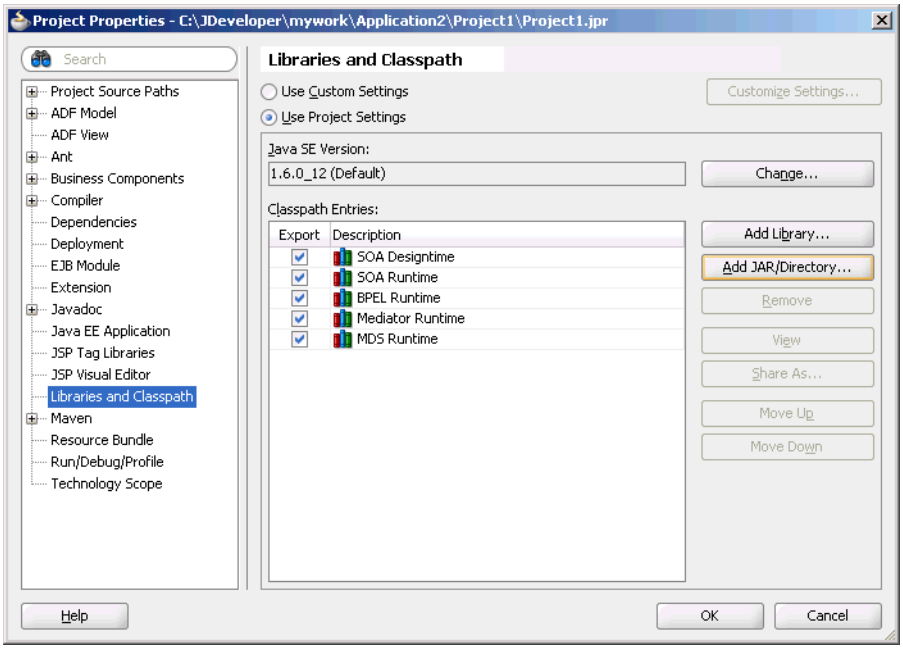
For more information on valves, see Appendix B, "Oracle JCA Adapter Valves."

Step 2 Compiling the Valves

You must use the `bpm-infra.jar` file to compile the valves. The `bpm-infra.jar` file is located at `$MW_HOME/AS11gR1SOA/soa/modules/oracle.soa.fabric_11.1.1/bpm-infra.jar`.

1. Reference the SOA project to the `bpm-infra.jar` file, by using the following procedure:
   a. In the Application Navigator, right-click the SOA project.
   b. Select **Project Properties**. The Project Properties dialog is displayed.
   c. Click **Libraries and Classpath**. The Libraries and Classpath pane is displayed as shown in Figure 4–5.

Figure 4–5 The Project Properties Dialog



   d. Click **Add Jar/Directory**. The Add Archive or Directory dialog is displayed.
   e. Browse to select the `bpm-infra.jar` file. The `Bpm-infra.jar` file is located at `$MW_HOME/AS11gR1SOA/soa/modules/oracle.soa.fabric_11.1.1/bpm-infra.jar`.
   f. Click **OK**. The `bpm-infra.jar` file is listed under Classpath Entries.
2. Compile the valves using the `bpm-infra.jar` file.
3. Make the `JAR` file containing the compiled valves available to the Oracle WebLogic Server classpath by adding the jar file to the `soainfra` domain classpath. For example, `$MW_HOME/user_projects/domains/soainfra/lib`.

> **Note:** Ensure that you compile `bpm-infra.jar` with JDK 6.0 to avoid compilation error such as `"class file has wrong version 50.0, should be 49.0"`.

Step 3 Creating a Pipeline

To configure a pipeline, you must create an XML file that conforms to the following schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
targetNamespace="http://www.oracle.com/adapter/pipeline">
  <xs:element name="pipeline">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="valves">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="valve" maxOccurs="unbounded">
               <xs:complexType mixed="true">
                <xs:attribute name="reentrant" type="xs:NMTOKEN" use="optional" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:attribute name="useStaging" type="xs:NMTOKEN" use="optional" />
    <xs:attribute name="batchNotificationHandler" type="xs:NMTOKEN" use="
optional" />
  </xs:element>
</xs:schema
```

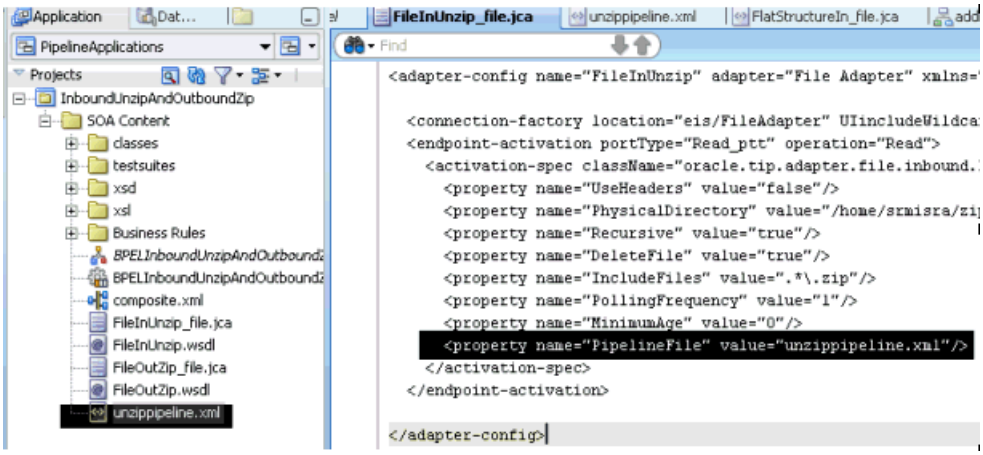
The following is a sample XML file configured for a pipeline with two valves, `SimpleUnzipValve` and `SimpleDecryptValve`:

```
<?xml version="1.0"?>
<pipeline xmlns="http://www.oracle.com/adapter/pipeline">
<valves>
        <valve>valves.SimpleUnzipValve</valve>
        <valve> valves.SimpleDecryptValve </valve>
</valves>
</pipeline>
```

Step 4 Adding the Pipeline to the SOA Project Directory

You must add the *`pipeline`*`.xml` file to the SOA project directory. This step is required to integrate the pipeline with the Oracle File or FTP Adapter. Figure 4–6 shows a sample *`pipeline`*`.xml` file (`unzippipeline.xml`) added to the `InboundUnzipAndOutboundZip` project.

Figure 4–6 Project with unzippipeline.xml File



Step 5 Registering the Pipeline

The pipeline that is a part of the SOA project must be registered by modifying the inbound JCA file, by adding the following property:

```
<property name="PipelineFile" value="pipeline.xml"/>
```

For example, in the JCA file shown in Figure 4–6, `FileInUnzip_file.jca`, the following property has been added to register an `Unzip` pipeline with an Oracle File Adapter:

```
<property name="PipelineFile" value="unzippipeline.xml"/>
```

There may be scenarios involving simple valves. A simple valve is one that does not require additional metadata such as reentrancy, and `batchNotificationHandlers`. If the scenario involves simple valves, then the pipeline can be configured as an `ActivationSpec` or an `InteractionSpec` property as shown in the following sample:

```
<adapter-config name="FileInUnzip" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
    <connection-factory location="eis/FileAdapter" UIincludeWildcard="*.zip"
adapterRef=""/>
  <endpoint-activation portType="Read_ptt" operation="Read">
    <activation-spec
className="oracle.tip.adapter.file.inbound.FileActivationSpec">
      <property../>
      <property name="PipelineValves"
value="valves.SimpleUnzipValve,valves.SimpleDecryptValve"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

Note: There is no space after the comma (,) in the `PipelineValves` property `value`.

Note: If you configure a pipeline using the `"PipelineValves"` property, then you cannot configure additional metadata such as Re-entrant Valve and Batch Notification Handler. Additional metadata can be configured only with `"PipelineFile"` that is used for the XML-based approach.

4.2.14.3 Using a Re-Entrant Valve For Processing Zip Files

The re-entrant valve enables you to process individual entries within a zip file. In a scenario that involves processing all entries within a zip file, wherein each entry is encrypted using the Data Encryption Standard (DES), you can configure the valve by adding the `reentrant="true"` attribute to the unzip valve as follows:

```
<?xml version="1.0"?>
<pipeline xmlns="http://www.oracle.com/adapter/pipeline">
<valves>
        <valve reentrant="true">valves.ReentrantUnzipValve</valve>
        <valve> valves.SimpleDecryptValve </valve>
</valves>
</pipeline>
```

In this example, the pipeline invokes the `ReentrantUnzipValve` and then the `SimpleDecryptValve` repeatedly in the same order until the entire zip file has been processed. In other words, the `ReentrantUnzipValve` is invoked first to return the data from the first zipped entry, which is then fed to the `SimpleDecryptValve` for decryption, and the final content is returned to the adapter. The process repeats until all the entries within the zip file are processed.

Additionally, the valve must set the message key using the `setMessageKey()` API. For more information refer to Appendix B.4, "An Unzip Valve for processing Multiple Files."

Error Handling For Zip Files

If there are *translation errors* for individual entries within the zip file, then entries with the translation errors are rejected and the other entries are processed.

If there are *errors during the publish operation*, then the publish operation is retried and the retry semantic holds. If the retry semantic does not hold, then the original file is rejected and the pipeline ends.

4.2.14.4 Configuring Batch Notification Handler

The `BatchNotificationHandler` API is used with the Oracle File and FTP Adapter inbound de-batchability. In a de-batching scenario, each file contains multiple messages, and some sort of bookkeeping is required for crash-recovery. This is facilitated by the `BatchNotificationHandler` API, which lets you receive notification from the pipeline whenever a batch begins, occurs, or ends. The following is the `BatchNotificationHandler` interface:

```
package oracle.tip.pc.services.pipeline;

/*
 * Whenever the caller processes de-batchable files, each file can
 * have multiple messages and this handler allows the user to plug in
 * a notification mechanism into the pipeline.
 *
 * This is particularly useful in crash recovery situations
 */

public interface BatchNotificationHandler {

      /*
       * The Pipeline instance is set by the PipelineFactory when the
       * BatchNotificationHandler instance is created
       */
      public void setPipeline(Pipeline pipeline);

      public Pipeline getPipeline();

      /*
       * Called when the BatchNotificationHandler is instantiated
       */
      public void initialize();

      /*
       * Called by the adapter when a batch begins, the implementation must
return
         * a BatchContext instance with the following information:
         * i) batchId: a unique id that will be returned every time onBatch is
invoked by called
```

```
         * ii)line/col/record/offset: for error recovery cases
        */
       public BatchContext onBatchBegin();

       /*
        * Called by the adapter when a batch is submitted. The parameter holds
the
        * line/column/record/offset for the successful batch that is published.
        * Here the implementation must save these in order to recover from
crashes
        */

       public void onBatch(BatchContext ctx);

       /*
        * Called by the adapter when a batch completes. This must be used to
clean
        * up
        */
       public void onBatchCompletion(boolean success);
}
```

To use a pipeline with de-batching, you must configure the pipeline with a `BatchNotificationHandler` instance as follows:

```
<?xml version="1.0"?>
<pipeline xmlns="http://www.oracle.com/adapter/pipeline"
batchNotificationHandler="oracle.tip.pc.services.pipeline.ConsoleBatchNotification
Handler">
<valves>
      <valve reentrant="true">valves.SimpleUnzipValve</valve>
      <valve>valves.SimpleDecryptValve</valve>
</valves>
</pipeline>
```

4.2.15 Error Handling

The Oracle File Adapter and Oracle FTP Adapter provide inbound error handling capabilities, such as the `uniqueMessageSeparator` property.

In the case of debatching (multiple messages in a single file), messages from the first bad message to the end of the file are rejected. If each message has a unique separator and that separator is not part of any data, then rejection can be more fine grained. In these cases, you can define a `uniqueMessageSeparator` property in the schema element of the native schema to have the value of this unique message separator. This property controls how the adapter translator works when parsing through multiple records in one file (debatching). This property enables recovery even when detecting bad messages inside a large batch file. When a bad record is detected, the adapter translator skips to the next unique message separator boundary and continues from there. If you do not set this property, then all records that follow the record with errors are also rejected.

The following schema file provides an example of using the `uniqueMessageSeparator` property:

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            targetNamespace="http://TargetNamespace.com/Reader"
```

```
            xmlns:tns="http://TargetNamespace.com/Reader"
            elementFormDefault="qualified" attributeFormDefault="unqualified"
            nxsd:encoding="US-ASCII" nxsd:stream="chars"
            nxsd:version="NXSD" nxsd:uniqueMessageSeparator="${eol}">
  <xsd:element name="emp-listing">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="emp" minOccurs="1" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="GUID" type="xsd:string" nxsd:style="terminated"
                           nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Designation" type="xsd:string"
                           nxsd:style="terminated" nxsd:terminatedBy=","
                           nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Car" type="xsd:string" nxsd:style="terminated"
                           nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Labtop" type="xsd:string"
                           nxsd:style="terminated" nxsd:terminatedBy=","
                           nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Location" type="xsd:string"
                           nxsd:style="terminated" nxsd:terminatedBy=","
                           nxsd:quotedBy="&quot;">
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
<!--NXSDWIZ:D:\work\jDevProjects\Temp_BPEL_process\Sample2\note.txt:-->
<!--USE-HEADER:false:-->
```

For information about handling rejected messages, connection errors, and message errors, see Section 2.21.1, "Handling Rejected Messages".

4.2.15.1 Sending a Malformed XML File to a Local File System Folder

During an Inbound Read operation, if a malformed XML file is read, the malformed file results in an error. The errored file is by default sent to the remote file system for archival.

The errored file can be archived at a *local* file system by specifying the `useRemoteErrorArchive` property in the `jca` file and setting that property to `false`.

The default value for this property is `true`.

4.2.16 Threading Model

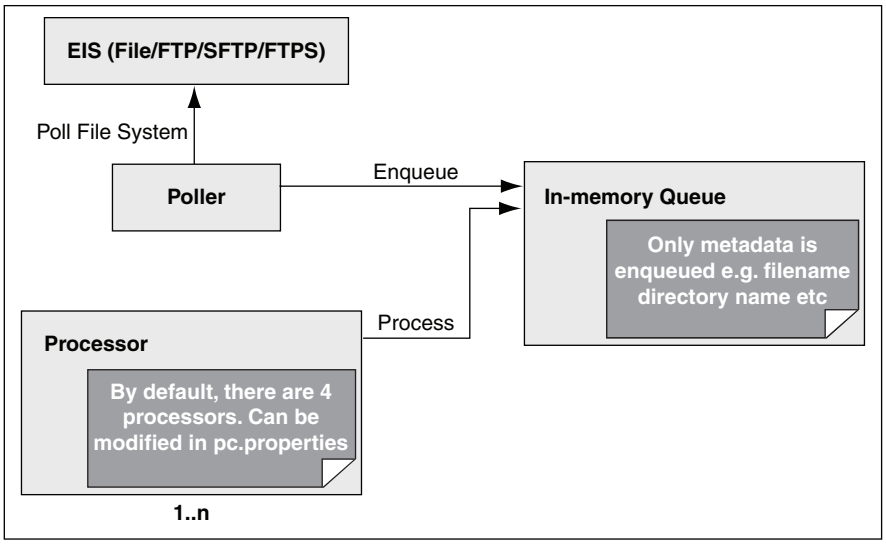
This section describes the threading models that Oracle File and FTP Adapters support. An understanding of the threading models is required to throttle or de-throttle the Oracle File and FTP Adapters. The Oracle File and FTP Adapters use the following threading models:

- Default Threading Model
- Modified Threading Model

4.2.16.1 Default Threading Model

In the default threading model, a poller is created for each inbound Oracle File or FTP Adapter endpoint. The poller enqueues file metadata into an in-memory queue, which is processed by a global pool of processor threads. Figure 4–7 shows a default threading model.

Figure 4–7 Default Threading Model



The following steps highlight the functioning of the default threading model:

1. The poller periodically looks for files in the input directory. The interval at which the poller looks for files is specified using the `PollingFrequency` parameter in the inbound `JCA` file.
2. For each new file that the poller detects in the configured inbound directory, the poller enqueues information such as file name, file directory, modified time, and file size into an internal in-memory queue.

   Note: New files are ones that are not being processed.

3. A global pool of processor worker threads wait to process from the in-memory queue.
4. Processor worker threads pick up files from the internal queue, and perform the following actions:
   a. Stream the file content to an appropriate translator (for example, a translator for reading text, binary, XML, or opaque data.)
   b. Publishes the XML result from the translator to the SCA infrastructure.
   c. Performs the required postprocessing, such as deletion or archival after the file is processed.

4.2.16.2 Modified Threading Model

You can modify the default threading behavior of Oracle File and FTP Adapters. Modifying the threading model results in a modified throttling behavior of Oracle File and FTP Adapters. The following sections describe the modified threading behavior of the Oracle File and FTP Adapters:

- Single Threaded Model
- Partitioned Threaded Model

Single Threaded Model

The single threaded model is a modified threaded model that enables the poller to assume the role of a processor. The poller thread processes the files in the same thread. The global pool of processor threads is not used in this model. You can define the property for a single threaded model in the inbound JCA file as follows:

```
<activation-spec className="oracle.tip.adapter.file.inbound.FileActivationSpec">
   <property../>
   <property name="SingleThreadModel" value="true"/>
   <property../>
</activation-spec>
```

Partitioned Threaded Model

The partitioned threaded model is a modified threaded model in which the in-memory queue is partitioned and each composite application receives its own in-memory queue. The Oracle File and FTP Adapters are enabled to create their own processor threads rather than depend on the global pool of processor worker threads for processing the enqueued files. You can define the property for a partitioned model in the inbound JCA file as follows:

```
<activation-spec
className="oracle.tip.adapter.file.inbound.FileActivationSpec">
  <property../>
  <property name="ThreadCount" value="4"/>
  <property../>
</activation-spec>
```

In the preceding example for defining the property for a partitioned model:

- If the `ThreadCount` property is set to `0`, then the threading behavior is like that of the single threaded model.
- If the `ThreadCount` property is set to `-1`, then the global thread pool is used, as in the default threading model.
- The maximum value for the `ThreadCount` property is `40`.

4.2.17 Performance Tuning

The Oracle File and FTP Adapters support the performance tuning feature by providing knobs to throttle the inbound and outbound operations. The Oracle File and FTP Adapters also provide knobs that you can use to tune the performance of outbound operations.

For more information about performance tuning, see "Oracle JCA Adapters for Files/FTP" in the *Oracle Fusion Middleware Performance and Tuning Guide*.

4.2.18 High Availability

The Oracle File and FTP Adapters support the high availability feature for the active-active topology with Oracle BPEL Process Manager and Mediator service engines. It supports this feature for both inbound and outbound operations.

4.2.19 Multiple Directories

The Oracle File and FTP Adapters support polling multiple directories within a single activation. You can specify multiple directories in JDeveloper as opposed to a single directory. This is applicable to both physical and logical directories.

Note: If the inbound Oracle File Adapter is configured for polling multiple directories for incoming files, then all the top-level directories (inbound directories where the input files appear) must exist before the file reader starts polling these directories.

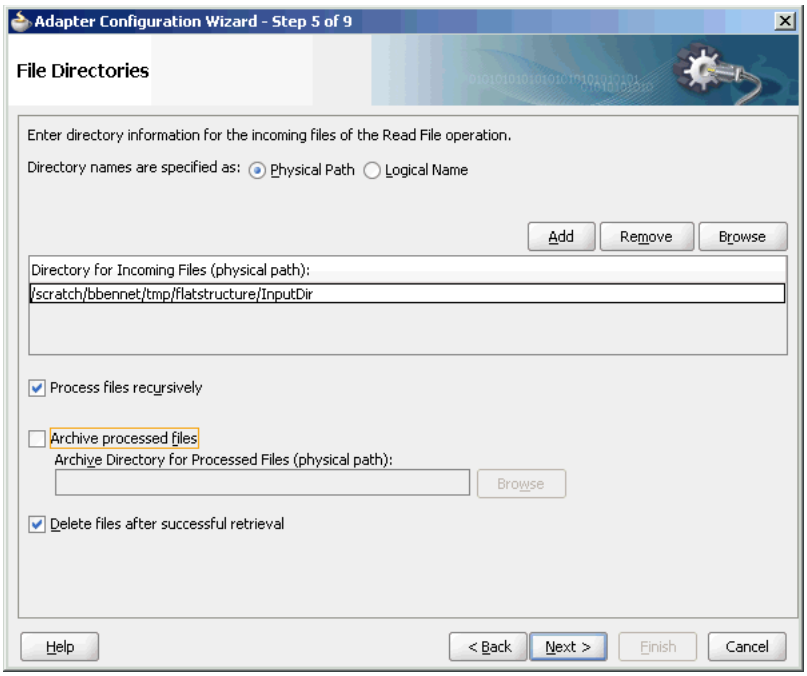
After selecting the inbound directory or directories, you can also specify whether the subdirectories must be processed recursively. If you check the Process Files Recursively option, then the directories would be processed recursively. By default, this option is selected, in the File Directories page, as shown in Figure 4–8.

When you choose multiple directories, the generated JCA files use semicolon(;) as the separator for these directories. However, you can change the separator to something else. If you do so, manually add `DirectorySeparator="`*`chosen separator`*`"` in the generated JCA file. For example, to use comma (,) as the separator, you must first change the separator to "," in the Physical directory and then add `<property name="DirectorySeparator" value=",/>`, in the JCA file.

Additionally, if you choose to process directories recursively and one or more subdirectories do not have the appropriate permissions, the inbound adapter throws an exception during processing. To ignore this exception, you must define a binding property with the name `ignoreListingErrors` in your `composite.xml` as shown in the following example:

```
<service name="FlatStructureIn">
<interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/file/FlatStructureIn/#wsdl.inte
rface(Read_ptt)"/>
<binding.jca config="FlatStructureIn_file.jca">
<property name="ignoreListingErrors" type="xs:string"
many="false">true</property>
</binding.jca>
</service>
```

Figure 4–8 The Adapter Configuration Wizard - File Directories Page



4.2.20 Append Mode

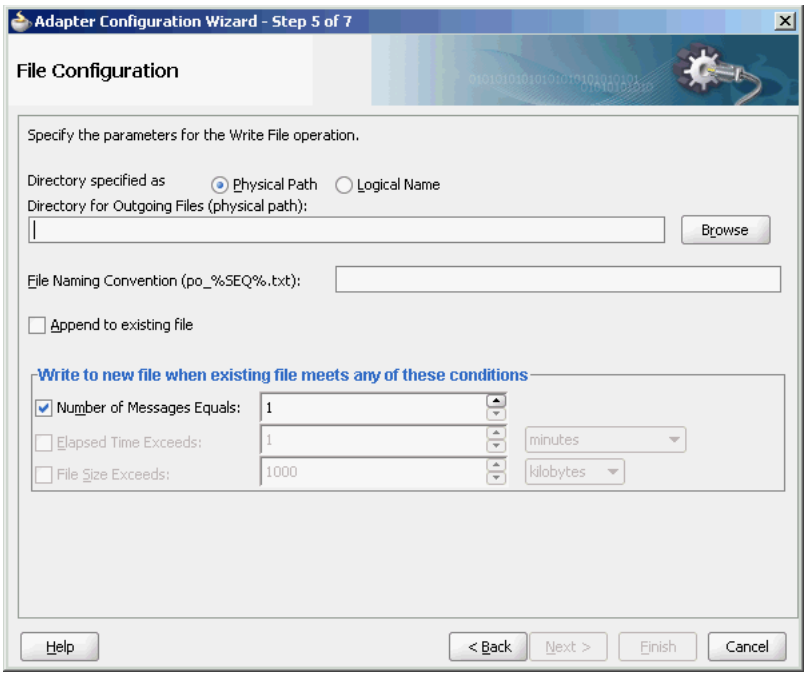
The Oracle File and FTP Adapters enable you to configure outbound interactions that append to an existing file. The Append to Existing File option enables the outbound invoke to write to the same file. There are two ways in which you can append to a file name:

- Statically - in the JCA file for the outbound Oracle File Adapter
- Dynamically - using the header mechanism

> **Note:** The append mode is not supported for SFTP scenarios, where instead of appending to the existing file, the file is overwritten.

When you select the Append to existing file option in the File Configuration page, the batching options such as Number of Messages Equals, Elapsed Time Exceeds, File Size Exceeds options are disabled. Figure 4–9 displays the Append to Existing File option.

Figure 4–9 The Adapter Configuration Wizard - File Configuration Page



Batching option is disabled if "Append" is chosen in the wizard. In addition, the following error message is displayed if the user specifies a dynamic file naming convention as opposed to a static file naming convention:

```
You cannot choose to Append Files and use a dynamic file naming convention at the
same time
```

If you are using the "Append" functionality in Oracle FTP Adapter, ensure that the FTP server supports the "APPE" command.

4.2.21 Recursive Processing of Files Within Directories in Oracle FTP Adapter

In earlier versions of the Oracle SOA Suite, the inbound Oracle FTP Adapter used the `NLST` (Name List) FTP command to read a list of file names from the FTP server. However, the NLST command does not return directory names and therefore does not allow recursive processing within directories. In the 11*g* release, the Oracle FTP Adapter uses the `LIST` command, instead.

However, the response from the `LIST` command is different for different FTP servers. To incorporate the subtle differences in results from the `LIST` command in a standard manner, the following parameters are added to the deployment descriptor for Oracle FTP Adapter:

- **defaultDateFormat**: This parameter specifies the default date format value. On the FTP server, this is the value for files that are older. The default value for this parameter is `MMM d yyyy` as most UNIX-type FTP servers return the last modified time stamp for older files in the `MMM d yyyy` format. For example, `Jan 31 2006`.

  You can find the default date format for your FTP server by using the `ls -l` command by using a FTP command-line client. For example, `ls -l` on a vsftpd server running on Linux returns the following:

  ```
  -rw-r--r--    1 500      500           377 Jan 22 2005 test.txt
  ```

For Microsoft Windows NT FTP servers, the `defaultDateFormat` is `MM-dd-yy hh:mma`, for example, `03-24-09 08:06AM <DIR> oracle`.

- **recentDateFormat**: This parameter specifies the recent date format value. On the FTP server, this is the value for files that were recently created.

  The default value for this parameter is `MMM d HH:mm` as most UNIX-type FTP servers return the last modified date for recently created files in `MMM d HH:mm` format, for example, `Jan 31 21:32`.

  You can find the default date format for your FTP server by using the `ls -l` command from an FTP command-line client. For example, `ls -l` on a vsftpd server running on Linux returns the following:

```
150 Here comes the directory listing.
-rw-r--r--    1 500      500           377 Jan 30 21:32 address.txt
-rw-r--r--    1 500      500           580 Jan 3121:32 container.txt
.....................................................................
......
```

  For Microsoft Windows NT FTP servers, the `recentDateFormat` parameter is in the `MM-dd-yy hh:mma`, format, for example, `03-24-09 08:06AM <DIR> oracle`.

- **serverTimeZone**: The server time zone, for example, *America/Los_Angeles*. If this parameter is set to blank, then the default time zone of the server running the Oracle FTP Adapter is used.
- **listParserKey:** Directs the Oracle FTP Adapter on how it should parse the response from the `LIST` command. The default value is UNIX, in which case the Oracle FTP Adapter uses a generic parser for UNIX-like FTP servers. Apart from `UNIX`, the other supported values are `WIN` and `WINDOWS`, which are specific to the Microsoft Windows NT FTP server.

  Note: The locale language for the FTP server can be different from the locale language for the operating system. Do not assume that the locale for the FTP server is the same locale for the operating system it is running on. You must set the `serverLocaleLanguage`, `serverLocaleCountry`, and `serverLocaleVariant` parameters in such cases.

- **serverLocaleLanguage**: This parameter specifies the locale construct for language.
- **serverLocaleCountry**: This parameter specifies the locale construct for country.
- **serverLocaleVariant**: This parameter specifies the locale construct for variant.

Configure the Parameters in the Deployment Descriptor

The standard date formats of an FTP server are usually configured when the FTP server is installed. If your FTP server uses a format "MMM d yyyy" for defaultDateFormat and "MMM d HH:mm" for recentDateFormat, then your Oracle FTP Adapter must use the same formats in its corresponding deployment descriptor.

If you enter "ls -l" from a command-line FTP, then you can see the following:

```
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r--    1 500      500           377 Jan 22 21:32 1.txt
-rw-r--r--    1 500      500           580 Jan 22 21:32 2.txt
.....................................................................
```

This is the recentDateFormat parameter for your FTP server, for example MMM d HH:mm (Jan 22 21:32). Similarly, if your server has an old file, then the server does not show the hour and minute part and it shows the following:

```
-rw-r--r--    1 500      500           377 Jan 22 2005 test.txt
```

This is the default date format, for example MMM d yyyy (Jan 22 2005).

Additionally, the serverTimeZone parameter is used to by the Oracle FTP Adapter to parse time stamps for FTP server running in a specific time zone. The value for this is either an abbreviation such as "PST" or a full name such as "America/Los_Angeles".

Additionally, the FTP server might be running on a different locale. The serverLocaleLanguage, serverLocaleCountry, and serverLocaleVariant parameters are used to construct a locale from language, country, variant where

- language is a lowercase two-letter ISO-639 code, for example, en
- country is an uppercase two-letter ISO-3166 code, for example, US
- variant is a vendor and browser-specific code.

If these locale parameters are absent, then the Oracle FTP Adapter uses the system locale to parse the time stamp.

Additionally, if the FTP server is running on a different system than the SOA suite, then you must handle the time zone differences between them. You must convert the time difference between the FTP server and the system running the Oracle FTP Adapter to milliseconds and add the value as a binding property:"timestampOffset" in the `composite.xml`.

For example, if the FTP server is six hours ahead of your local time, you must add the following endpoint property to your service or reference:

```
<service name="FTPDebatchingIn">
    <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/ftp/FTPDebatchingIn/#wsdl.
interface(Get_ptt)"/>
    <binding.jca config="DebatchingIn_ftp.jca">
<property name=" timestampOffset" type="xs:string" many="false" source=""
override="may"> 21600000</property>
    </binding.jca>
  </service>.
```

Some FTP servers do not work well with the `LIST` command. In such cases, use the `NLST` command for listing, but you cannot process directories recursively with `NLST`.

To use the `NLST` command, then you must add the following property to the JCA file, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FTPDebatchingIn" adapter="Ftp Adapter"
 xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
<connection-factory location="eis/Ftp/FtpAdapter" UIincludeWildcard="*.txt"
 adapterRef=""/>
    <activation-spec
className="oracle.tip.adapter.ftp.inbound.FTPActivationSpec">
       ……..
       ……..
      <property name="UseNlst" value="true"/>
    </activation-spec>
  </endpoint-activation>
```

```
</adapter-config>
```

4.2.22 Securing Enterprise Information System Credentials

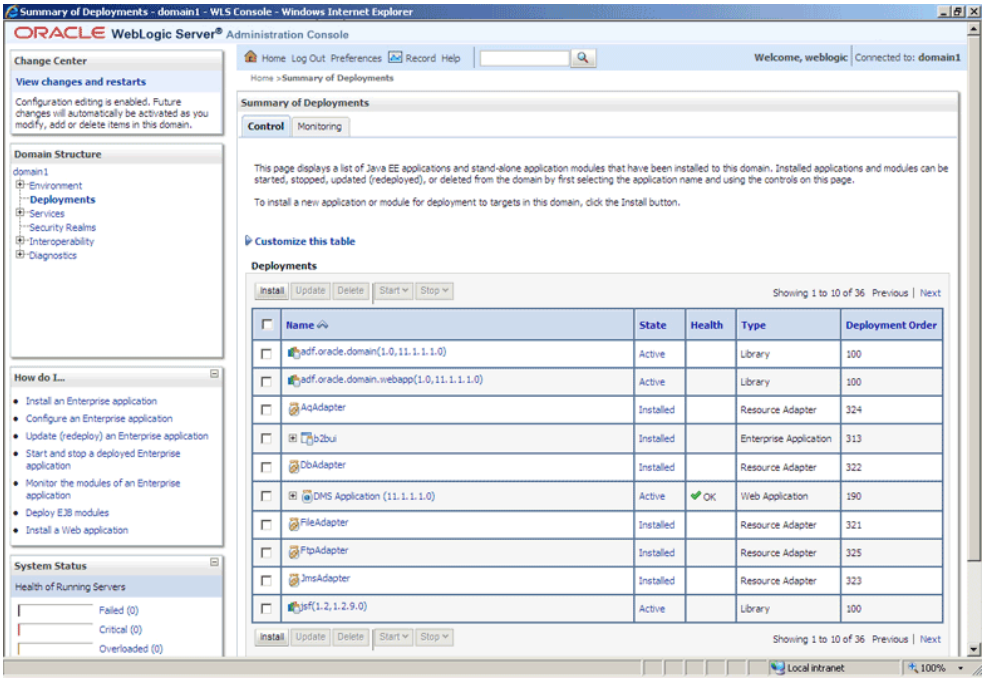
When a resource adapter makes an outbound connection with an Enterprise Information System (EIS), it must sign on with valid security credentials. In accordance with the J2CA 1.5 specification, Oracle WebLogic Server supports both container-managed and application-managed sign-on for outbound connections. At run time, Oracle WebLogic Server determines the chosen sign-on mechanism, based on the information specified in either the invoking client component's deployment descriptor or the `res-auth` element of the resource adapter deployment descriptor. This section describes the procedure for securing the user name and password for Oracle JCA Adapters by using Oracle WebLogic Server container-managed sign-on.

Both, Oracle WebLogic Server and EIS maintain independent security realms. A container-managed sign-on enables you to sign on to Oracle WebLogic Server and also be able to use applications that access EIS through a resource adapter without having to sign on separately to the EIS. Container-managed sign-on in Oracle WebLogic Server uses credential mappings. The credentials (user name/password pairs or security tokens) of Oracle WebLogic security principals (authenticated individual users or client applications) are mapped to the corresponding credentials required to access EIS. You can configure credential mappings for applicable security principals for any deployed resource adapter.

To configure credential mappings, you can specify the user names and passwords in the `weblogic-ra.xml` file for the corresponding adapter or perform the following procedure by accessing the Oracle WebLogic Server Administration Console:

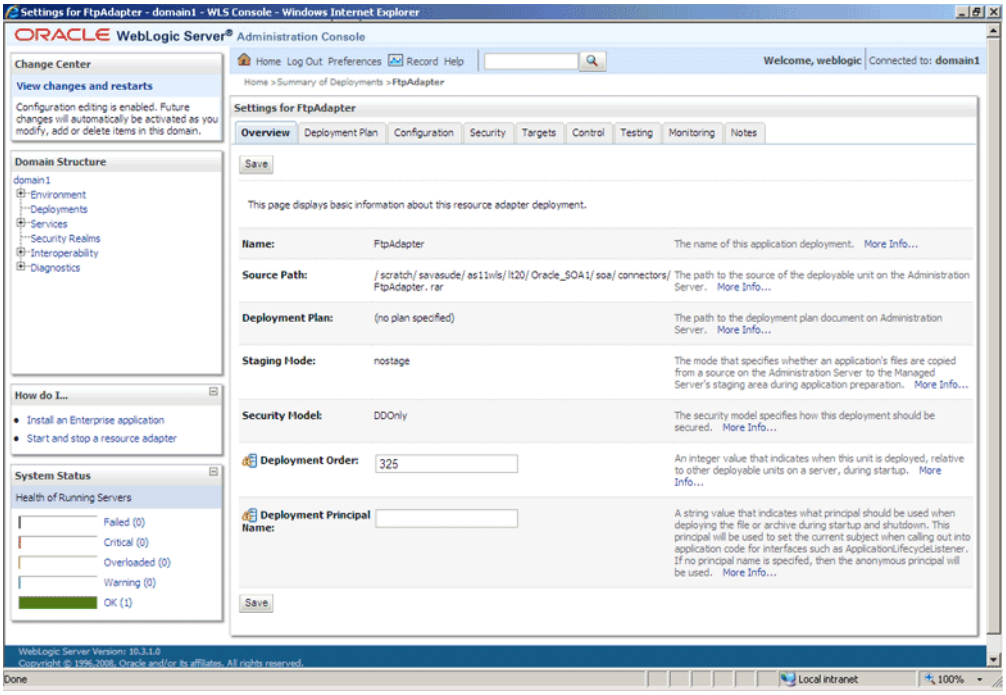
1. Log in to the Oracle WebLogic Server Administration Console.
2. Click **Deployments** in the Domain Structure pane. The deployed applications and adapters are displayed, as shown in Figure 4–10.

Figure 4–10 The Oracle WebLogic Server Administration Console - Summary of Deployments Page



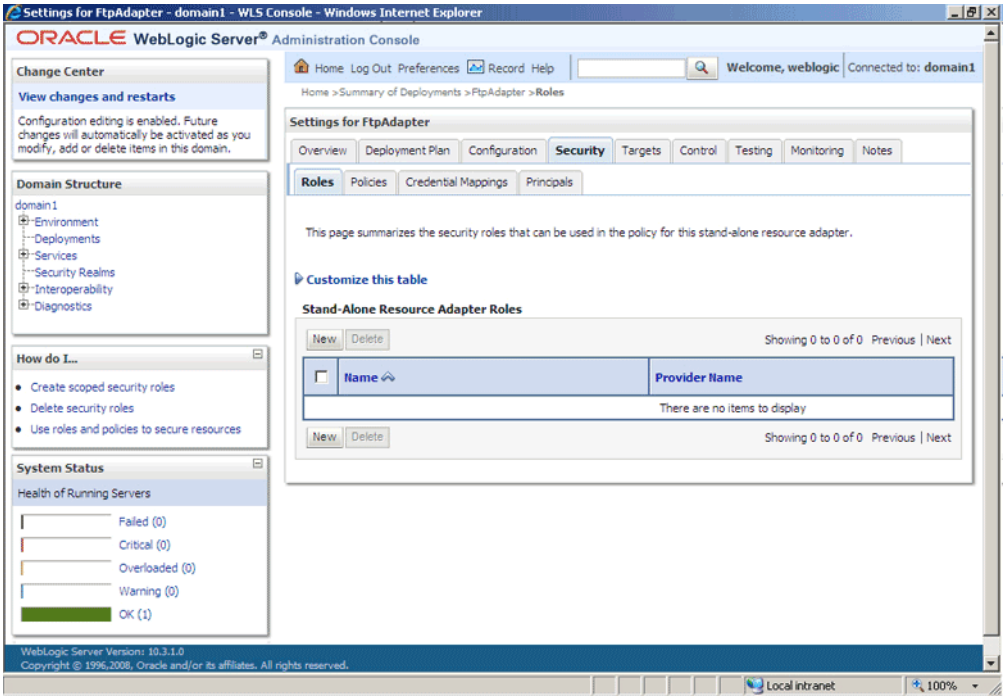
3. Click the adapter for which you must create the security credentials. For example, click `FtpAdapter`. The Settings for FtpAdapter page is displayed, as shown in Figure 4–11.

Figure 4–11 The Oracle WebLogic Server Administration Console - Settings for FTPAdapter Page



4. Click the **Security** tab. The Settings for FTPAdapter page with the Stand-Alone Resource Adapter Roles pane displayed, as shown in Figure 4–12.

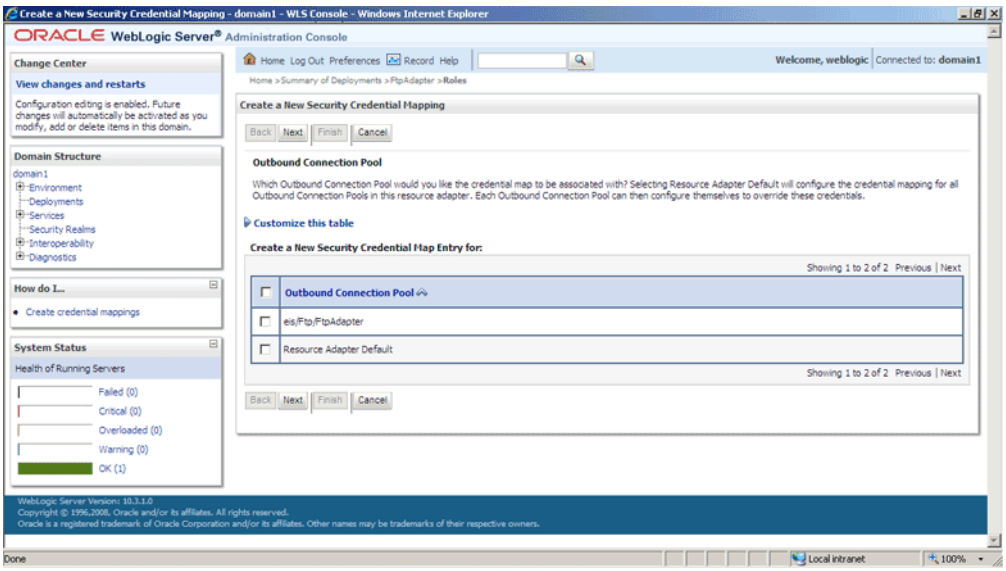
Figure 4–12 The Oracle WebLogic Server Administration Console - Settings for FTPAdapter Page



5. Click the **Credential Mappings** tab.

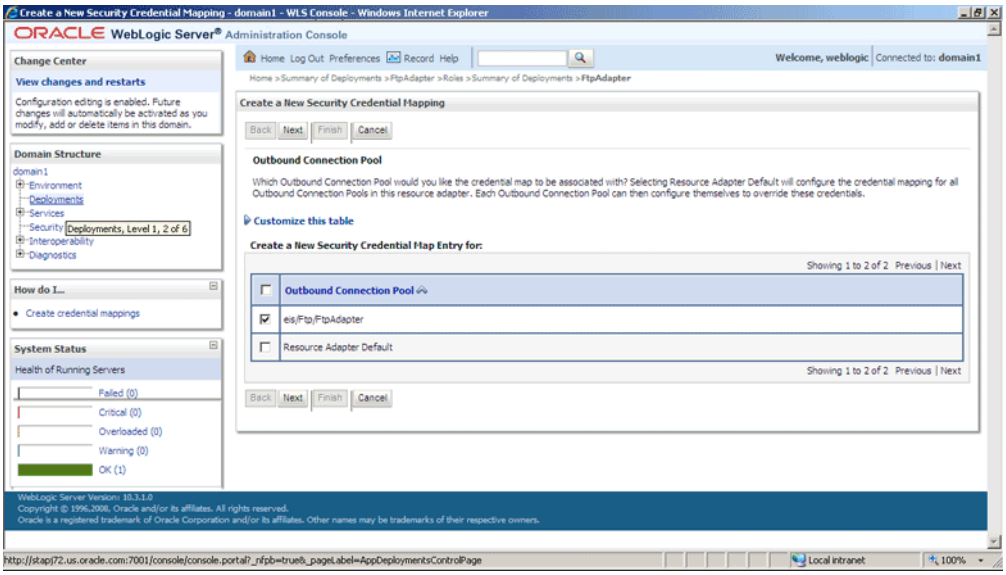
6. Click **New** in the Credential Mappings pane. The Create a New Security Credential Mapping page is displayed, as shown in Figure 4–13.

Figure 4–13 The Oracle WebLogic Server Administration Console - Create a New Security Credential Mapping Page



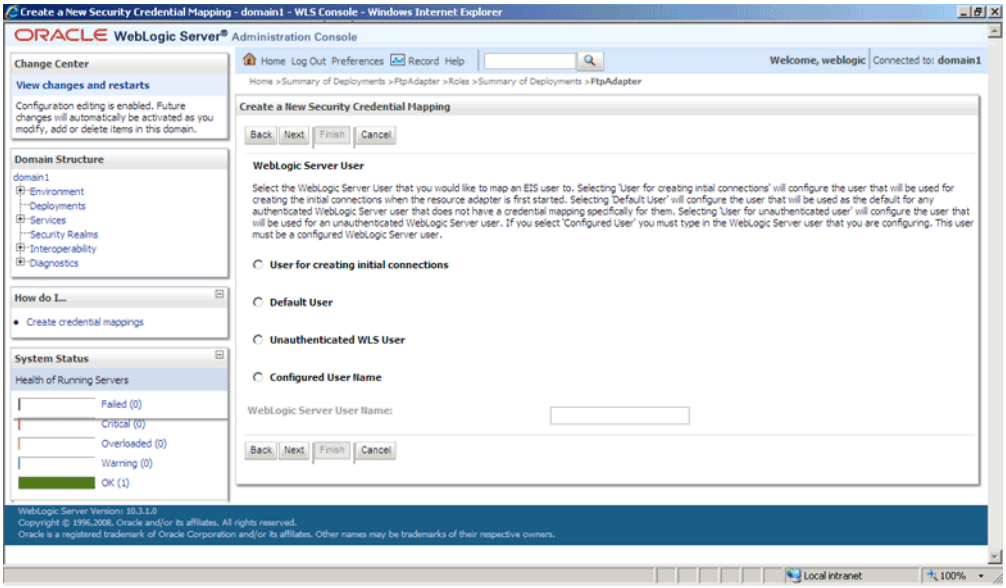
7. Select **eis/Ftp/FtpAdapter** (JNDI for Oracle FTP Adapter) to create a security credential map entry for Oracle FTP Adapter, as shown in Figure 4–14.

Figure 4–14 The Oracle WebLogic Server Administration Console - Create a New Security Credential Mapping Page



8. Click **Next**. The Create a New Security Credential Mapping – WebLogic Server User page is displayed, as shown in Figure 4–15.

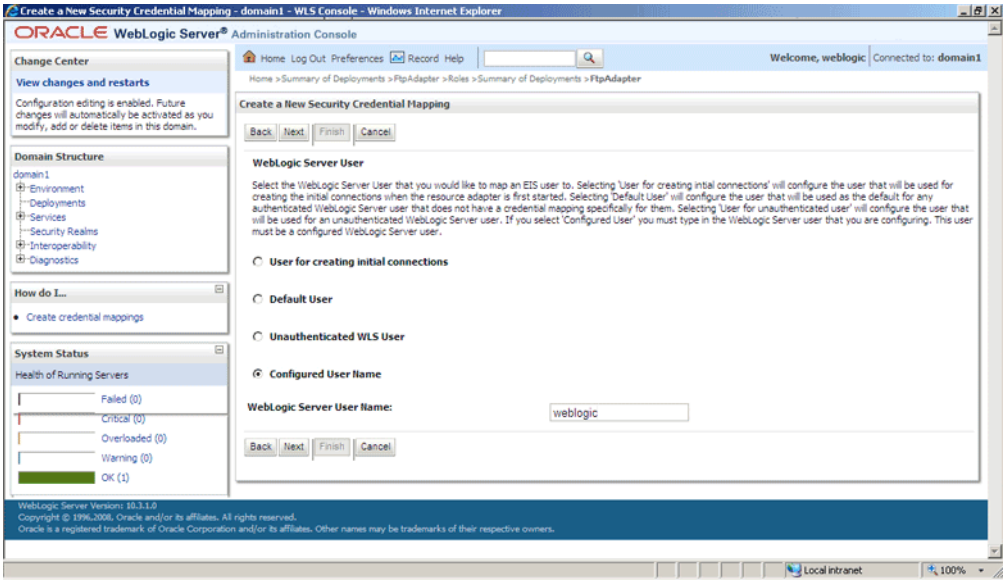
Figure 4–15 The Oracle WebLogic Server Administration Console - Create a New Security Credential Mapping Page



Note: Credential mapping is not supported for the **User for creating initial connections** and **Unauthenticated WLS User** options.

9. Select **Configured User Name** and enter the Oracle WebLogic Server user name in the **WebLogic Server User Name** field, as shown in Figure 4–16. For example, enter `weblogic`, which is the default user name.

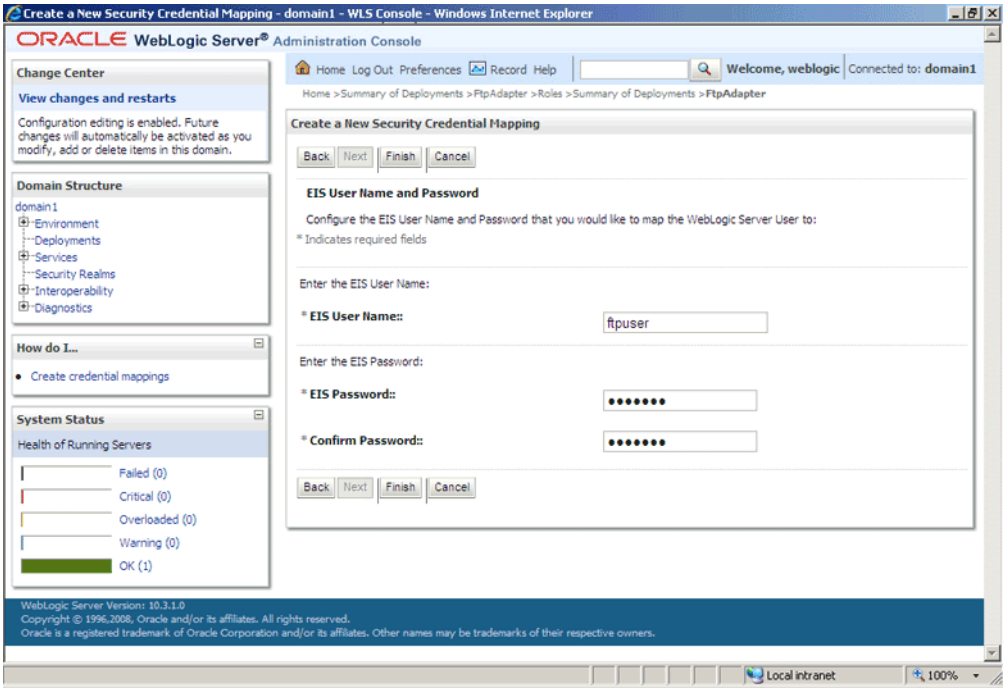
Figure 4–16 The Oracle WebLogic Server Administration Console - Create a New Security Credential Mapping Page



10. Click **Next**. The Create a New Security Credential Mapping – EIS User Name and Password page is displayed.

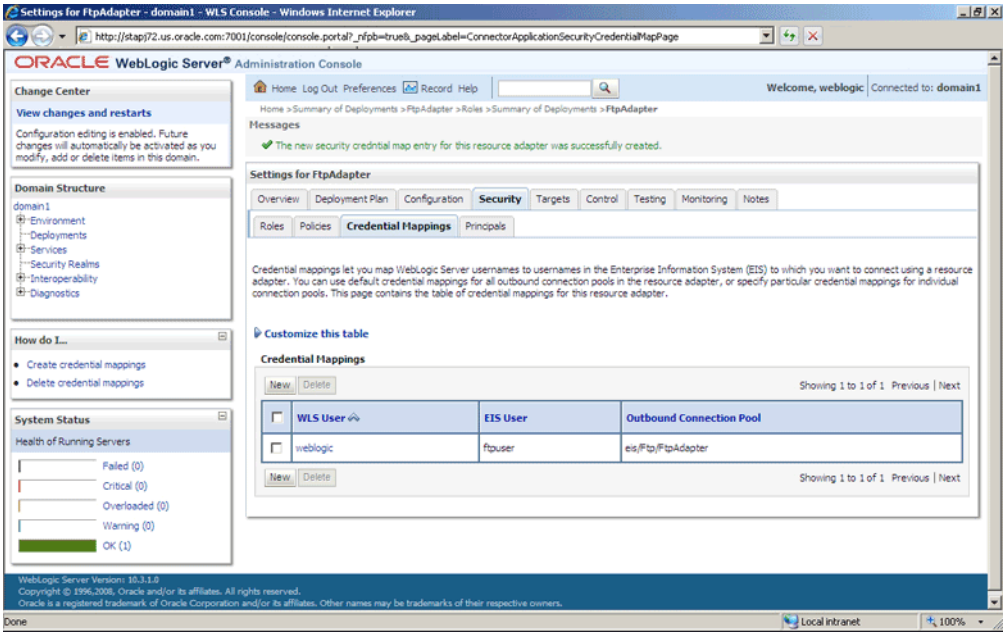
11. Enter the EIS user name in the **EIS User Name** field, the EIS password in the **EIS Password** field, and then reenter the EIS password in the **Confirm Password** field to confirm the password, as shown in Figure 4–17.

Figure 4–17 The Oracle WebLogic Server Administration Console - Create a New Security Credential Mapping Page



12. Click **Finish**. The new security credential mapping is created, as shown in Figure 4–18.

Figure 4–18 The Oracle WebLogic Server Administration Console - Settings for FTPAdapter Page



4.3 Oracle File and FTP Adapter Concepts

The Oracle File and FTP Adapters concepts are discussed in the following sections:

- Section 4.3.1, "Oracle File Adapter Read File Concepts"
- Section 4.3.2, "Oracle File Adapter Write File Concepts"
- Section 4.3.3, "Oracle File Adapter Synchronous Read Concepts"
- Section 4.3.4, "Oracle File Adapter File Listing Concepts"
- Section 4.3.5, "Oracle FTP Adapter Get File Concepts"
- Section 4.3.6, "Oracle FTP Adapter Put File Concepts"
- Section 4.3.7, "Oracle FTP Adapter Synchronous Get File Concepts"
- Section 4.3.8, "Oracle FTP Adapter File Listing Concepts"

4.3.1 Oracle File Adapter Read File Concepts

In the inbound direction, the Oracle File Adapter polls and reads files from a file system for processing. This section provides an overview of the inbound file reading capabilities of the Oracle File Adapter. You use the Adapter Configuration Wizard to configure the Oracle File Adapter for use with a BPEL process or a Mediator. Configuring the Oracle File Adapter creates an inbound `WSDL` and `JCA` file pair.

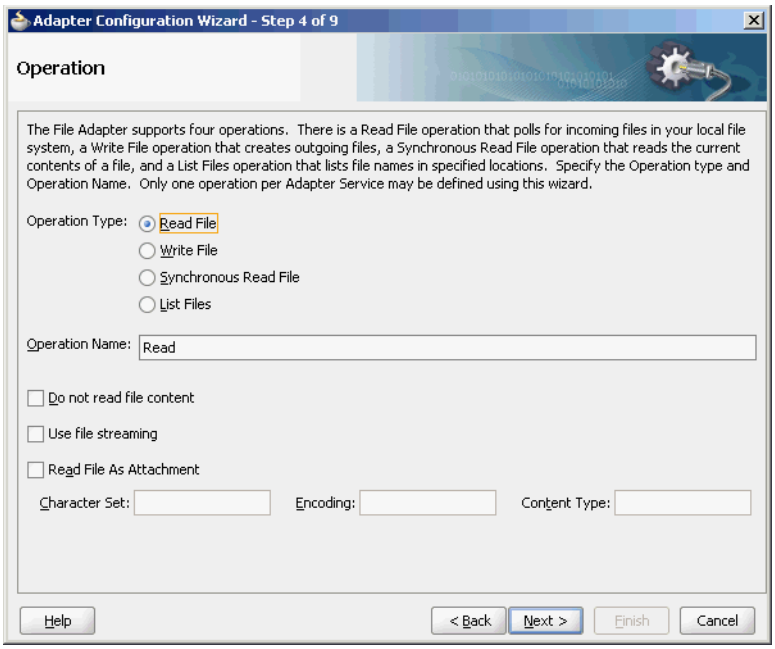
The following sections describe the Oracle File Adapter read file concepts:

- Section 4.3.1.1, "Inbound Operation"
- Section 4.3.1.2, "Inbound File Directory Specifications"
- Section 4.3.1.3, "File Matching and Batch Processing"
- Section 4.3.1.4, "File Polling"
- Section 4.3.1.5, "Postprocessing"
- Section 4.3.1.6, "Native Data Translation"
- Section 4.3.1.7, "Inbound Service"
- Section 4.3.1.8, "Inbound Headers"

4.3.1.1 Inbound Operation

For inbound operations with the Oracle File Adapter, select the **Read File** operation, as shown in Figure 4–19.

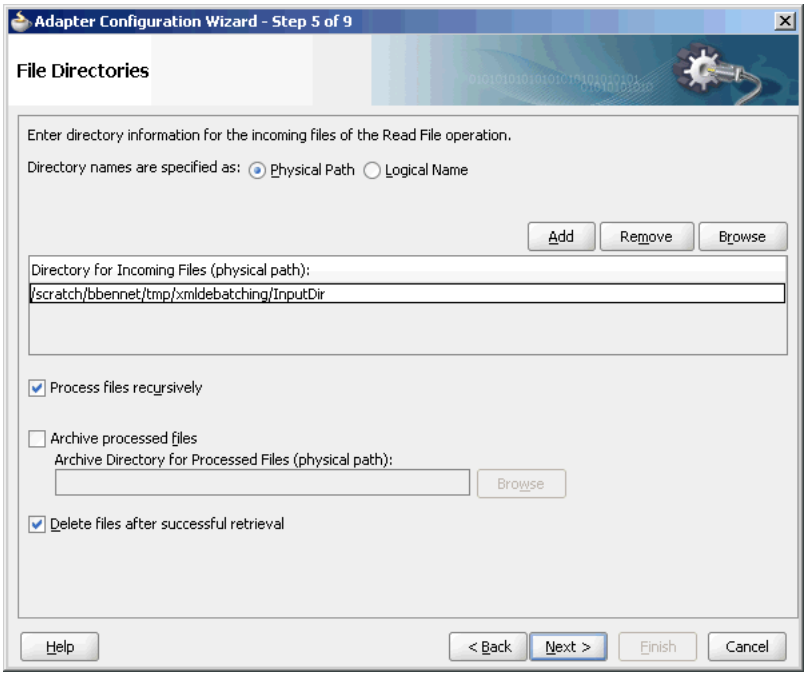
Figure 4–19 Selecting the Read File Operation



4.3.1.2 Inbound File Directory Specifications

The File Directories page of the Adapter Configuration Wizard shown in Figure 4–20 enables you to specify information about the directory to use for reading inbound files and the directories in which to place successfully processed files. You can choose to process files recursively within directories. You can also specify multiple directories.

Figure 4–20 The Adapter Configuration Wizard - Specifying Incoming Files



The following sections describe the file directory information to specify:

- Section 4.3.1.2.1, "Specifying Inbound Physical or Logical Directory Paths in SOA Composite"
- Section 4.3.1.2.2, "Archiving Successfully Processed Files"
- Section 4.3.1.2.3, "Deleting Files After Retrieval"

4.3.1.2.1 Specifying Inbound Physical or Logical Directory Paths in SOA Composite

You can specify inbound directory names as physical or logical paths in the composite involving Oracle BPEL PM and Mediator. Physical paths are values such as `c:\inputDir`.

> **Note:** If the inbound Oracle File Adapter is configured for polling multiple directories for incoming files, then all the top-level directories (inbound directories where the input file appears) must exist before the file reader starts polling these directories.

In the composite, logical properties are specified in the inbound `JCA` file and their logical-physical mapping is resolved by using binding properties. You specify the logical parameters once at design time, and you can later modify the physical directory name as needed.

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

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FlatStructureIn" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">

  <connection-factory location="eis/FileAdapter" UIincludeWildcard="*.txt"
adapterRef=""/>
  <endpoint-activation operation="Read">
    <activation-spec
className="oracle.tip.adapter.file.inbound.FileActivationSpec">
      <property name="UseHeaders" value="false"/>
      <property name="LogicalDirectory" value="InputFileDir"/>
      <property name="Recursive" value="true"/>
      <property name="DeleteFile" value="true"/>
      <property name="IncludeFiles" value=".*\.txt"/>
      <property name="PollingFrequency" value="10"/>
      <property name="MinimumAge" value="0"/>
      <property name="OpaqueSchema" value="false"/>
    </activation-spec>
  </endpoint-activation>

</adapter-config>
```

In the `composite.xml` file, you then provide the physical parameter values (in this case, the directory path) of the corresponding logical `ActivationSpec` or `InteractionSpec`. This resolves the mapping between the logical directory name and actual physical directory name.

```
<service name="FlatStructureIn">
    <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/file/FlatStructureIn/#wsdl.
interface(Read_ptt)"/>
    <binding.jca config="FlatStructureIn_file.jca">
<property name=" InputFileDir" type="xs:string" many="false" source=""
```

```
override="may"> /home/user/inputDir</property>

 </binding.jca>
</service>
```

4.3.1.2.2 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 Section 4.3.1.2.1, "Specifying Inbound Physical or Logical Directory Paths in SOA Composite."

4.3.1.2.3 Deleting Files After Retrieval

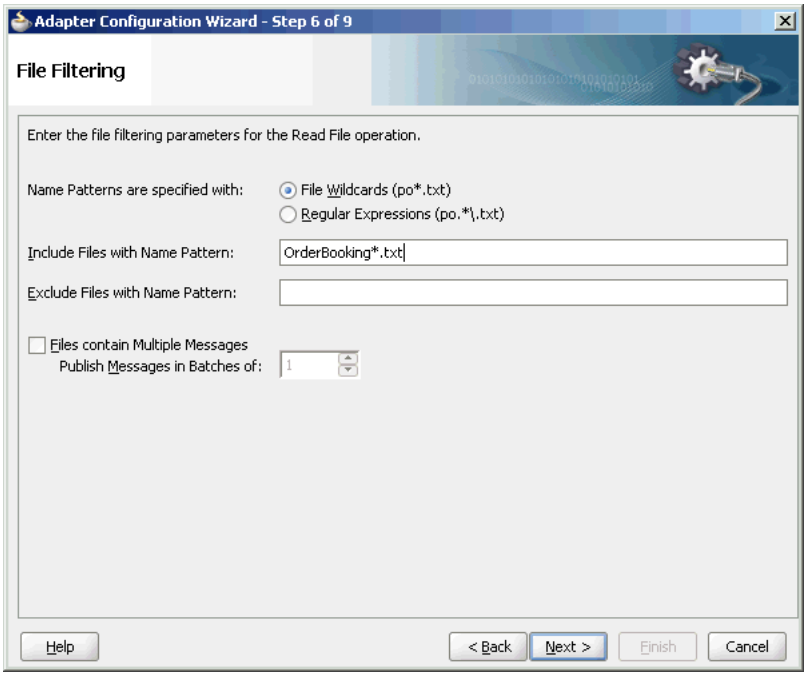
This option enables you to specify whether to delete files after a successful retrieval. If this check box is not selected, processed files remain in the inbound directory but are ignored. Only files with modification dates more recent than the last processed file are retrieved. If you place another file in the inbound directory with the same name as a file that has been processed but the modification date remains the same, then that file is not retrieved.

4.3.1.3 File Matching and Batch Processing

The File Filtering page of the Adapter Configuration Wizard shown in Figure 4–21 enables you to specify details about the files to retrieve or ignore.

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

Figure 4–21 The Adapter Configuration Wizard-File Filtering Page



The following sections describe the file filtering information to specify:

- Section 4.3.1.3.1, "Specifying a Naming Pattern"

- Section 4.3.1.3.2, "Including and Excluding Files"
- Section 4.3.1.3.4, "Debatching Multiple Inbound Messages"

4.3.1.3.1 Specifying a Naming Pattern

Specify the naming convention that the Oracle File Adapter uses to poll for inbound files. You can also specify the naming convention for files you do not want to process. Two naming conventions are available for selection. The Oracle File Adapter matches the files that appear in the inbound directory.

- File wildcards (`po*.txt`)

  Retrieves all files that start with `po` and end with `.txt`. This convention conforms to Windows operating system standards.

- Regular expressions (`po.*\.txt`)

  Retrieves all files that start with `po` and end with `.txt`. This convention conforms to Java Development Kit (JDK) regular expression (regex) constructs.

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 file name, plus the complete directory path). Some operating systems also have restrictions on the use of specific characters in file names. For example, Windows operating systems do not allow characters such as `backslash(\)`, slash (`/`), colon (`:`), asterisk (`*`), left angle bracket (<), right angle bracket (>), or vertical bar(`|`).

4.3.1.3.2 Including and Excluding Files

If you use regular expressions, the values you specify in the Include Files and Exclude Files fields must conform to JDK regular expression (regex) constructs. For both fields, different regex patterns must be provided separately. The Include Files and Exclude Files fields correspond to the `IncludeFiles` and `ExcludeFiles` parameters, respectively, of the inbound `WSDL` file.

Note: The regex pattern complies with the JDK regex pattern. According to the JDK regex pattern, the correct connotation for a pattern of any characters with any number of occurrences is a period followed by a plus sign `(.+)`. An asterisk (*) in a JDK regex is not a placeholder for a string of any characters with any number of occurrences.

For the inbound Oracle File Adapter to pick up all file names that start with `po` and which have the extension `txt`, you must specify the Include Files field as `po.*\.txt` when the name pattern is a regular expression. In this regex pattern example:

- A period (`.`) indicates any character.
- An asterisk (`*`) indicates any number of occurrences.
- A backslash followed by a period (`\.`) indicates the character period (.) as indicated with the backslash escape character.

The Exclude Files field is constructed similarly.

If you have Include Files field and Exclude Files field expressions that have an overlap, then the exclude files expression takes precedence. For example, if Include Files is set to `abc*.txt` and Exclude Files is set to `abcd*.txt`, then no `abcd*.txt` files are received.

Note: You *must* enter a name pattern in the **Include Files with Name Pattern** field and not leave it empty. Otherwise, the inbound adapter service reads all the files present in the inbound directory, resulting in incorrect results.

Table 4–3 lists details of Java regex constructs.

Note: Do not begin JDK regex pattern names with the following characters: plus sign (+), question mark (?), or asterisk (*).

Table 4–3 Java Regular Expression Constructs

Matches	Construct
Characters	-
The character `x`	`x`
The backslash character	`\\`
The character with octal value `0n (0 <= n <= 7)`	`\0n`
The character with octal value `0nn (0 <= n <= 7)`	`\0nn`
The character with octal value `0mnn (0 <= m <= 3, 0 <= n <= 7)`	`\0mnn`
The character with hexadecimal value `0xhh`	`\xhh`
The character with hexadecimal value `0xhhhh`	`\uhhhh`
The tab character `('\u0009')`	`\t`
The new line (line feed) character `('\u000A')`	`\n`
The carriage-return character `('\u000D')`	`\r`
The form-feed character `('\u000C')`	`\f`
The alert (bell) character `('\u0007')`	`\a`
The escape character `('\u001B')`	`\e`
The control character corresponding to `x`	`\cx`
-	-
Character classes	-
`a`, `b`, or `c` (simple class)	`[abc]`
Any character except `a`, `b`, or `c` (negation)	`[^abc]`

Table 4–3 (Cont.) Java Regular Expression Constructs

Matches	Construct
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 white space character: `[ \t\n\x0B\f\r]`	`\s`
A nonwhitespace character: `[^\s]`	`\S`
A word character: `[a-zA-Z_0-9]`	`\w`
A nonword character: `[^\w]`	`\W`
Greedy quantifiers	-
X, once or not at all	`X?`
X, zero or more times	`X*`
X, one or more times	`X+`
X, exactly *n* times	`X{n}`
X, at least *n* times	`X{n,}`
X, at least *n*, but not more than *m* times	`X{n,m}`

For details about Java regex constructs, go to

http://java.sun.com/j2se/1.5.0/docs/api

4.3.1.3.3 File Include and Exclude

The FileList operation does not expose the java.file.IncludeFiles property. This property is configured while designing the adapter interaction and cannot be overridden through headers, for example:

```
<adapter-config name="ListFiles" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
<connection-factory location="eis/FileAdapter" UIincludeWildcard="*.txt" adapterRef=""/>
<endpoint-interaction portType="FileListing_ptt" operation="FileListing">
 <interaction-spec className="oracle.tip.adapter.file.outbound.FileListInteractionSpec">
  <property name="PhysicalDirectory" value="%INP_DIR%"/>
  <property name="PhysicalDirectory" value="%INP_DIR%"/>
  <property name="Recursive" value="true"/>
  <property name="Recursive" value="true"/>
  <property name="IncludeFiles" value=".*\.txt"/>
</interaction-spec>
</endpoint-interaction>
</adapter-config>
```

In this example, after you set the IncludeFiles, they cannot be changed.

4.3.1.3.4 Debatching Multiple Inbound Messages

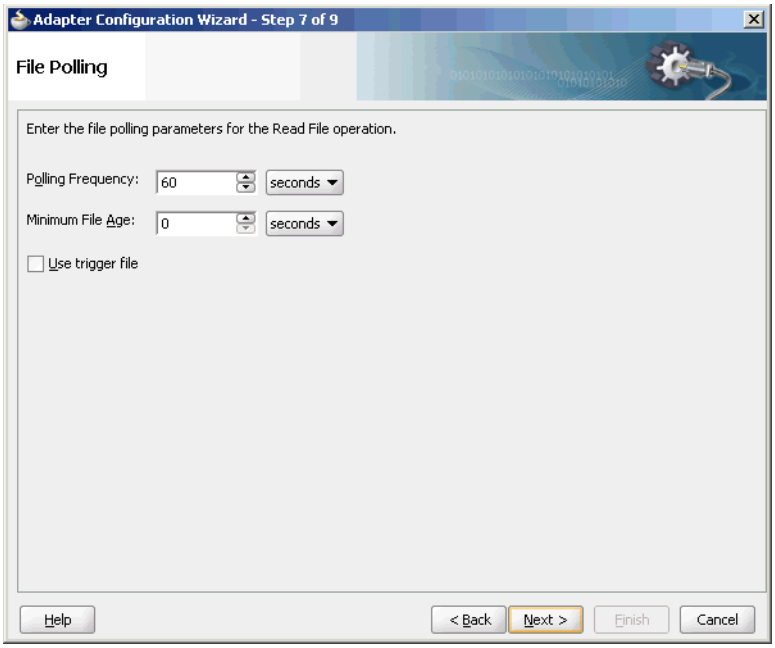
You can select whether incoming files have multiple messages, and specify the number of messages in one batch file to publish. When a file contains multiple messages and this check box is selected, this is referred to as debatching. Nondebatching is applied when the file contains only a single message and the check box is not selected. Debatching is supported for native and XML files.

4.3.1.4 File Polling

The File Polling page of the Adapter Configuration Wizard shown in Figure 4–22 enables you to specify the following inbound polling parameters:

- The frequency with which to poll the inbound directory for new files to retrieve.
- The minimum file age of files to retrieve. For example, this polling parameter enables a large file to be completely copied into the directory before it is retrieved for processing. The age is determined by the last modified time stamp. For example, if you know that it takes three to four minutes for a file to be written, then set the minimum age to five minutes. If a file is detected in the input directory and its modification time is less than five minutes older than the current time, then the file is not retrieved because it is still potentially being written to.

Figure 4–22 The Adapter Configuration Wizard-File Polling Page



Note: You *must not* manually change the value of polling parameters in JCA files. You must use the Adapter Configuration Wizard to modify this parameter.

Using Trigger Files

By default, polling by inbound Oracle File and FTP Adapters start as soon as the endpoint is activated. However, to obtain more control over polling, you can use a

file-based trigger. Once the Oracle File or FTP Adapter finds the specified trigger file in a local or remote directory, it starts polling for the files in the inbound directory.

For example, a BPEL process is writing files to a directory and a second BPEL process is polling the same directory for files. To have the second process start polling the directory only after the first process has written all the files, you can use a trigger file. You can configure the first process to create a trigger file at the end. The second process starts polling the inbound directory after it finds the trigger file.

The trigger file directory can be the same as the inbound polling directory or different from the inbound polling directory. However, if your trigger file directory and the inbound polling directory are the same, then you should ensure that the name of the trigger file is not similar to the file filter specified in the Adapter Configuration page shown in Figure 4–21.

The content of a trigger file is never read and therefore should not be used as payload for an inbound receive activity.

Table 4–4 lists the parameters that you must specify in the inbound service JCA file:

Table 4–4 Trigger File Parameters

Parameter	Description	Example

`TriggerFilePhysicalDirectory`
or
`TriggerFileLogicalDirectory`
	The physical or logical name of the directory in which the Oracle File and FTP Adapters look for the trigger file.
	The `TriggerFilePhysicalDirectory` and `TriggerFileLogicalDirectory` parameters are optional. These parameters must be used only if the trigger file directory is different from the inbound polling directory. By default, the Oracle File and FTP Adapters looks for the trigger file in the inbound polling directory.
		`TriggerFilePhysicalDirectory="C:\foo"`
		
		`TriggerFileLogicalDirectory= "TriggerFileDir"`

`TriggerFile`	The name of the trigger file.	`TriggerFile="Purchase order.trg"`

Table 4–4 (Cont.) Trigger File Parameters

Parameter	Description	Example
`TriggerFileStrategy`	Strategy that is used as the triggering mechanism. The value can be: *EndpointActivation:* The adapter looks for the trigger file every time the composite is activated. **Note:** The composite gets activated every time you start the container or redeploy the application, or retire or activate the composite application from Oracle Enterprise Manager. Every time you restart the container, the composite application is not triggered until it sees the trigger file in the specified directory. *OnceOnly:* The adapter looks for the trigger file only once in its lifetime. After it finds the trigger file, it remember that across restarts and redeployments. *EveryTime:* The adapter looks for the trigger file on each polling cycle.The default value for `TriggerFileStrategy` is EndpointActivation.	`TriggerFileStrategy=" EndpointActivation "`

The following is a sample JCA file for the inbound service:

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FlatStructureIn" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">

<connection-factory location="eis/FileAdapter" UIincludeWildcard="*.txt"
adapterRef=""/>
<endpoint-activation operation="Read">
<activation-spec className="oracle.tip.adapter.file.inbound.FileActivationSpec">
<property.../>
<property name="TriggerFilePhysicalDirectory" value="/tmp/flat/ArchiveDir"/>
</activation-spec>
</endpoint-activation>

</adapter-config>
```

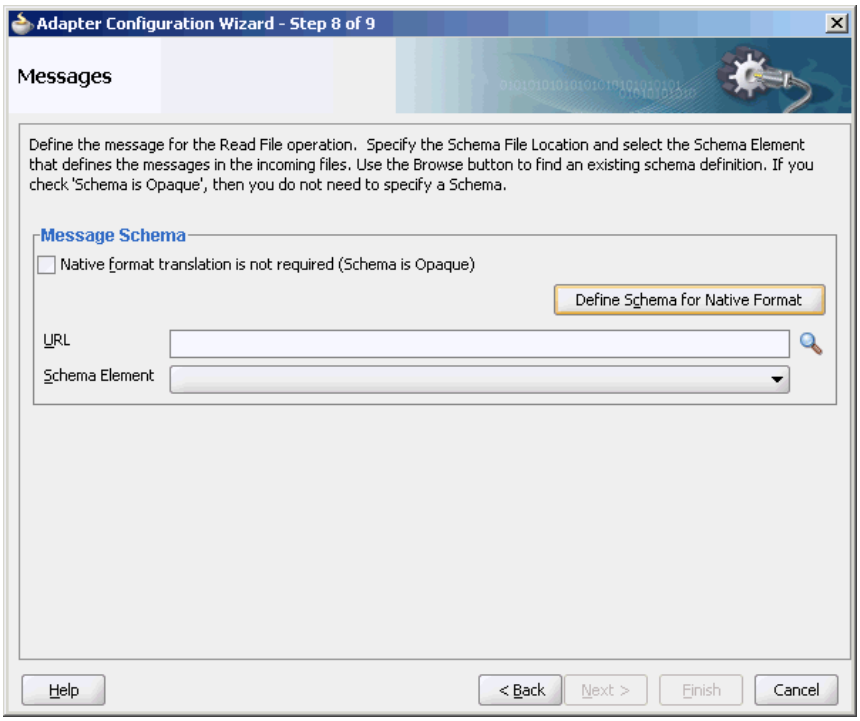
4.3.1.5 Postprocessing

The Oracle File Adapter supports several postprocessing options. After processing the file, files are deleted if specified in the File Polling page shown in Figure 4–22. Files can also be moved to a completion (archive) directory if specified in the File Directories page shown in Figure 4–20.

4.3.1.6 Native Data Translation

The next Adapter Configuration Wizard page that appears is the Messages page shown in Figure 4–23. This page enables you to select the XSD schema file for translation.

Figure 4–23 Specifying the Schema - Messages Page



If native format translation is not required (for example, a JPG or GIF image is being processed), then select the **Native format translation is not required** check box. The file is passed through in base-64 encoding.

XSD files are required for translation. To define a new schema or convert an existing data type definition (DTD) or COBOL Copybook, then select **Define Schema for Native Format**. This starts the Native Format Builder wizard. This wizard guides you through the creation of a native schema file from file formats such as comma-delimited value (CSV), fixed-length, DTD, and COBOL Copybook. After the native schema file is created, the Messages page is displayed, with the Schema File URL and Schema Element fields filled in. For more information, see Section 6.1.1, "Supported File Formats".

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

4.3.1.7 Inbound Service

When you finish configuring the Oracle File Adapter, a `JCA` file is generated for the inbound service. The file is named after the service name you specified on the Service Name page of the Adapter Configuration Wizard. You can rerun the wizard later to change your operation definitions.

The ActivationSpec parameter holds the inbound configuration information. The ActivationSpec and a set of inbound Oracle File Adapter properties are part of the inbound JCA file.

Table 4–5 lists the properties of a sample inbound JCA file.

Table 4–5 Sample JCA Properties for Inbound Service

Property	Sample Value
UseHeaders	true
PhysicalDirectory	/tmp/opaque/in
Recursive	true
DeleteFile	false
IncludeFiles	.*\.xml
PollingFrequency	1
MinimumAge	0

The ActivationSpec property values are specified in the Adapter Configuration Wizard during design time and, as shown in Table 4–5. The inbound Oracle File Adapter uses the following configuration properties:

- PollingFrequency
- MinimumAge
- PhysicalDirectory
- LogicalDirectory
- PublishSize
- PhysicalArchiveDirectory
- LogicalArchiveDirectory
- IncludeFiles
- ExcludeFiles
- UseHeaders
- ListSorter
- ThreadCount
- Recursive
- MaxRaiseSize

For a description of these configuration properties, see Appendix A of this book.

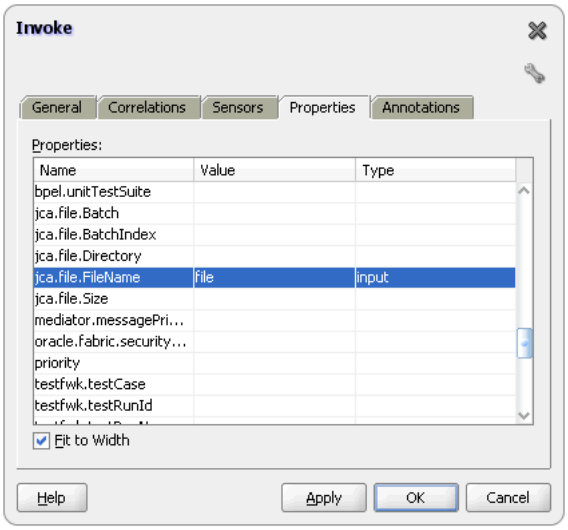
4.3.1.8 Inbound Headers

Apart from the payload, Oracle File Adapter publishes the following header metadata, from the inbound service, as shown in Figure 4–24:

- jca.file.FileName: file name
- jca.file.Directory: directory name
- jca.file.Batch: a unique name for a batch in case of debatching

- `jca.file.BatchIndex`: the batch index for each message within the batch for debatching
- `jca.file.Size`: the file size
- `jca.file.LastModifiedTime`: the last modified time for the file

Figure 4–24 The Invoke Dialog



4.3.2 Oracle File Adapter Write File Concepts

In the outbound direction, the Oracle File Adapter receives messages from the service engine and writes the messages to a file in a file system. This section provides an overview of the outbound file writing capabilities of the Oracle File Adapter. You use the Adapter Configuration Wizard to configure the Oracle File Adapter for use with a BPEL process or a Mediator Service. This creates an outbound `WSDL` and a `JCA` file pair.

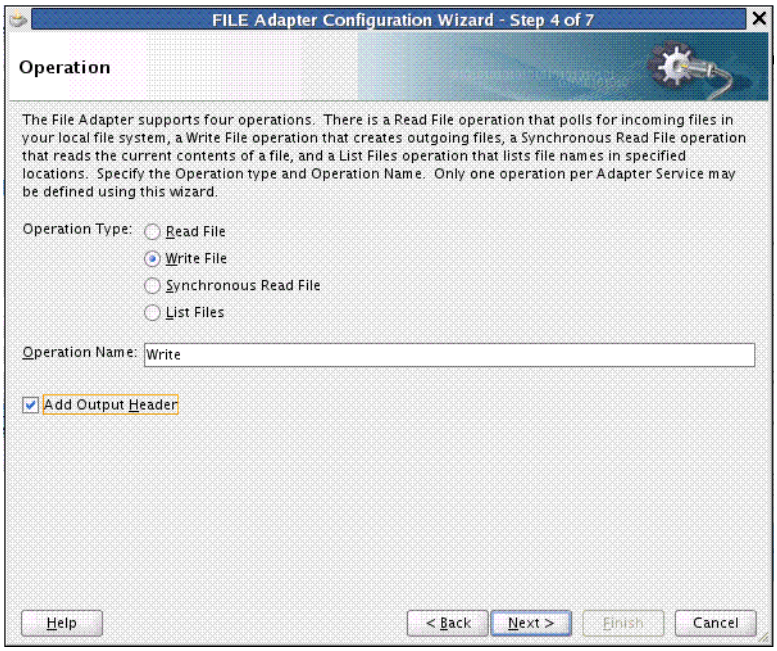
This section includes the following topics:

- Section 4.3.2.1, "Outbound Operation"
- Section 4.3.2.2, "Outbound File Directory Creation"
- Section 4.3.2.3, "Native Data Translation"
- Section 4.3.2.4, "Outbound Service Files"
- Section 4.3.2.5, "Outbound Headers"

4.3.2.1 Outbound Operation

For outbound operations with the Oracle File Adapter, select the **Write File** operation, as shown in Figure 4–25.

Figure 4–25 Selecting the Write File Operation



The Add Output Header check box is visible when you select File Write. When you select this check box, the adapter WSDL has an output message pointing to a header schema, shown by the bold highlight below.

```
 <wsdl:definitions name="fileout3"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/SOAApp1/NewJCAFmwk/
fileout3"
    xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
    xmlns:FILEAPP="http://xmlns.oracle.com/pcbpel/adapter/file/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/file/SOAApp1/NewJCAFmwk/
fileout3"
xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">"
xmlns:opaque="http://xmlns.oracle.com/pcbpel/adapter/opaque/"
<plt:role name="Write_role" >
  <plt:portType name="tns:Write_ptt" />
  </plt:role>
</plt:partnerLinkType>"
<wsdl:types>
<schema TargetNamespace="http://xlmns.oracle.com/pcbpel/adapter/opaque/"
   xmlns:opaque="http://xmlns.oracle.com/pcbpel/adapter/opaque/"
      xmlns="http://www.w3.org/2001/XMLSchema" >
   <element name="opaqueElement" type="base64Binary" />
</schema>
<schema targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/"
xmlns="http://www.w3.org/2001/XMLSchema"
  attributeFormDefault="qualified"
  <element name="OutboundFileHeaderType" >
   <complexType>
    <sequence>
     <element name="filename" type="string" />
     <element name="directory" type="string" />
   </sequence>
   </complexType>
   </element>
</schema>
```

```
</wsdl:types>
   <wsdl:message name="Write_msg">
   <wsdl:part name="opaque" element="opaque:opaqueElement"/>
</wsdl:message>
   <wsdl:message name="Output_msg">
   <wsdl:part name="body" element="FILEAPP:OutboundFileHeaderType"/>
</wsdl:message>
   <wsdl:portType name="Write_ptt">
   <wsdl:operation name="Write">
     <wsdl:input message="tns:Write_msg"/>
     <wsdl:output message="tns:Output_msg"/>
       </wsdl:operation>
     </wsdl:portType>
</wsdl:definitions>
```

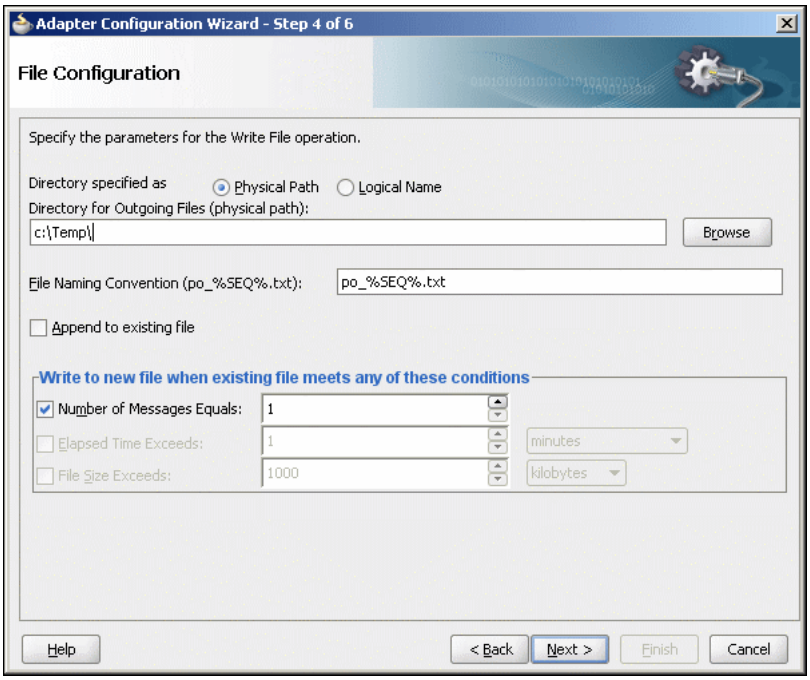
You can select the 'Update Output Header' check box in edit mode, and the output message/ header schema is removed from the adapter WSDL.

4.3.2.2 Outbound File Directory Creation

For the outbound operation, you can specify the outbound directory, outbound file naming convention to use, and, if necessary, the batch file conventions to use.

The File Configuration page of the Adapter Configuration Wizard shown in Figure 4–26 enables you to specify the directory for outgoing files and the outbound file naming convention.

Figure 4–26 The Adapter Configuration Wizard-Parameters for Outgoing Files



The following sections describe the file configuration information to specify:

- Section 4.3.2.2.1, "Specifying Outbound Physical or Logical Directory Paths in Oracle BPEL PM"
- Section 4.3.2.2.4, "Specifying the Outbound File Naming Convention"
- Section 4.3.2.2.5, "Specifying a Dynamic Outbound File Name"

- Section 4.3.2.2.6, "Batching Multiple Outbound Messages"

4.3.2.2.1 Specifying Outbound Physical or Logical Directory Paths in Oracle BPEL PM

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="FlatStructureOut" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">

  <connection-factory location="eis/FileAdapter" adapterRef=""/>
  <endpoint-interaction operation="Write">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.FileInteractionSpec">
      <property name="LogicalDirectory" value="OutputFileDir"/>
      <property name="FileNamingConvention" value="%yyMMddHHmmssSS%_%SEQ%_
%yyyyMMdd%_%SEQ%.out.%SEQ%"/>
      <property name="Append" value="false"/>
      <property name="NumberMessages" value="1"/>
      <property name="OpaqueSchema" value="false"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

Select the outbound adapter in the "External References" swim lane in JDeveloper wizard (it is present in the composite.xml tab). Create a "Binding Property" in the Property Inspector for the outbound adapter (you must scroll down to find it). Once the Create Property box appears, enter `OutputFileDir` in the Name field and the actual output dir name, example, `C:\outputDir` in the Value field. The composite.xml file appears as follows:

```
<reference name="FlatStructureOut">
    <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/file/FlatStructureOut/#wsdl.
interface(Write_ptt)"/>
    <binding.jca config="FlatStructureOut_file.jca">
      <property name="OutputFileDir" type="xs:string" many="false"
                override="may">C:\outputDir</property>
    </binding.jca>
  </reference>
```

> **Note:** Ensure that you limit the length of outbound file names (the file name, plus the complete directory path) to 200 characters. This is not an exact limit but rather a recommendation. When an outbound file name is long (for example, 215 characters), a blank file with that name is created in the outbound directory.

4.3.2.2.2 Specifying Outbound Physical or Logical Directory Paths in Mediator

You can specify outbound directory names as physical or logical paths in Mediator. Physical paths are values such as `c:\inputDir`.

You can specify the logical names at design time in the File Directories page shown in Figure 4–20 and then provide logical-physical mapping by using the Endpoint properties. For example, `WriteFile` is an outbound adapter service. You have specified `OutDir` as the logical directory name during design time.

4.3.2.2.3 Specifying a Dynamic Outbound Directory Name

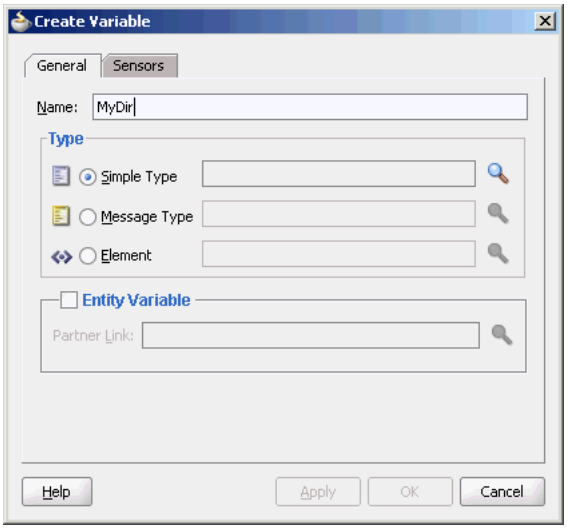
For outbound operation, you can specify a dynamic outbound directory name. You can set variables to specify dynamic outbound directory names.

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="ReadAddressChunk" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter" adapterRef=""/>
  <endpoint-interaction operation="ChunkedRead">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.ChunkedInteractionSpec">
      <property name="PhysicalDirectory" value="C:\foo"/>
      <property name="FileName" value="dummy.txt"/>
      <property name="ChunkSize" value="1"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

In the preceding example, in the `JCA` file, the physical directory is set to `"C:\foo"` but during run time it is dynamically changed to the assigned value. In this example, the physical directory is dynamically changed to `"C:\out"`. You must perform the following steps to specify the dynamic outbound directory name:

1. Double-click the invoke activity.
2. Click the Browse Variables icon.
3. In the Variable Chooser dialog, click the **Create an Object** icon.
4. Create a variable `MyDir` of type `xsd:string`, as shown in Figure 4–27.

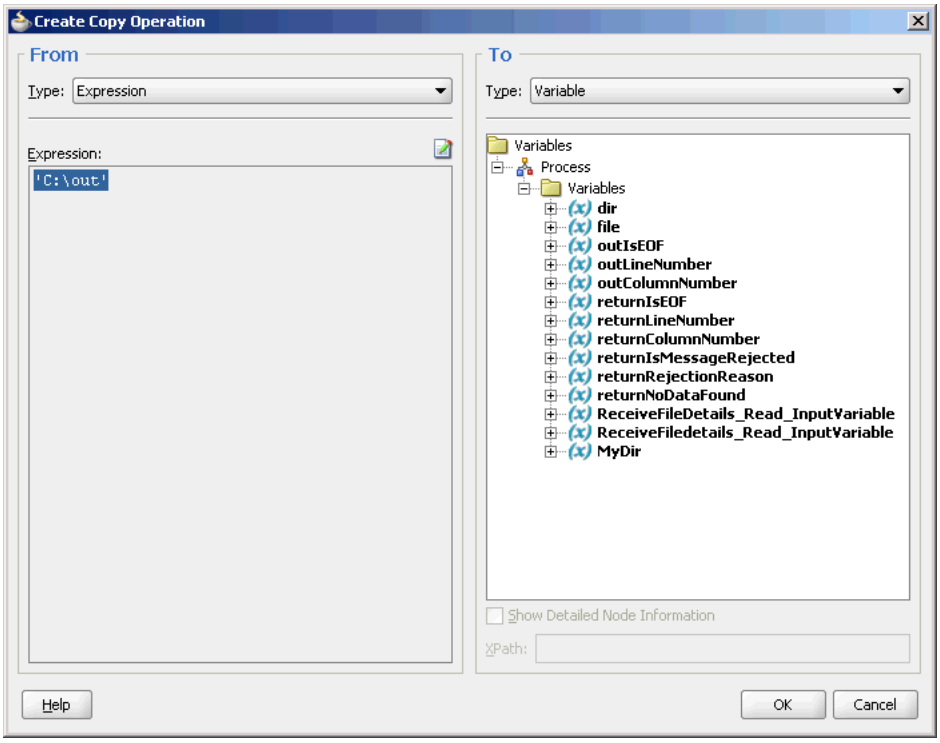
Figure 4–27 Create Variable Dialog



5. Drag and drop an Assign activity from the Component Palette in between the Receive and Invoke activities in the design area.
6. Double-click the assign activity and click the **Copy Operation** tab.

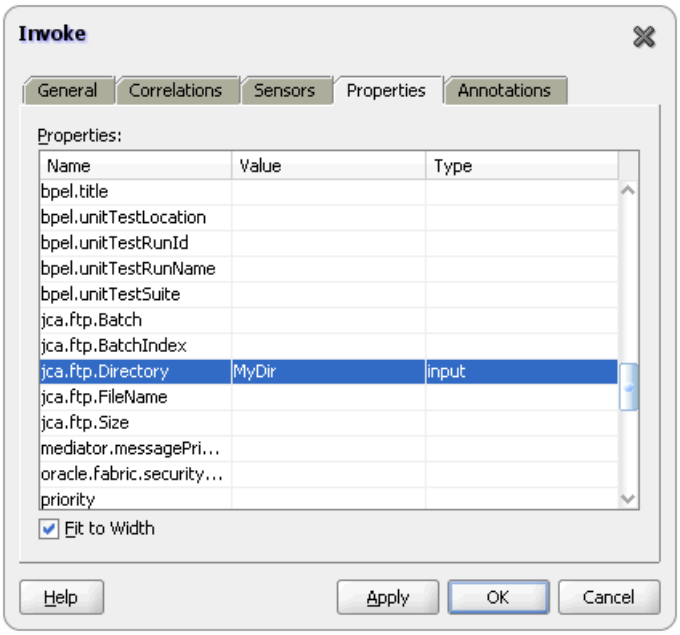
7. Click **Create** and then **Copy Operation**. The Create Copy Operation dialog is displayed.
8. In the Create Copy Operation dialog, select **Expression** from Type and specify the directory name and path, as shown in Figure 4–28. The operation writes the output file to this directory.

Figure 4–28 Create Copy Operation Dialog



9. Click **OK** in the Create Copy Operation dialog and then click **OK** in the Assign dialog. The `.bpel` page is displayed.
10. Double-click the invoke activity. The Invoke dialog is displayed.
11. Click the **Properties** tab.
12. Select the `jca.file.Directory` property from the **Properties** column and set the **Value** as `MyDir` (the directory that you created in Step 4.) Ensure that the **Type** column is set to `input`, as shown in Figure 4–29.

Figure 4–29 The Invoke Dialog



Note: When using dynamic directories, ensure that parameters such as `NumberMessages`, `ElapsedTime`, and `FileSize` are not defined in the outbound adapter service `WSDL` file. These parameters are not supported with dynamic directories.

4.3.2.2.4 Specifying the Outbound File Naming Convention

Specify the naming convention to use for outgoing files. You cannot enter completely static names such as `po.txt`. This is to ensure the uniqueness in names of outgoing files, which prevents files from being inadvertently overwritten. Instead, outgoing file names must be a combination of static and dynamic portions.

The prefix and suffix portions of the file example shown in Figure 4–26 are static (for example, `po_` and `.xml`). The `%SEQ%` variable of the name is dynamic and can be a sequence number or a time stamp (for example, `po_%yyMMddHHmmss%.xml` to create a file with a time stamp).

If you choose a name starting with `po_`, followed by a sequence number and the extension `txt` as the naming convention of the outgoing files, then you must specify `po_%SEQ%.txt`.

If you choose a name starting with `po_`, followed by a time stamp with the pattern `yyyy.MM.dd` and the extension `txt` as the naming convention of the outgoing file, then you must specify `po_%yyyy.MM.dd%.txt`. For example, the outgoing file name can be `po_2004.11.29.txt`.

Additionally, you can combine file naming conventions. For example, you can specify the file naming convention as `po_%SEQ%_%yyyy.MM.dd%_%SEQ%.txt`.

Note: When you use the time stamp pattern, the same time stamp may be generated on subsequent calls and you may lose messages. The workaround is to combine the time-stamp pattern with a sequence pattern. Alternatively, you can use a time-stamp pattern closest to a millisecond, in which case the adapter handles the uniqueness of the file names.

You cannot use a regular expression for outbound synchronous reads. In these cases, the exact file name must be known.

A time stamp is specified by date and time pattern strings. Within date and time pattern strings, unquoted letters from `'A'` to `'Z'` and from `'a'` to `'z'` are interpreted as pattern letters representing the components of a date or time string. Text can be quoted using single quotation marks (`'`) to avoid interpretation. The characters `"''"` represent single quotation marks. All other characters are not interpreted.

The Java pattern letters are defined in Table 4–6.

Table 4–6 Java Pattern Letters

Letter	Date or Time Component	Presentation	Examples
`G`	Era designator	Text	`AD`
`y`	Year	Year	`1996; 96`
`M`	Month in year	Month	`July; Jul; 07`
`w`	Week in year	Number	`27`
`W`	Week in month	Number	`2`
`D`	Day in year	Number	`189`
`d`	Day in month	Number	`10`
`F`	Day of week in month	Number	`2`
`E`	Day in week	Text	`Tuesday; Tue`
`a`	AM/PM marker	Text	`PM`
`H`	Hour in day (0-23)	Number	`0`
`k`	Hour in day (1-24)	Number	`24`
`K`	Hour in AM/PM (0-11)	Number	`0`
`h`	Hour in AM/PM (1-12)	Number	`12`
`m`	Minute in hour	Number	`30`
`s`	Second in minute	Number	`55`
`S`	Millisecond	Number	`978`
`z`	Time zone	General Time Zone	`Pacific Standard Time; PST; GMT-08:00`
`Z`	Time zone	RFC 822 Time Zone	`-0800`

Different presentations in the pattern are as follows:

- Text

  For formatting, if the number of pattern letters is four or more, then the full form is used; otherwise, a short or abbreviated form is used if available. For parsing, both forms are accepted, independent of the number of pattern letters.

- Number

  For formatting, the number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this number. For parsing, the number of pattern letters is ignored unless it is needed to separate two adjacent fields.

- Year

  For formatting, if the number of pattern letters is two, then the year is truncated to two digits; otherwise, it is interpreted as a number.

For parsing, if the number of pattern letters is more than two, then the year is interpreted literally, regardless of the number of digits. Using the pattern `MM/dd/yyyy`, `01/11/12` parses to `Jan 11, 12 A.D`.

For parsing with the abbreviated year pattern (`y` or `yy`), the abbreviated year is interpreted relative to some century. The date is adjusted to be within 80 years before and 20 years after the time instance is created. For example, using a pattern of `MM/dd/yy` and `Jan 1, 1997` is created; the string `01/11/12` is interpreted as `Jan 11, 2012`, while the string `05/04/64` is interpreted as `May 4, 1964`. During parsing, only strings consisting of exactly two digits are parsed into the default century. Any other numeric string, such as a one-digit string, a three-or-more-digit string, or a two-digit string that is not all digits (for example, `-1`), is interpreted literally. So, `01/02/3` or `01/02/003` is parsed using the same pattern as `Jan 2, 3 AD`. Likewise, `01/02/-3` is parsed as `Jan 2, 4 BC`.

- Month

  If the number of pattern letters is `3` or more, then the month is interpreted as text; otherwise, it is interpreted as a number.

- General time zone

  Time zones are interpreted as text if they have names. For time zones representing a `GMT` offset value, the following syntax is used:

```
GMTOffsetTimeZone:
      GMT Sign Hours : Minutes
   Sign: one of
      + -
   Hours:
      Digit
      Digit Digit
   Minutes:
      Digit Digit
   Digit: one of
      0 1 2 3 4 5 6 7 8 9
```

`Hours` must be between `0` and `23`, and `Minutes` must be between `00` and `59`. The format is locale-independent and digits must be taken from the Basic Latin block of the Unicode standard.

For parsing, RFC 822 time zones are also accepted.

For formatting, the RFC 822 4-digit time zone format is used:

```
RFC822TimeZone:
      Sign TwoDigitHours Minutes
```

```
   TwoDigitHours:
      Digit Digit
```

`TwoDigitHours` must be between `00` and `23`. Other definitions are the same as for general time zones.

For parsing, general time zones are also accepted.

4.3.2.2.5 Specifying a Dynamic Outbound File Name

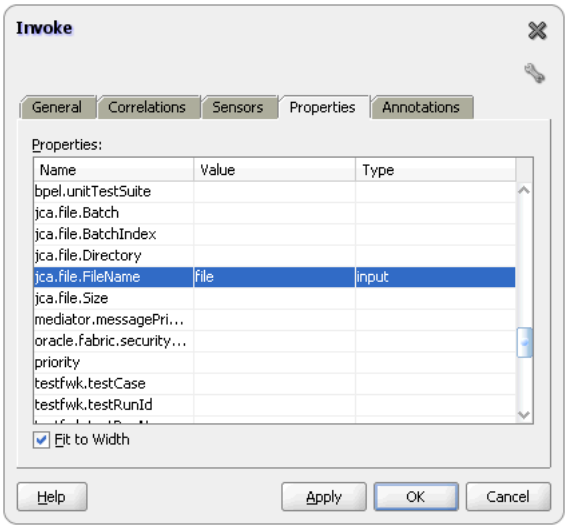
For outbound operation, you can specify a dynamic outbound file name. You can set variables to specify dynamic outbound file names.

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="ReadAddressChunk" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter" adapterRef=""/>
  <endpoint-interaction operation="ChunkedRead">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.ChunkedInteractionSpec">
      <property name="PhysicalDirectory" value="C:\foo"/>
      <property name="FileName" value="dummy.txt"/>
      <property name="ChunkSize" value="1"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

In the preceding example, in the JCA file, the physical directory is set to `"C:\foo"` but during run time it is dynamically changed to the assigned value. In this example, the physical directory is dynamically changed to `"C:\out"`. You must perform the following steps to specify the dynamic outbound directory name:

1. Double-click the invoke activity.
2. Click the Browse Variables icon.
3. In the Variable Chooser dialog, click the **Create an Object** icon.
4. Create a variable `file` of type `xsd:string`, as shown in Figure 4–27.
5. Drag and drop an Assign activity from the Component Palette in between the Receive and Invoke activities in the design area.
6. Double-click the assign activity and click the **Copy Operation** tab.
7. Click **Create** and then **Copy Operation**. The Create Copy Operation dialog is displayed.
8. In the Create Copy Operation dialog, select **Expression** from Type and specify the file name, as shown in Figure 4–28. The file operation writes the output file to this file.
9. Click **OK** till you exit the assign activity dialog.
10. Double-click the invoke activity. The Invoke dialog is displayed.
11. Click the **Properties** tab.
12. Select the `jca.file.FileName` property from the **Properties** column and set the **Value** as `file` (the file that you created in Step 4.) Ensure that the Type column is set to `input`, as shown in Figure 4–30.

Figure 4–30 The Invoke Dialog



Note: When using dynamic files, ensure that parameters such as `NumberMessages`, `ElapsedTime`, and `FileSize` are not defined in the outbound adapter service `WSDL` file. These parameters are not supported with dynamic files.

4.3.2.2.6 Batching Multiple Outbound Messages

In the simplest scenario, you specify writing a single file to a single message. You can also specify the outbound method for batch file writing. This method enables you to specify the number of messages to publish in one batch file. The following batch file settings are provided in the File Configuration page shown in Figure 4–26:

- Number of Messages Equals

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

- Elapsed Time Exceeds

  Specify a time which, when exceeded, causes a new outgoing file to be created.

  Note: The Elapsed Time Exceeds batching criteria is evaluated and a new outgoing file is created, only when an invocation happens.

  For example, if you specify that elapsed time exceeds 15 seconds, then the first message that is received is not written out, even after 15 seconds, as batching conditions are not valid. If a second message is received, then batching conditions become valid for the first one, and an output file is created when the elapsed time exceeds 15 seconds.

- File Size Exceeds

  Specify a file size which, when equaled, causes an outgoing file to be created. For example, assume that you specify a value of `3` for the number of messages received and a value of 1 MB for the file size. When you receive two messages that when combined equal or exceed 1 MB, or three messages that are less than 1 MB, an output file is created.

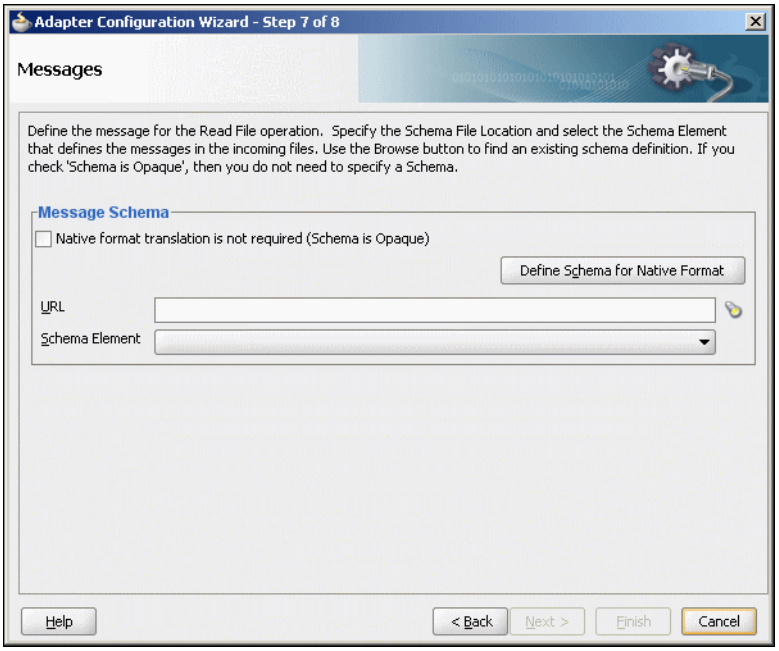
Note: You *must not* manually change the file configurations specified in the preceding list in the JCA files. You must use the Adapter Configuration Wizard to modify these configurations.

If the Oracle File Adapter encounters some problem during batching, then it starts batching at the point at which it left off on recovery.

4.3.2.3 Native Data Translation

The next Adapter Configuration Wizard page that appears is the Messages page shown in Figure 4–31. This page enables you to select the XSD schema file for translation.

Figure 4–31 Specifying the Schema



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

- Specify whether native format translation is not required.
- Select the XSD schema file for translation.
- Start the Native Format Builder wizard to create an XSD file from file formats such as CSV, fixed-length, DTD, and COBOL Copybook.

For more information about Messages page, see Section 4.3.1.6, "Native Data Translation."

4.3.2.4 Outbound Service Files

When you complete configuration of the Oracle File Adapter with the Adapter Configuration Wizard, a WSDL and a JCA file pair is generated for the outbound operation. The files are named after the service name you specified on the Service Name page of the Adapter Configuration Wizard shown in Figure 2–8, "Application

Profile Deployment". You can rerun the wizard later to change your operation definitions.

A sample outbound JCA file includes the information listed in Table 4–7:

Table 4–7 Sample JCA Properties for Outbound Service

Property	Sample Value
PhysicalDirectory	/tmp/flat/OutputDir
FileNamingConvention	address-csv%SEQ%.txt
Append	true
NumberMessages	1
ConcurrentThreshold	0
OpaqueSchema	false

The outbound Oracle File Adapter uses the following configuration parameters:

- PhysicalDirectory
- LogicalDirectory
- NumberMessages
- ElapsedTime
- FileSize
- FileNamingConvention
- Append

For a description of these configuration properties, see Appendix A of this book.

4.3.2.5 Outbound Headers

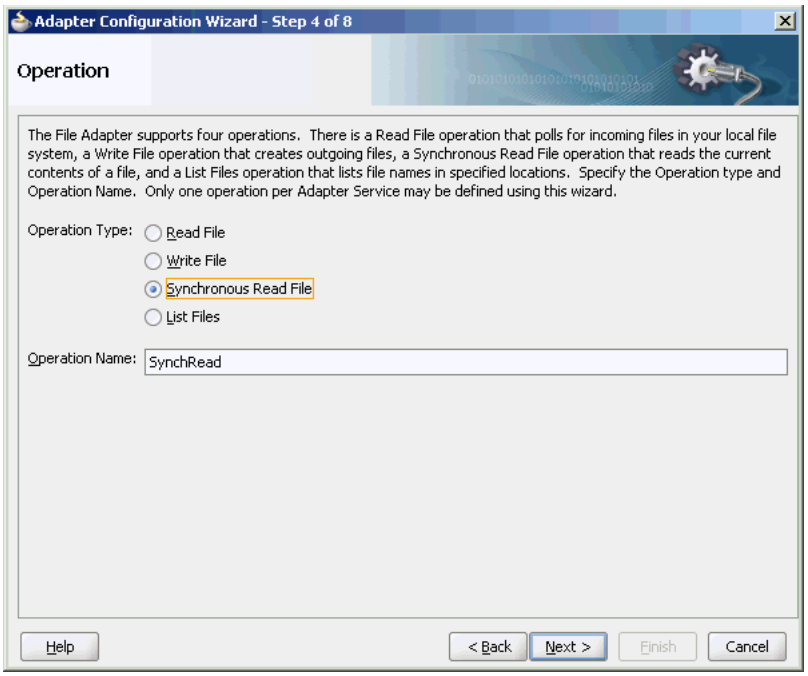
Apart from the payload, the Oracle File Adapter receives the following headers from the component:

- jca.file.FileName: file name
- jca.file.Directory: directory name

4.3.3 Oracle File Adapter Synchronous Read Concepts

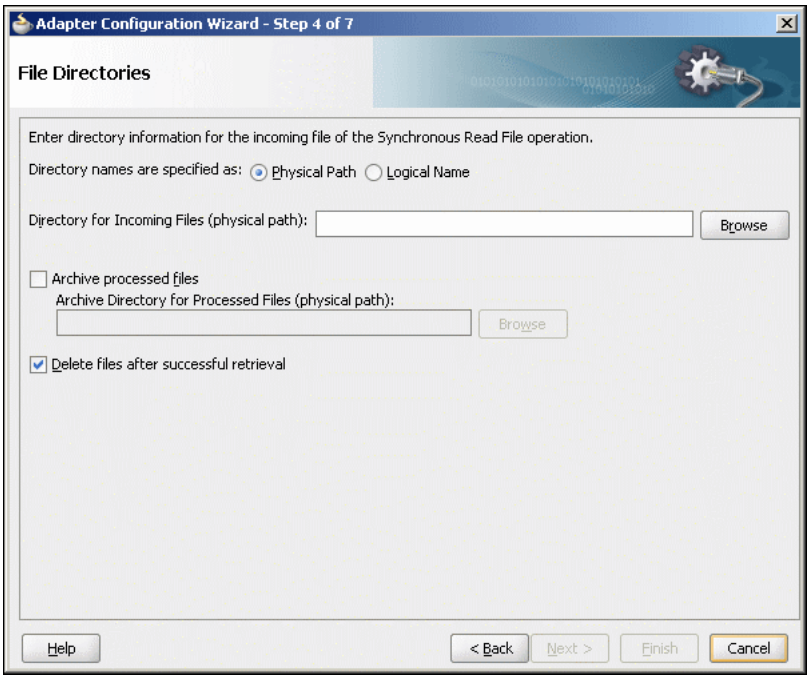
In the outbound direction, the Oracle File Adapter polls and reads the current contents of files. This section provides an overview of the outbound synchronous file reading capabilities of the Oracle File Adapter. For reading a file synchronously, you select Synchronous Read File operation, as shown in Figure 4–32.

Figure 4–32 Synchronous Read Operation Page



All the pages of the Adapter Configuration Wizard are similar to the Read File operation except the File Name page. You can specify the name of the file to be read in the **File Name** field, as shown in Figure 4–33.

Figure 4–33 File Directories Page



4.3.4 Oracle File Adapter File Listing Concepts

This feature of the Oracle File Adapter lets you use a BPEL activity to retrieve a list of files from a target directory. This list of files is returned as an XML document and

contains information such as file name, directory name, file size, and last modified time. This section provides an overview of the file listing capabilities of the Oracle File Adapter. You use the Adapter Configuration Wizard to configure the Oracle File Adapter for use with a BPEL process or a Mediator service. This creates an outbound WSDL and JCA file pair.

Note: The file creation time property, `creationTime`, is not supported because the standard Java APIs do not provide a mechanism to retrieve the creation time. The value of the `creationTime` property is always displayed as `0`.

For example,

```
<creationTime
xmlns="http://xmlns.oracle.com/pcbpel/adapter/file/FAListFiles/FALi
stFilesTest/ReadS/">0</creationTime>
```

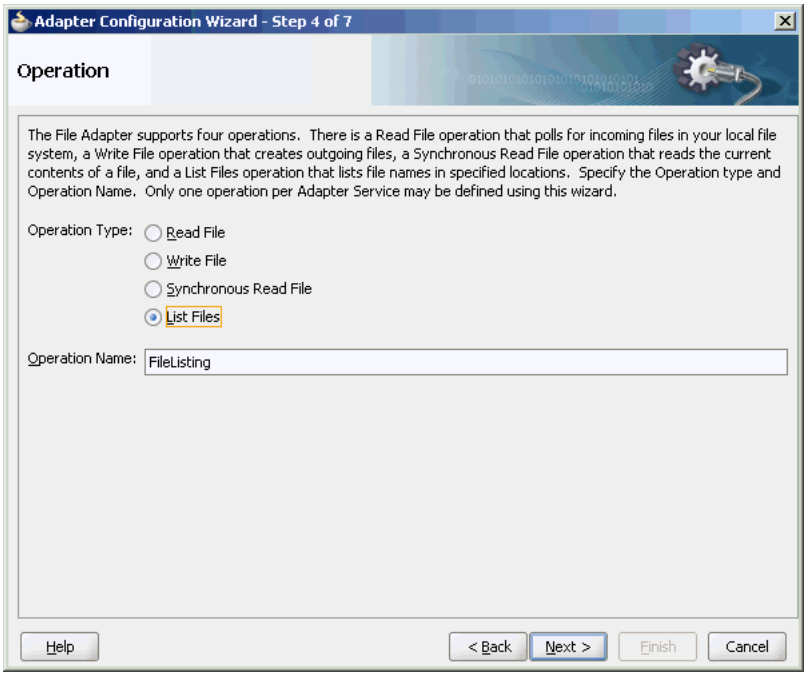
This section includes the following topics:

- Section 4.3.4.1, "Listing Operation"
- Section 4.3.4.2, "File Directory Specifications"
- Section 4.3.4.3, "File Matching"

4.3.4.1 Listing Operation

For listing files, you must select the List Files operation, as shown in Figure 4–34.

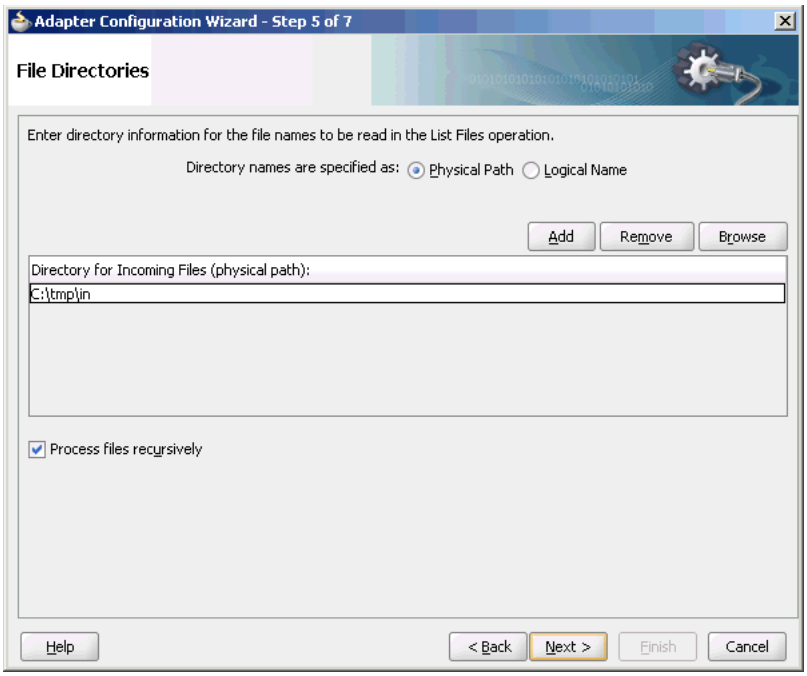
Figure 4–34 List Files Operation Page



4.3.4.2 File Directory Specifications

The File Directories page of the Adapter Configuration Wizard shown in Figure 4–35 enables you to specify information about the directory to use for reading files names for the list operation. You can choose to list files recursively within directories.

Figure 4–35 The Adapter Configuration Wizard-Specifying Incoming Files



The following section describes the file directory information to specify:

4.3.4.2.1 Specifying Inbound Physical or Logical Directory Paths in SOA Composite

You can specify directory names as physical or logical paths for composites involving Oracle BPEL PM and Mediator. Physical paths are values such as `C:\inputDir`.

In the composite, logical properties are specified in the `JCA` file, and their logical-physical mapping is resolved by using binding properties. You specify the logical directory once at design time, and you can later modify the directory name as needed.

For example, the generated `JCA` file looks as follows for the logical input directory name `C:\inputDir`:

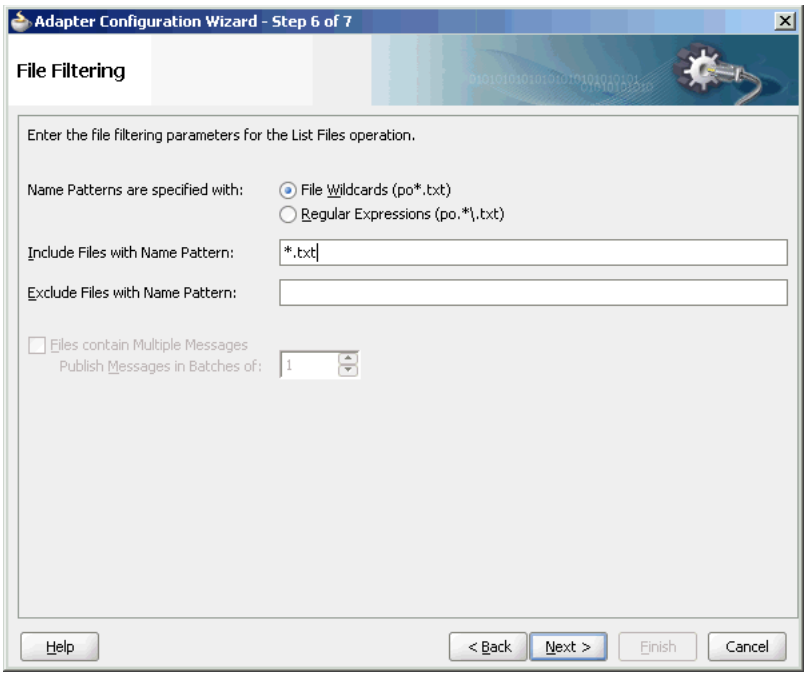
```
<adapter-config name="ListFiles" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">

  <connection-factory location="eis/FileAdapter" UIincludeWildcard="*.txt"
adapterRef=""/>
  <endpoint-interaction portType="FileListing_ptt" operation="FileListing">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.FileListInteractionSpec">
      <property name="PhysicalDirectory" value="C:\inputDir"/>
      <property name="Recursive" value="true"/>
      <property name="IncludeFiles" value=".*\.txt"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

4.3.4.3 File Matching

The File Filtering page of the Adapter Configuration Wizard shown in Figure 4–36 enables you to specify details about the files to retrieve or ignore.

The Oracle File Adapter acts as a file listener and polls the specified directory on a local or remote file system and looks for files that match specified naming criteria.

Figure 4–36 The Adapter Configuration Wizard - File Filtering



The following sections describe the file filtering information to specify:

- Section 4.3.4.3.1, "Specifying a Naming Pattern"
- Section 4.3.4.3.2, "Including and Excluding Files"

4.3.4.3.1 Specifying a Naming Pattern

Specify the naming convention that the Oracle File Adapter uses to poll for inbound files. You can also specify the naming convention for files you do not want to process. Two naming conventions are available for selection. The Oracle File Adapter matches the files that appear in the inbound directory.

- File wildcards (`po*.txt`)

  Retrieve all files that start with `po` and end with `.txt`. This convention conforms to operating system standards.

- Regular expressions (`po.*\.txt`)

  Retrieve all files that start with `po` and end with `.txt`. This convention conforms to Java Development Kit (JDK) regular expression (regex) constructs.

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 file name, plus the complete directory path). Some operating systems also have restrictions on the use of specific characters in file names. For example, Windows operating systems do not allow characters such as `backslash(\)`, slash (`/`), colon (`:`), asterisk (`*`), left angle bracket (<), right angle bracket (>), or vertical bar(`|`).

4.3.4.3.2 Including and Excluding Files

If you use regular expressions, the values you specify in the Include Files and Exclude Files fields must conform to JDK regular expression (regex) constructs. For both fields, different regex patterns must be provided separately. The Include Files and Exclude Files fields correspond to the `IncludeFiles` and `ExcludeFiles` parameters, respectively, of the inbound `WSDL` file.

Note: The regex pattern complies with the JDK regex pattern. According to the JDK regex pattern, the correct connotation for a pattern of any characters with any number of occurrences is a period followed by a plus sign `(.+)`. An asterisk (*) in a JDK regex is not a placeholder for a string of any characters with any number of occurrences.

To have the inbound Oracle File Adapter to pick up all file names that start with `po` and which have the extension `txt`, you must specify the Include Files field as `po.*\.txt` when the name pattern is a regular expression. In this regex pattern example:

- A period (`.`) indicates any character.
- An asterisk (`*`) indicates any number of occurrences.
- A backslash followed by a period (`\.`) indicates the character period (.) as indicated with the backslash escape character.

The Exclude Files field is constructed similarly.

If you have Include Files field and Exclude Files field expressions that have an overlap, then the exclude files expression takes precedence. For example, if Include Files is set to `abc*.txt` and Exclude Files is set to `abcd*.txt`, then you receive any files prefixed with `abcd*`.

Note: Do not begin JDK regex pattern names with the following characters: plus sign (+), question mark (`?`), or asterisk (`*`).

For details about Java regex constructs, go to

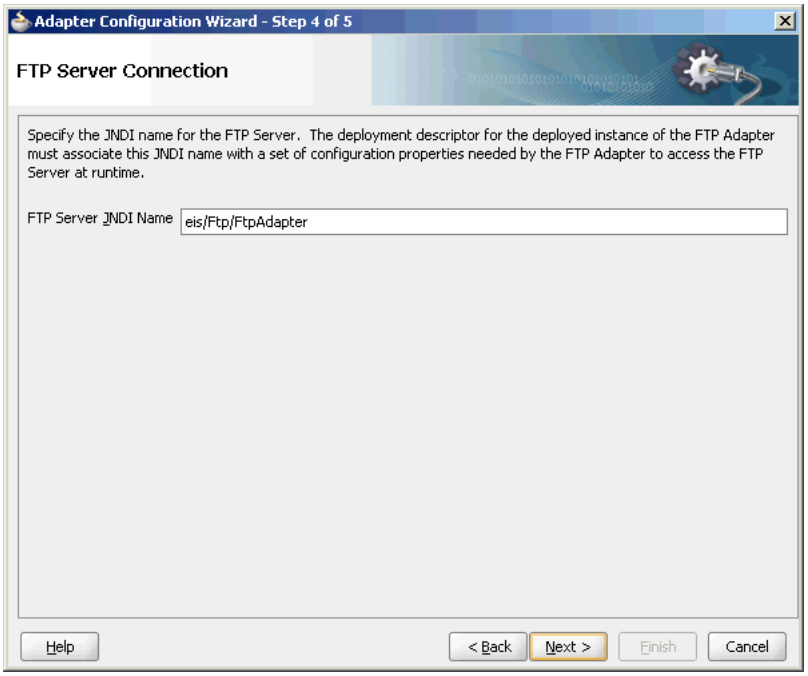
http://java.sun.com/j2se/1.5.0/docs/api

Note: Files are not read and therefore there is no native data translation.

4.3.5 Oracle FTP Adapter Get File Concepts

In the inbound direction, the Oracle FTP Adapter works the same way as the Read File operations of the Oracle File Adapter in that it polls and gets files from a file system for processing. The major difference is that the Oracle FTP Adapter is used for remote file exchanges. To configure the FTP adapter for remote file exchanges, the Adapter Configuration Wizard asks for connection information to an FTP server to be used later, as shown in Figure 4–37.

Figure 4–37 Specifying FTP Server Connection Information

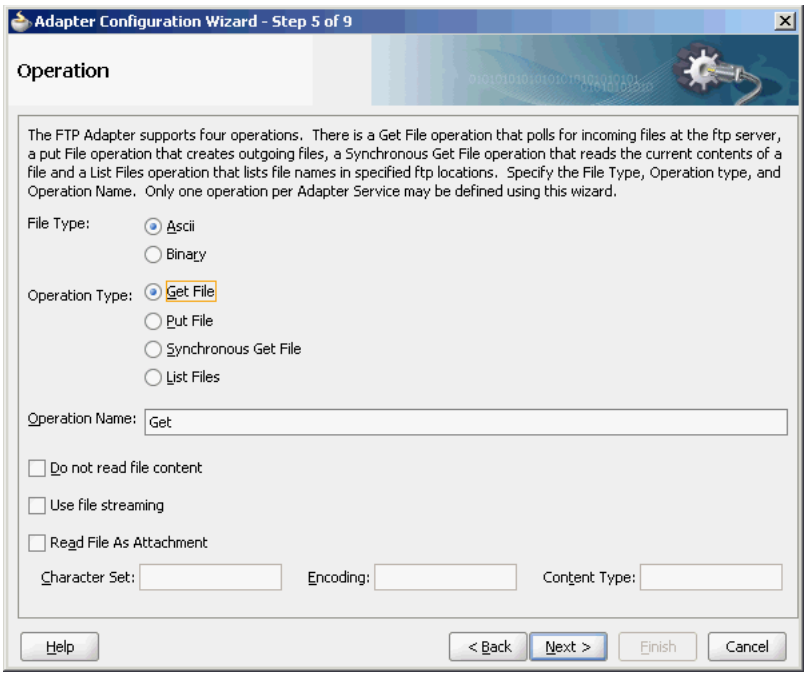


The default adapter instance JNDI name is **`eis/Ftp/FtpAdapter`**, or use a custom name. This name connects to the FTP server during run time.

Note: The Oracle FTP Adapter does not support the FTP commands `RESTART` and `RECOVERY` during the transfer of large files.

After logging in, you select the Get File (read) operation and the type of file to deliver. Figure 4–38 shows this selection.

Figure 4–38 Selecting the Get File Operation



The `serverType` property in the deployment descriptor is used to determine line separators when you transfer data. You can specify `unix`, `win`, or `mac` as property values. These values represent the operating system on which the FTP server is running. By default, the serverType property contains `unix`.

When you specify `mac` as the value, `\r` is used as line separator. For `unix`, `\n` is used and for `win`, `\r\n` is used. You must note that this property is used by the NXSD translator component to write the line separator during an outbound operation.

From this point onwards, pages of the Adapter Configuration Wizard for the Get File operation are the same as those for the Read File operation of the file. Table 4–8 lists the pages that are displayed and provides references to sections that describe their functionality.

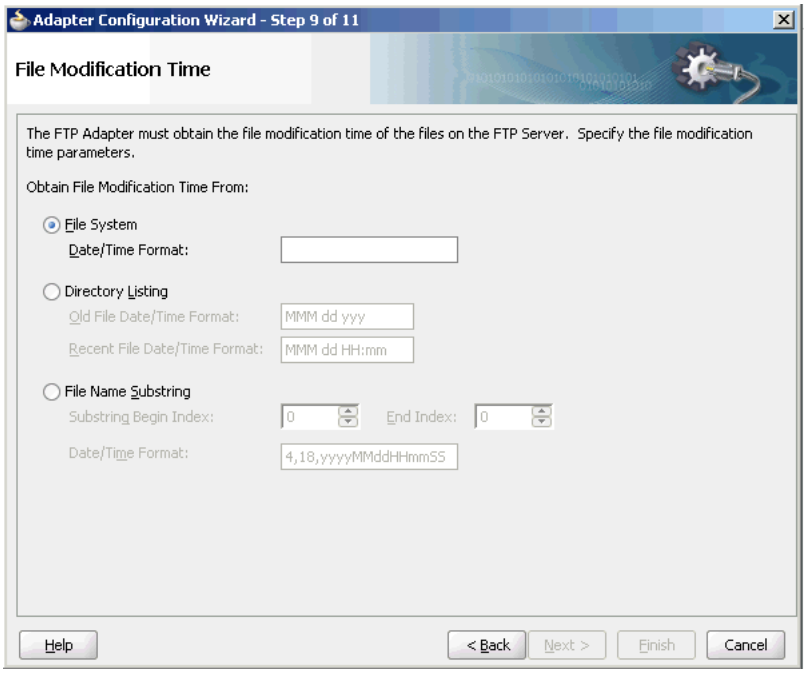
Table 4–8 Adapter Configuration Wizard Windows for Get File Operation

Page	See Section...
File Directories (Figure 4–20)	Section 4.3.1.2, "Inbound File Directory Specifications"
File Filtering (Figure 4–21)	Section 4.3.1.3, "File Matching and Batch Processing"
File Polling (Figure 4–22)	Section 4.3.1.4, "File Polling"
Messages (Figure 4–23)	Section 4.3.1.6, "Native Data Translation"

An additional Adapter Configuration Wizard page is also available for advanced users. This page is shown in Figure 4–39 and appears only after you make either or both of the following selections on the File Polling page shown in Figure 4–22:

- Do not select the **Delete Files After Successful Retrieval** check box.
- Set the value of the **Minimum File Age** field to a value greater than **0**.

Figure 4–39 File Modification Time



This page enables you to specify a method for obtaining the modification time of files on the remote FTP server:

Note: The Oracle FTP Adapter uses the `LIST` command as opposed to `NLST` for listing and retrieves the time stamps because of which you must not specify the time formats. However, you must specify the time formats as shown below if you do any of the following:

- If you specify `NLST` as the listing command (either through the mapping file or the `UseNlst="true"` parameter in the inbound`JCA` file)
- To use the File Name Substring option

This note is not applicable if your case does not fall under neither of these categories.

- **File System**

  This option enables you to obtain the date/time format of the file modification time with the file system listing command. However, this option is rarely used and is not supported by all FTP servers. See your FTP server documentation to determine whether your server supports the file system listing command, which command-line syntax to use, and how to interpret the output.

  For example, if the file system listing command `quote mdtm` *`filename`* is supported and returns the following information:

  ```
  213 20050602102633
  ```

  specify the start index, end index, and date/time format of the file modification time in the **Data/Time Format** field as a single value separated by commas (for example, **4,18,yyyyMMddHHmmss**).

Where:

- 4 is the start index of the file modification time.
- **18** is the end index of the file modification time.
- yyyyMMddHHmmss is the data/time format of the file modification time obtained with the `quote mdtm` *`filename`* command.

The resulting JCA file includes the following parameters and values:

```
<property name=" FileModificationTime " value=" FileSystem "/>
<property name=" ModificationTimeFormat" value=" 4,18,yyyyMMddHHmmss "/>
```

To handle the time zone issue, you must also be aware of the time stamp difference. The time zone of the FTP server is determined by using the Windows date/time properties (for example, by double-clicking the time being displayed in the Windows task bar). You must then convert the time difference between the FTP server and the system on which the Oracle FTP Adapter is running to milliseconds and add the value as a binding property in the `composite.xml` file:

```
<binding.jca config="FlatStructureIn_file.jca">
   <property name="timestampOffset" source="" type="xs:string" many="false"
override="may">238488888</property-->
</binding.jca>
```

- **Directory Listing**

  This option enables you to obtain the date/time format from the file modification time with the FTP directory listing command. For example, if the directory listing command (`ls -l`) returns the following information:

  ```
  12-27-04  07:44AM                2829 NativeData2.txt
  ```

  specify the start index, end index, and date/time format of the file modification time as a single value separated by commas in either the Old File Date/Time Format field or the Recent File Date/Time Format field (for example, `0,17, MM-dd-yy hh:mma`).

  Where:

  - `0` is the start index of the file modification time.
  - `17` is the end index of the file modification time.
  - MM-dd-yy hh:mma is the date/time format of the file modification time obtained with the `ls -l` command. For this example, the value is entered in the Recent File Date/Time Format field. This field indicates that the format is obtained from the most recent file adhering to the naming convention, whereas the Old File Date/Time Format field obtains the format from the oldest file.

  The resulting JCA file includes the following parameters and values:

  ```
  <property name=" FileModificationTime " value=" DirListing"/>
  <property name=" ModificationTimeFormat" value="0,17, MM-dd-yy hh:mma "/>
  ```

  To handle the time zone issue, you must also be aware of the time stamp difference. The time zone of the FTP server is determined by using the Windows date/time properties (for example, by double-clicking the time being displayed in the Windows task bar). You must then convert the time difference between the FTP server and the system on which the Oracle FTP Adapter is running to milliseconds and add the value as a binding property in the `composite.xml` file:

```
<binding.jca config="FlatStructureIn_file.jca">
   <property name="timestampOffset" source="" type="xs:string" many="false"
override="may">238488888</property-->
</binding.jca>
```

- **File Name Substring**

  This option enables you to obtain the modification time from the file name. For example, if the name of the file is `fixedLength_20050324.txt`, you can specify the following values:

  - The start index in the 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 JCA file includes the following parameters and values:

```
<property name=" FileModificationTime " value=" Filename"/>
<property name=" FileNameSubstringBegin " value="12 "/>
<property name=" FileNameSubstringEnd " value="20"/>
<property name=" ModificationTimeFormat " value=" yyyyMMdd "/>
```

After the completion of the Adapter Configuration Wizard, configuration files are created in the Applications section of JDeveloper.

See Figure 2–21, "Custom Adapter Configuration Wizard JCA Properties Screen" for more information about error handling.

You must also add the `DefaultDateFormat` and the `RecentDateFormat` parameters to the deployment descriptor for Oracle FTP Adapter, as shown in the following sample:

```
<non-managed-connection
managedConnectionFactoryClassName="oracle.tip.adapter.ftp.FTPManagedConnection
 Factory">
   <property name="host" value="localhost"/>
   <property name="port" value="21"/>
   <property name="username" value="****"/>
   <property name="password" value="****"/>
    <property name="listParserKey" value="UNIX"/>
    <property name="defaultDateFormat" value="MMM d yyyy"/>
    <property name="recentDateFormat" value="MMM d HH:mm"/>
</non-managed-connection>
```

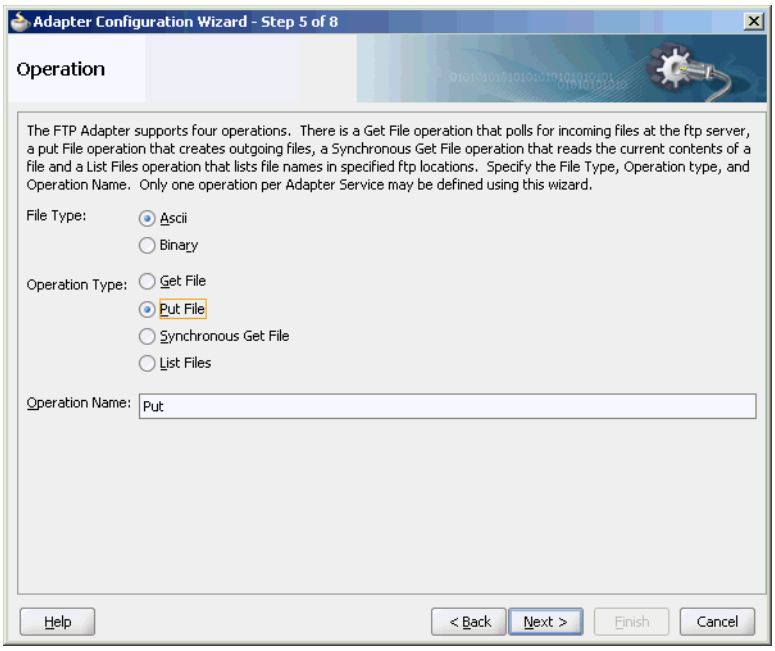
For more information on the `DefaultDateFormat` and the `RecentDateFormat` parameters, refer to Section 4.2.21, "Recursive Processing of Files Within Directories in Oracle FTP Adapter."

4.3.6 Oracle FTP Adapter Put File Concepts

In the outbound direction, the Oracle FTP Adapter works the same as the Write File operations of the Oracle File Adapter. The Oracle FTP Adapter receives messages from a BPEL process or a Mediator service and writes the messages in a file to a file system (in this case, remote). Because the messages must be written to a remote system, the Adapter Configuration Wizard prompts you to connect to the FTP server with the adapter instance JNDI name, as shown in Figure 4–37.

After logging in, you select the Put File (write) operation and the type of file to deliver. Figure 4–40 shows this selection.

Figure 4–40 Selecting the Put File Operation



From this point onwards, pages of the Adapter Configuration Wizard for the Put File operation are the same as those for the Write File operation of the Oracle File Adapter. Table 4–9 lists the pages that display and provide references to sections that describe their functionality.

Table 4–9 The Adapter Configuration Wizard Pages for Put File Operation

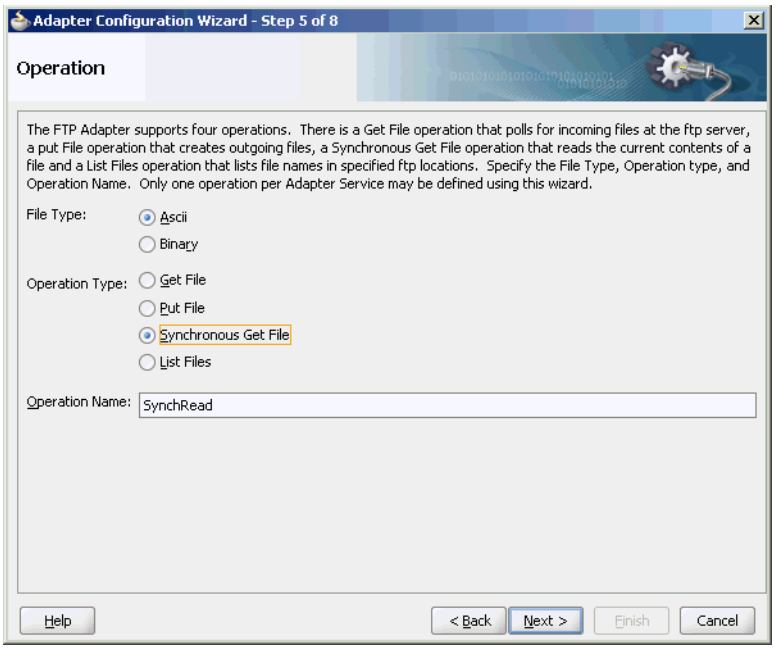
Page	See Section...
File Configuration (Figure 4–26)	Section 4.3.2.2, "Outbound File Directory Creation"
Messages (Figure 4–31)	Section 4.3.2.3, "Native Data Translation"

After the completion of the Adapter Configuration Wizard, configuration files are created in the Applications section of JDeveloper.

4.3.7 Oracle FTP Adapter Synchronous Get File Concepts

In the outbound direction, the Oracle FTP Adapter works the same way as the Synchronous Read File operations of the Oracle File Adapter in that it polls and gets files from a file system and reads the current contents of the file. The major difference is that the Oracle FTP Adapter is used for remote file exchanges. Because of this polling, the Adapter Configuration Wizard asks for connection information to an FTP server to be used later. For reading a file synchronously, you select Synchronous Get File operation, as shown in Figure 4–41.

Figure 4–41 Selecting the Synchronous Get File Operation



4.3.8 Oracle FTP Adapter File Listing Concepts

The Oracle FTP Adapter file listing concepts are similar to the Oracle File Adapter file listing concepts discussed in Section 4.3.4, "Oracle File Adapter File Listing Concepts." The Oracle FTP Adapter polls for files in a target directory and lists files from the target directory to specified FTP locations. The contents of the files are not read. This feature of the Oracle FTP Adapter lets you use an invoke activity to retrieve a list of files from a target directory. This list of files is returned as an XML document and contains information such as file name, directory name, file size, and last modified time.

> **Note:** The file creation time property, `creationTime`, is not supported for FTP because the standard Java APIs do not provide a mechanism to retrieve the creation time. The value of the `creationTime` property is always displayed as `0`.
>
> The `creationTime` property is supported for SFTP only.

You use the Adapter Configuration Wizard to configure the Oracle FTP Adapter for use with a BPEL process or a Mediator service. This creates an outbound `WSDL` and `JCA` file pair.

For listing files, you must select the `List Files` operation from the Operation Type page of the Adapter Configuration Wizard. In the File Directories page of the Adapter Configuration Wizard, you must specify information about the directory to use for reading file names for the list operation. You can choose to list files recursively within directories. The File Filtering page of the Adapter Configuration Wizard enables you to specify details of the files to retrieve or ignore.

The Oracle FTP Adapter acts as a listener and polls the specified directory on a local or remote file system and looks for files that match specified naming criteria.

4.4 Configuring Oracle File and FTP Adapters

Various configuration tasks for Oracle File and FTP Adapters are discussed in the following sections:

- Section 4.4.1, "Configuring the Credentials for Accessing a Remote FTP Server"
- Section 4.4.2, "Configuring Oracle File and FTP Adapters for High Availability"
- Section 4.4.3, "Using Secure FTP with the Oracle FTP Adapter"
- Section 4.4.4, "Using SFTP with Oracle FTP Adapter"
- Section 4.4.5, "Configuring Oracle FTP Adapter for HTTP Proxy"

4.4.1 Configuring the Credentials for Accessing a Remote FTP Server

To access a remote FTP server, you must configure the following credentials:

- User name: the user name to use on the remote FTP server.
- Password: the password to use on the remote FTP server.
- Port: 21
- Host: the IP address of the remote FTP server.

You must configure these credentials by modifying the `weblogic-ra.xml` file using the Oracle WebLogic Server console.

To do so, in the Oracle WebLogic Server Admin Console:

1. Select **Deployments** from the Navigation pane on the left.
2. Select **FtpAdapter** from the table of Deployments shown on the right.
3. Select the **Configuration** subtab for the FtpAdapter and then **Outbound Connection Pools**.
4. Expand `javax.resource.cci.ConnectionFactory` and then select the instance that you are modifying. (For example, choose the `eis/Ftp/FtpAdapter` instance for the non-HA use case.)

4.4.2 Configuring Oracle File and FTP Adapters for High Availability

The requirements and procedure to configure the Oracle File and FTP Adapters for high availability for an active-active topology are discussed in the following sections:

- Section 4.4.2.1, "Prerequisites for High Availability"
- Section 4.4.2.2, "High Availability in Inbound Operations"
- Section 4.4.2.3, "High Availability in Outbound Operations"

4.4.2.1 Prerequisites for High Availability

Before you configure the Oracle File or FTP Adapter for high availability, you must ensure that the following prerequisites are met:

- Clustered processes must use the same physical directory.
- Connection-factories must specify the same shared directory as the control directory, and their names must match. For example, if the deployment descriptor for one connection factory has `/shared/control_dir` as the value for `controlDir`, then the other deployment descriptor must also have the same value.

- Fault-policies and fault-bindings must be created for remote faults to ensure that the adapter acts correctly. For more information on fault-policies and fault-bindings, see Section 2.21, "Error Handling".
- The `MaxRaiseSize` property must be set in the inbound JCA file.

Note: For large payloads, you must increase the transaction time out for the `SOADataSource` by adding the following:

```
<xa-set-transaction-timeout>true</xa-set-transaction-timeout>
<xa-transaction-timeout>1000</xa-transaction-timeout>
```

Note: For Windows platforms, you must ensure that the input and output directories are canonicalized. For example, you must use `C:\bpel\input` instead of `c:\bpel\input`. Note the use of capitalized drive letter `C:` instead of `c:`.

Note: On all platforms, you must not end input or output directory names with the Java system property file.separator value. For example, `/tmp/file/in/` is invalid but `/tmp/file/in` is valid.

4.4.2.2 High Availability in Inbound Operations

The Oracle File and FTP Adapters must ensure that only one node processes a particular file in a distributed topology. You can use the database table as a coordinator to ensure that Oracle File and FTP Adapters are highly available for inbound operations.

Using Database Table as a Coordinator

You must use the following procedure to make an inbound Oracle File or FTP Adapter service highly available by using database table as a coordinator:

Note: You must increase global transaction timeouts if you use database as a coordinator.

---

1. **Create Database Tables**

   You are not required to perform this step because the database schemas are pre-created as a part of soainfra.

2. **Modify Deployment Descriptor for Oracle File Adapter**

   Modify Oracle File Adapter deployment descriptor for the connection-instance corresponding to `eis/HAFileAdapter` from the Oracle WebLogic Server Administration Console:

   a. Log in to your Oracle WebLogic Server Administration Console. To access the console, navigate to `http://`*`servername`*`:`*`portnumber`*`/console`.

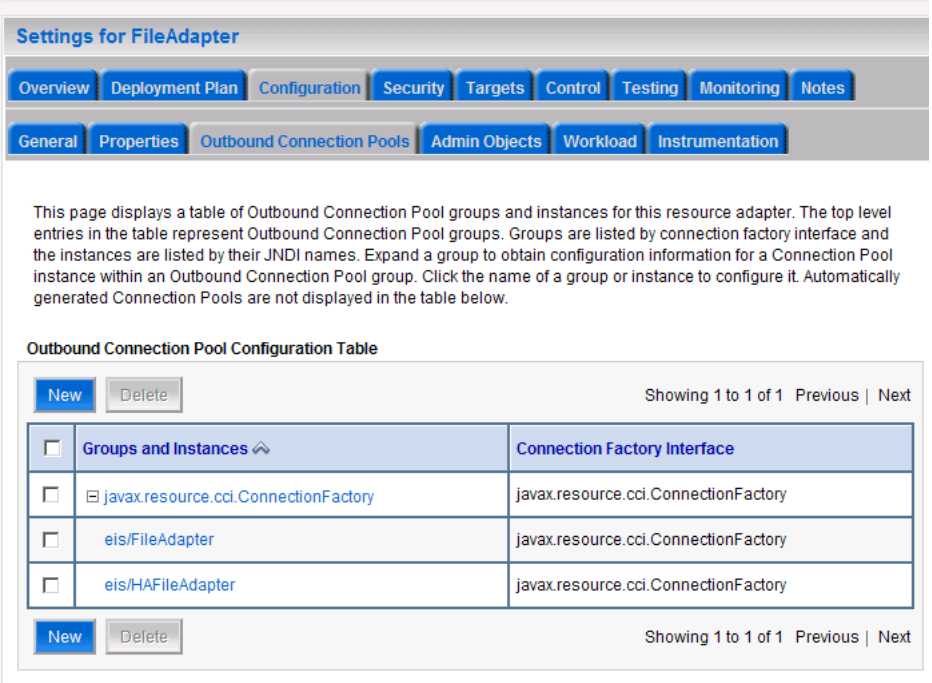
   b. Click **Deployments** in the left pane for Domain Structure.

   c. Click **FileAdapter** under Summary of Deployments on the right pane.

   d. Click the **Configuration** tab.

e. Click the **Outbound Connection Pools** tab, and expand **javax.resource.cci.ConnectionFactory** to see the configured connection factories, as shown in Figure 4–42:

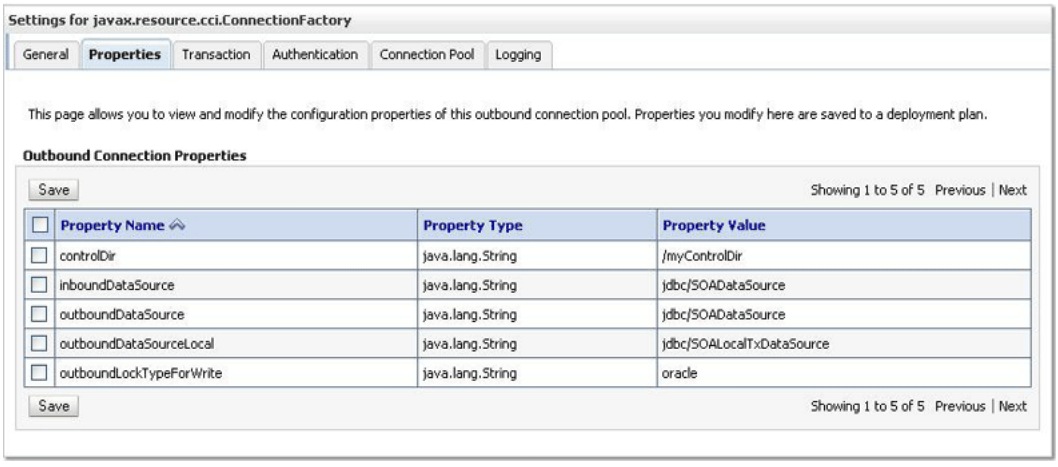
Figure 4–42 Oracle WebLogic Server Administration Console - Settings for FileAdapter Page



f. Click **eis/HAFileAdapter**. The Outbound Connection Properties for the connection factory corresponding to high availability is displayed.

g. Update the connection factory properties, as shown in Figure 4–43.

Figure 4–43 Oracle WebLogic Server Administration Console - Settings for javax.resource.cci.ConnectionFactory Page



The new parameters in connection factory for Oracle File and FTP Adapters are as follows:

`controlDir` - Set it to the directory structure where you want the control files to be stored. You must set it to a shared location if multiple WebLogic Server instances run in a cluster.

`inboundDataSource` - Set the value to `jdbc/SOADataSource`. This is the data source, where the schemas corresponding to high availability are pre-created. The pre-created schema file can be found under `$BEA_HOME/AS11gR1SOA/rcu/integration/soainfra/sql/adapter/createschema_adapter_oracle.sql`. To create the schemas elsewhere, use this script. You must set the inboundDataSource property accordingly if you choose a different schema.

h. Configure BPEL Process or Mediator Scenario to use the connection factory, as shown in the following example:

```
<adapter-config name="FlatStructureIn" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/HAFileAdapter"
UIincludeWildcard="*.txt" adapterRef=""/>
  <endpoint-activation portType="Read_ptt" operation="Read">
    <activation-spec
className="oracle.tip.adapter.file.inbound.FileActivationSpec"../>
      <property../>
      <property../>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

Note: The location attribute is set to `eis/HAFileAdapter` for the connection factory.

4.4.2.3 High Availability in Outbound Operations

The Oracle File and FTP Adapters must ensure that if multiple references write to the same directory, then these do not overwrite each other. The following locking capabilities you can use to make Oracle File and FTP Adapters highly available for outbound operations:

- Database mutex
- User-defined mutex

Using a Database Mutex

You must use the following procedure to make an outbound Oracle File or FTP Adapter service highly available by using database table as a coordinator:

Note: You must increase global transaction timeouts if you use the database as a coordinator.

1. **Create Database Tables**

   You are not required to perform this step as the database schemas are precreated as a part of soainfra.

2. **Modify Deployment Descriptor for Oracle File Adapter**

Modify Oracle File Adapter deployment descriptor for the connection-instance corresponding to `eis/HAFileAdapter` from the Oracle WebLogic Server Administration Console:

a. Log in to your Oracle WebLogic Server Administration Console. To access the console, navigate to `http://servername:portnumber/console`.

b. Click **Deployments** in the left pane for Domain Structure.

c. Click **FileAdapter** under Summary of Deployments on the right pane.

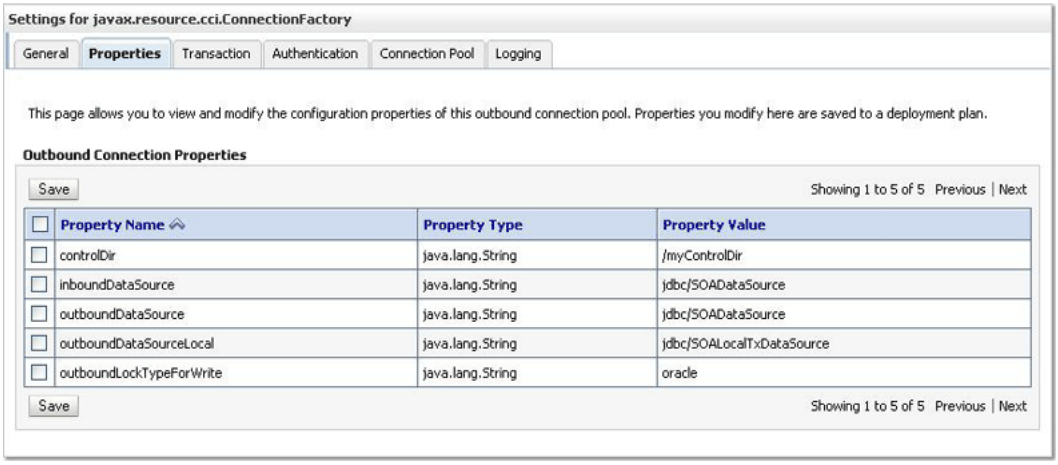
d. Click the **Configuration** tab.

e. Click the **Outbound Connection Pools** tab, and expand **javax.resource.cci.ConnectionFactory** to see the configured connection factories, as shown in Figure 4–42.

f. Click **eis/HAFileAdapter**. The Outbound Connection Properties page is displayed with the connection factory corresponding to high availability.

g. Update the connection factory properties, as shown in Figure 4–44.

Figure 4–44 Oracle WebLogic Server Administration Console - Settings for javax.resource.cci.Connectionfactory Page



The new parameters in connection factory for Oracle File and FTP Adapters are as follows:

`controlDir` - Set it to the directory structure where you want the control files to be stored. You must set it to a shared location if multiple WebLogic Server instances run in a cluster.

`inboundDataSource` - Set the value to `jdbc/SOADataSource`. This is the data source, where the schemas corresponding to high availability are precreated. The precreated schemas can be found under `$BEA_HOME/AS11gR1SOA/rcu/integration/soainfra/sql/adapter/createschema_adapter_oracle.sql`. To create the schemas elsewhere, use this script. You must set the inboundDataSource property accordingly if you choose a different schema.

`outboundDataSource` - Set the value to `jdbc/SOADataSource`. This is the data source where the schemas corresponding to high availability are precreated. The precreated schemas can be found under `$BEA_HOME/AS11gR1SOA/rcu/integration/soainfra/sql/adapter/creat`

eschema_adapter_oracle.sql. To create the schemas elsewhere, use this script. You must set the outboundDataSource property if you choose to do so.

outboundLockTypeForWrite - Set the value to oracle if you are using Oracle Database. By default the Oracle File and FTP Adapters use an in-memory mutex to lock outbound write operations. You must choose from the following values for synchronizing write operations:

memory - The Oracle File and FTP Adapters use an in-memory mutex to synchronize access to the file system.

oracle - The adapter uses the Oracle Database sequence.

db - The adapter uses a precreated database table (FILEADAPTER_MUTEX) as the locking mechanism. You must use this option only if you are using a schema other than the Oracle Database schema.

user-defined - The adapter uses a user-defined mutex. To configure the user-defined mutex, you must implement the mutex interface "oracle.tip.adapter.file.Mutex" and then configure a new binding-property with the name "oracle.tip.adapter.file.mutex" and value as the fully qualified class name for the mutex for the outbound reference.

h. Configure BPEL Process or Mediator Scenario to use the connection factory, as shown in the following example:

```
<adapter-config name="FlatStructureOut" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/HAFileAdapter" adapterRef=""/>
  <endpoint-interaction portType="Write_ptt" operation="Write">
<interaction-spec
className="oracle.tip.adapter.file.outbound.FileInteractionSpec">
      <property../>
      <property../>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

Note: The location attribute is set to eis/HAFileAdapter for the connection factory.

4.4.3 Using Secure FTP with the Oracle FTP Adapter

The Oracle FTP Adapter supports the use of the secure FTP feature on Windows, Solaris, and Linux. For Windows, this feature is certified on FileZilla FTP server with OpenSSL. This section provides an overview of secure FTP functionality and describes how to install and configure this feature.

This section includes the following topics:

- Section 4.4.3.1, "Secure FTP Overview"
- Section 4.4.3.2, "Installing and Configuring FTP Over SSL on Solaris and Linux"
- Section 4.4.3.3, "Installing and Configuring FTP Over SSL on Windows"

4.4.3.1 Secure FTP Overview

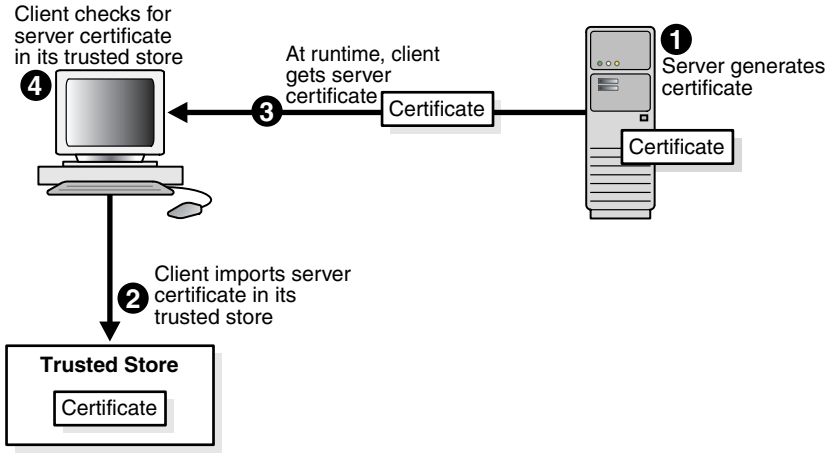
In environments in which sensitive data is transferred to remote servers (for example, sending credit card information to HTTP servers), the issue of security is very important. Security in these cases primarily refers to two requirements:

- Trust in the remote server with which you are exchanging data
- Protection from third parties trying to intercept the data

Secure socket layer (SSL) certificates and encryption focus on satisfying these two security requirements. When SSL is used for FTP, the resulting security mechanism is known as FTPS (or FTP over SSL).

To gain the trust of clients in SSL environments, servers obtain certificates (typically, X.509 certificates) from recognized certificate authorities. When you set up the FTP server, you use openSSL to create a certificate for the server. Every client trusts a few parties, to begin with. If the server is one of these trusted parties, or if the server's certificate was issued by one of these parties, then you have established trust, even indirectly. For example, if the server's certificate was issued by authority A, which has a certificate issued by authority B, and the client trusts B, that is good enough. For the setup shown in Figure 4–45, the server's certificate is directly imported into the client's certificate store (or Oracle Wallet) as a trusted certificate.

Figure 4–45 Establishing Trust



You make the data being transferred immune to spying by encrypting it before sending it and decrypting it after receiving it. Symmetric encryption (using the same key to encrypt and decrypt data) is much faster for large amounts of data than the public key and private key approach. Symmetric encryption is the approach used by FTPS. However, before the client and server can use the same key to encrypt and decrypt data, they must agree on a common key. This client typically does this by performing the following tasks:

- Generating a session key (to be used to encrypt and decrypt data)
- Encrypting this session key using the server's public key that is part of the server's certificate
- Sending the key to the server

The server decrypts this session key by using its private key and subsequently uses it to encrypt file data before sending it to the client.

4.4.3.2 Installing and Configuring FTP Over SSL on Solaris and Linux

The following subsections describe how to install and configure secure FTP for Solaris and Linux:

- Section 4.4.3.2.1, "Installing and Configuring OpenSSL"
- Section 4.4.3.2.2, "Installing and Configuring vsftpd"
- Section 4.4.3.2.3, "Creating an Oracle Wallet"
- Section 4.4.3.2.4, "Setting Up the Oracle FTP Adapter"

4.4.3.2.1 Installing and Configuring OpenSSL

OpenSSL is an open source implementation of the SSL protocol. OpenSSL implements basic cryptographic functions and provides utility functions. Install and configure OpenSSL on the Solaris or Linux host to be used as the FTP server.

1. Go to the following URL:

   ```
   http://www.openssl.org/source
   ```

2. Locate `openssl-0.9.7g.tar.gz` in the list of available files. For example:

   ```
   3132217 Apr 11 17:21:51 2005 openssl-0.9.7g.tar.gz (MD5) (PGP sign)
   ```

3. Download the following files:
   - `openssl-0.9.7g.tar.gz`
   - openssl-0.9.7g.tar.gz.md5 (under the `MD5` link)
   - openssl-0.9.7g.tar.gz.asc (under the `PGP sign` link
4. Unzip the following file using `gunzip`.

   ```
   gunzip openssl-0.9.7g.tar.gz
   ```

5. Untar the following file:

   ```
   tar xvf openssl-0.9.7g.tar
   ```

6. Change directories to the following location:

   ```
   cd openssl-0.9.7g
   ```

7. Run the following command:

   ```
   ./config --prefix=/usr --openssldir=/usr/local/openssl
   ```

8. Change to the Bourne shell (if you are not using it):

   ```
   sh
   ```

9. Configure and export the `PATH` variable:

   ```
   PATH=${PATH}:/usr/ccs/bin; export PATH
   ```

10. Run the following command:

    ```
    make
    ```

11. Exit the Bourne shell:

    ```
    exit
    ```

12. Run the following command:

```
make test
```

13. Log in as the super user:

```
msu
```

14. Enter the password when prompted.
15. Run the following command:

```
make install
```

4.4.3.2.2 Installing and Configuring vsftpd

The vsftpd server is a secure and fast FTP server for UNIX systems. Install and configure vsftpd on the Solaris or Linux host to be used as the FTP server.

1. Go to the following location:

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

2. Download `vsftpd-2.0.5` (You need the tar and signature file (`.asc` file)). For example:

```
[BINARY]     vsftpd-2.0.5.tar.gz. . . . . .    [Mar 19 21:26]    149K
[FILE]       vsftpd-2.0.5.tar.gz.asc. . . . . .    [Mar 19 21:26]    189B
```

3. Unzip the following file using `gunzip`.

```
gunzip vsftpd-2.0.5.tar.gz
```

4. Unzip the tar file:

```
tar xvf vsftpd-2.0.5.tar
```

5. Change directories to the following location:

```
cd vsftpd-2.0.5
```

6. Make the following change in the `builddefs.h` file:

```
#undef VSF_BUILD_SSL
```

to

```
#define VSF_BUILD_SSL
```

7. Log in as the super user:

```
msu
```

8. Enter the password when prompted.
9. Create a file named `vsftpd.conf` with the following settings in the `/etc` directory:

```
# Standalone mode
listen=YES
max_clients=200
max_per_ip=4
# Access rights
anonymous_enable=YES
#chroot_local_user=YES
#userlist_enable=YES
ftp_username=ftp
```

```
local_enable=YES
write_enable=YES
anon_upload_enable=YES
anon_mkdir_write_enable=YES
anon_other_write_enable=YES
chown_uploads=YES
chown_username=ftp
# Security
anon_world_readable_only=NO
allow_anon_ssl=YES
ssl_enable=YES
connect_from_port_20=YES
hide_ids=YES
pasv_min_port=50000
pasv_max_port=60000
# Features
ftpd_banner="Welcome to the FTP Service"
xferlog_enable=YES
ls_recurse_enable=NO
ascii_download_enable=NO
async_abor_enable=YES
# Performance
one_process_model=NO
idle_session_timeout=120
data_connection_timeout=300
accept_timeout=60
connect_timeout=60
anon_max_rate=50000
```

Note: Copies of the `vsftpd.conf` file appear in several locations in the `vsftpd-2.0.5` directory structure. If you use one of those files to create the `vsftpd.conf` file in the `/etc` directory, then ensure that it only includes the parameters and settings described in Step 9.

10. Run the following commands:

```
mkdir /var/ftp
useradd -d /var/ftp ftp
chown root /var/ftp
chmod og-w /var/ftp
mkdir /usr/share/empty
mkdir /usr/share/ssl
mkdir /usr/share/ssl/certs
```

11. Run the following command:

```
openssl req -x509 -nodes -newkey rsa:1024 -keyout
/usr/share/ssl/certs/vsftpd.pem -out /usr/share/ssl/certs/vsftpd.pem
```

12. Run the `vsftpd` daemon from the `vsftpd-2.0.5` directory:

```
./vsftpd
```

4.4.3.2.3 Creating an Oracle Wallet s

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 of Section 4.4.3.2.2, "Installing and Configuring vsftpd" as a trusted certificate in this wallet.
3. Save this wallet in PKCS # 12 (`.p12`) format.

See *Oracle Fusion Middleware Administrator's Guide* for details about using Oracle Wallet Manager.

4.4.3.2.4 Setting Up the Oracle FTP Adapter

Perform the following tasks to set up the Oracle FTP Adapter:

1. On your Solaris or Linux host, run the following commands:

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

2. Specify the FTP connection parameters in the Oracle FTP Adapter deployment descriptor from the Oracle WebLogic Server Administration Console.

Where...	Is...
`useFtps`	Set to `True`. This setting is required to use FTP over SSL. The default is `False`.
`walletLocation`	The location of the wallet created in Section 4.4.3.2.3, "Creating an Oracle Wallet."
`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`.
`securePort`	The port for FTP over SSL. The default is `990`.
`keyStoreProviderName`	The keystore provider class. The default is `oracle.security.pki.OraclePKIProvider`.
`keystoreType`	The keystore type. The default is `PKCS12`.
`keystoreAlgorithm`	The keystore algorithm. The default is `OracleX509`.
`enableCipherSuits`	List of comma separated cipher suites. The default is blank, in which case the default list of cipher suites are used. For most cases, you are not required to change this.
`pkiProvider`	The PKI provider name. The default is `OraclePKI`.
`jsseProvider`	The JSSE provider name. The default is `OracleJSSE`.

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

4.4.3.3 Installing and Configuring FTP Over SSL on Windows

The FTPS feature is certified on FileZilla FTP server with OpenSSL. You must follow the procedure in the following subsections for installing and configuring OpenSSL for FileZilla on Windows:

- Section 4.4.3.3.1, "Installing OpenSSL"

- Section 4.4.3.3.2, "Generating OpenSSL Server Key and Certificate"
- Section 4.4.3.3.3, "Importing the Server Key and Certificate Into FileZilla Server"
- Section 4.4.3.3.4, "Converting the Server Key From PEM to PKCS12 Format"
- Section 4.4.3.3.5, "Configuring Oracle FTP Adapter Deployment Descriptor to Use the New Key"

4.4.3.3.1 Installing OpenSSL

OpenSSL is an open source implementation of the SSL protocol. OpenSSL implements basic cryptographic functions and provides utility functions. Perform the following steps to install and configure OpenSSL on the Windows host to be used as the FTP server.

1. Go to the following URL:

   `http://www.slproweb.com/products/Win32OpenSSL.html`

2. Download and install `Visual C++ 2008 Redistributables`.
3. Download and install `Win32 OpenSSL v0.9.8k Light`.

4.4.3.3.2 Generating OpenSSL Server Key and Certificate

To create the server key and certificate files, you must perform the following steps:

1. Open the command prompt and browse to the `OpenSSL\bin` directory.
2. Run the following command:

   ```
   openssl req -new -x509 -keyout mykey.pem -out mycert.pem -days 365
   ```

   A sample command output is as follows:

   ```
   C:\OpenSSL\bin>openssl req -new -x509 -keyout mykey.pem -out mycert.pem -days
   365
   Loading 'screen' into random state - done
   Generating a 1024 bit RSA private key
   ..........++++++
   .......++++++
   writing new private key to 'mykey.pem'
   Enter PEM pass phrase:
   Verifying - Enter PEM pass phrase:
   -----
   You are about to be asked to enter information that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or a DN.
   There are quite a few fields but you can leave some blank
   For some fields there will be a default value,
   If you enter '.', the field will be left blank.
   -----
   Country Name (2 letter code) [AU]:US
   State or Province Name (full name) [Some-State]:CA
   Locality Name (eg, city) []:Belmont
   Organization Name (eg, company) [Internet Widgits Pty Ltd]:Test
   Organizational Unit Name (eg, section) []:Test
   Common Name (eg, YOUR name) []:Test test
   Email Address []:test@test.com
   ```

3. Enter a PEM pass phrase when prompted.
4. Re-enter PEM pass phrase entered in step 3 for verification.

5. Enter the requested details.

   The server key (`mykey.pem`) and certificate (`mycert.pem`) are generated in the `OpenSSL\bin` directory.

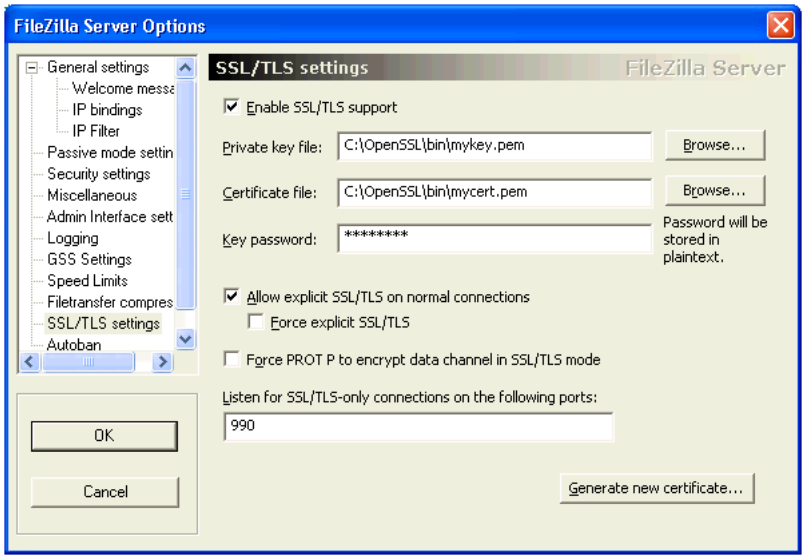
4.4.3.3.3 Importing the Server Key and Certificate Into FileZilla Server

To import the server key and certificate into FileZilla, you must perform the following steps:

1. Open a FileZilla Server interface from your Windows Start menu.
2. Click **Edit**, and then click **Settings**.

   The FileZilla Server Options dialog is displayed.
3. Click **SSL/TLS settings**.
4. Enter the server key and certificate details as shown in Figure 4–46.

Figure 4–46 The FileZilla Server Options Dialog



> **Note:** In the **Key password** field, you must use the PEM pass phrase generated in Step 3 of Section 4.4.3.3.2, "Generating OpenSSL Server Key and Certificate."

4.4.3.3.4 Converting the Server Key From PEM to PKCS12 Format

You must convert the server key and the server certificate from the PEM format to the PKCS#12 format as the Oracle FTP Adapter does not recognize the PEM format. To convert the server key and certificate to the PKCS#12 format, you must perform the following steps:

1. Open the command prompt and browse to the `OpenSSL\bin` directory.
2. Run the following command:

   ```
   openssl pkcs12 -export -out mykeyz.p12 -in mycert.pem  -inkey mykey.pem
   ```

   The command output is as follows:

   ```
   C:\OpenSSL\bin>openssl pkcs12 -export -out mykeyz.p12 -in mycert.pem  -inkey
   ```

```
mykey.pem
Loading 'screen' into random state - done
Enter pass phrase for mykey.pem:
Enter Export Password:
Verifying - Enter Export Password:
```

3. Enter a PEM pass phrase when prompted. This is the pass phrase that you created while generating OpenSSL server key and certificate in Section 4.4.3.3.2, "Generating OpenSSL Server Key and Certificate."
4. Enter an export password for the PKCS#12 file.
5. Re-enter the export password for verification.
6. Enter the requested details.

   The `mykeyz.p12` file is generated in the `OpenSSL\bin` directory.
7. Copy the `mykeyz.p12` file to the managed Oracle WebLogic Server instance running the Oracle FTP Adapter.

   For example,

   ```
   /scratch/$user/private/mykeyz.p12
   ```

4.4.3.3.5 Configuring Oracle FTP Adapter Deployment Descriptor to Use the New Key

You must perform the following steps to configure the Oracle FTP Adapter deployment descriptor:

1. Navigate to `http://`*`servername:portnumber`*`/console`.
2. Use the required credentials to open the home page of the Oracle WebLogic Server Administration Console.
3. Select **Deployments** in the Domain Structure pane.

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

   The Oracle WebLogic Server Administration Console - Settings for FtpAdapter page is displayed.
5. Click the **Configuration** tab, and then click the **Outbound Connection Pools** tab.

   The Outbound Connection Pool Configuration table is displayed.
6. Select the JNDI name for the Ftp Adapter instance you configure. For example, `"eis/Ftp/FtpAdapter"`.
7. Configure the deployment descriptors listed in Table 4–10:

Table 4–10 JCA Properties for Oracle File and FTP Adapters

Property Name	Property Value
`useFtps`	Set the value to `true`.
`walletLocation`	Set it to the location of the PKCS#12 file in the managed Oracle WebLogic Server instance: `/scratch/$user/private/mykeyz.p12`.
`walletPassword`	Set the value to the export password generated in Step 4 of Section 4.4.3.3.4, "Converting the Server Key From PEM to PKCS12 Format."
`keyStoreProviderName`	Set the value to `sun.security.provider.Sun`

Table 4–10 (Cont.) JCA Properties for Oracle File and FTP Adapters

Property Name	Property Value
`keystoreType`	Set the value to `PKCS12`
`keystoreAlgorithm`	Set the value to `SunX509`
`pkiProvider`	Must be left blank.
`jsseProvider`	Must be left blank.

4.4.4 Using SFTP with Oracle FTP Adapter

SSH file transfer protocol (SFTP) is a network protocol that enables secure file transfer over a network. Oracle FTP Adapter supports the use of the SFTP feature on Windows and Linux. This section provides an overview of the SFTP functionality and describes how to install and configure this feature.

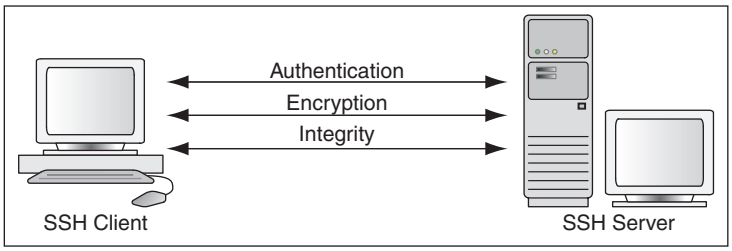
This section includes the following tasks:

- SFTP Overview
- Install and Configure OpenSSH for Windows
- Set Up Oracle FTP Adapter for SFTP

4.4.4.1 SFTP Overview

FTP is the network protocol that enables clients to securely transfer files over the underlying SSH transport. SFTP is not similar to FTP over SSH or File Transfer Protocol (FTP). Figure 4–47 displays the communication process between an SSH client and an SSH server. SFTP is supported in Windows and Linux.

Figure 4–47 SFTP Communication



SFTP has the following features:

- Encryption
- Authentication
- Integrity
- Data Compression

4.4.4.1.1 Encryption The SSH protocol uses public key cryptography for encryption. This section explains how data is encrypted:

1. The SSH subsystem uses symmetric key ciphers such as Data Encryption Standard (DES) or Blowfish to generate a session key. The SSH protocol currently uses the Diffie-Hellman Key Exchange Algorithm to derive the symmetric key for the session.
2. The data is encrypted using the session key.

3. The session key is encrypted by using the recipient's public key. Because the recipient has the private key, it can decrypt the message by using its preferred PKI algorithm such as Rivest-Shamir-Adleman (RSA) or Digital Signature Algorithm (DSA).

4.4.4.1.2 Authentication The SSH protocol inherently supports password authentication by encrypting passwords or session keys as they are transferred over the network. In addition, the SSH protocol uses a mechanism known as 'known hosts' to prevent threats such as IP spoofing. When this mechanism is used, both the client and the server have to prove their identity to each other before any kind of communication exchange.

4.4.4.1.3 Integrity The SSH protocol uses widely trusted bulk hashing algorithms such as Message Digest Algorithm 5 (MD5) or Secure Hash Algorithm (SHA-1) to prevent insertion attacks. Implementation of data integrity checksum by using the algorithms mentioned in Section 4.4.4.1.1, "Encryption" prevents deliberate tampering of data during transmission.

4.4.4.1.4 Data Compression The SSH protocol supports zlib, an open-source cross-platform algorithm for data compression. SSH uses `zlib` to compress in-flight data to reduce network bandwidth.

4.4.4.2 Install and Configure OpenSSH for Windows

OpenSSH for Windows is the free implementation of the SSH protocol on Windows. Perform the following steps to install and configure OpenSSH on Windows XP:

1. Log in as a user with Administrator privileges.
2. Download `setup.exe` from the following location:

   `http://www.cygwin.com`

3. Run `setup.exe`. The Cygwin Net Release Setup window is displayed.
4. Click **Next**. The Choose Installation type window is displayed.
5. Select **Install from Internet** as the download source and click **Next**. The Choose Installation Directory window is displayed.
6. Leave the root directory as C:\cygwin. Also, keep the default options for the Install For and the Default Text File Type fields.
7. Click **Next**. The Select Local Package Directory window is displayed.
8. Click **Browse** and select **C:\cygwin** as the local package directory.
9. Click **Next**. The Select Connection Type window is displayed.
10. Select a setting for Internet connection and click **Next**. The Choose Download Site(s) window is displayed.
11. Select a site from the Available Download Sites list and click **Next**. The Select Packages window is displayed.
12. Click **View** to see the complete list of packages available for installation.
13. Select **openssh** if it is not the default value.
14. Select the **Binaries** box for openssh.
15. Click **Next** to start the installation.
16. On Windows XP desktop, right -click **My Computer** and select **Properties**.

17. Click the **Advanced** tab and click **Environment Variables**.
18. Click **New** and enter `CYGWIN` in the **Variable Name** field and `ntsec` in the **Variable Value** field.
19. Add `C:\cygwin\bin` to the system path.
20. Open the cygwin window.
21. Type `ssh-host-config`.
22. You are prompted with the following questions:
    a. `Shall privilege separation be used? (yes/no)`

       Enter `yes`.

    b. `Shall this script create a local user 'sshd' on this machine?`

       Enter `yes`.

    c. `Do you want to install sshd as service?`

       `(Say "no" if it's already installed as service) (yes/no)`

       Enter `yes`.

    d. `Which value should the environment variable CYGWIN have when sshd starts? It's recommended to set at least "ntsec" to be able to change user context without password. Default is "binmode ntsec tty".`

       Enter `ntsec`.
23. Type `net start sshd` to start the sshd service.
24. Run the following command in the cygwin window to replicate the Windows local user accounts to cygwin:

```
mkpasswd --local > /etc/passwd
mkgroup --local > /etc/group
```

25. To test the setup, type `ssh localhost` in the cygwin window.

4.4.4.3 Set Up Oracle FTP Adapter for SFTP

To use the SFTP functionality, you must modify the deployment descriptor for Oracle FTP Adapter.

Table 4–11 lists the properties for which you must specify a value in the deployment descriptor. The values of these properties depend on the type of authentication and the location of OpenSSH.

Table 4–11 SFTP Properties

Property	Description
`useSftp`	Specify `true`. **Mandatory**: Yes **Default value**: `false`

Table 4–11 (Cont.) SFTP Properties

Property	Description
authenticationType	Specify PASSWORD for password-based authentication or PUBLICKEY for public key authentication. For password-based authentication, the user name and password specified in the weblogic-ra.xml file are used. Ensure that there is a Windows user with the same name and password as specified in the weblogic-ra.xml file. In addition, the user should have administrative privileges. For public key authentication, the privateKeyFile parameter must be set to the location of the private key file. **Mandatory**: Yes
preferredKey ExchangeAlgorithm	Specify diffie-hellman-group1-sha1 or diffie-hellman-group-exchange-sha1. This is an optional parameter where the user can select the default key exchange protocol for negotiating the session key for encrypting the message. **Mandatory**: No **Default value**: diffie-hellman-group1-sha1
preferred CompressionAlgorithm	Specify none or zlib. This parameter enables the user to choose whether in-flight data should be compressed or not. **Mandatory**: No
preferred DataIntegrityAlgorithm	Specify hmac-md5 or hmac-sha1. This parameter enables the user to select the bulk-hashing algorithm for data integrity checks. **Mandatory**: No **Default value**: hmac-md5
preferredPKIAlgorithm	Specify ssh-rsa or ssh-dsa. This parameter enables the user to configure the asymmetric cipher for the communication. **Mandatory**: No **Default value**: ssh-rsa
privateKeyFile	Specify the path to the private key file. This is required if the authenticationType parameter is set to PUBLICKEY. **Mandatory**: No

Table 4–11 (Cont.) SFTP Properties

Property	Description
`preferredCipherSuite`	Specify a cipher from the following list: ■ twofish192-cbc ■ cast128-cbc ■ twofish256-cbc ■ aes128-cbc ■ twofish128-cbc ■ 3des-cbc ■ blowfish-cbc ■ aes256-cbc ■ aes192-cbc **Mandatory**: No **Default value**: blowfish-cbc
`transportProvider`	Specify `socket` or `HTTP`. Specify `socket` if the SSH server is inside a firewall. Specify `HTTP` if the SSH server is outside the firewall or a server is exposed through an HTTP server. If you select HTTP, then you must provide values for the following parameters: ■ `proxyHost` ■ `proxyPort` ■ `proxyUser` ■ `proxyPassword` ■ `useProxy` **Mandatory**: Yes

4.4.4.3.1 Configuring Oracle FTP Adapter for Password Authentication

To set up the Oracle FTP Adapter for password authentication, the deployment descriptor for Oracle FTP Adapter must specify the values of the properties listed in Table 4–11. Ensure that the `authenticationType` property is set to `password`.

Specify the following properties and values listed in Table 4–12:

Table 4–12 Sample SFTP Properties and Values

Property	Value
`useSftp`	`true`
`authenticationType`	`PASSWORD`
`preferredKey ExchangeAlgorithm`	`diffie-hellman-group1-sha1`
`preferred CompressionAlgorithm`	`none`
`preferred DataIntegrityAlgorithm`	`hmac-md5`
`preferredPKIAlgorithm`	`ssh-rsa`
`privateKeyFile`	-

Table 4–12 (Cont.) Sample SFTP Properties and Values

Property	Value
preferredCipherSuite	blowfish-cbc
transportProvider	socket

4.4.4.3.2 Configuring Oracle FTP Adapter for Public Key Authentication

For public key authentication, you must first configure OpenSSH and then set up the Oracle FTP Adapter. The Oracle FTP Adapter setup depends on whether the OpenSSH is running inside a firewall or outside a firewall. If OpenSSH is running inside the firewall, then see the following sections:

- Section 4.4.4.3.3, "Configuring OpenSSH for Public-Key Authentication"
- Section 4.4.4.3.4, "Configuring Oracle FTP Adapter for Public Key Authentication with OpenSSH Running Inside a Firewall"

If OpenSSH is running outside the firewall, then see the following sections:

- Section 4.4.4.3.3, "Configuring OpenSSH for Public-Key Authentication"
- Section 4.4.4.3.5, "Configuring Oracle FTP Adapter for Public Key Authentication with OpenSSH Running Outside a Firewall"

4.4.4.3.3 Configuring OpenSSH for Public-Key Authentication

Perform the following steps:

1. Go to the `C:\cygwin\etc` directory. If required, configure the `sshd_config` file to force public key authentication. For more information, see openssh help or manual.
2. Go to the `C:\cygwin\bin` directory.
3. Run the following command to generate the key pair:

   ```
   ssh-keygen -t rsa
   ```

4. Enter `/etc/id_rsa` when prompted for the file in which the key should be saved.
5. Enter the passphrase.
6. Enter the passphrase again.
7. Go to the `/etc` directory and verify that both the public key file (`id_rsa.pub`) and the private key file (`id_rsa`) are generated.
8. Run the following command to create a copy of the public key file:

   ```
   cp id_rsa.pub authorized_keys
   ```

9. Create a copy of the private key file in a secured location such as `C:\my-secured-folder\`. The Oracle FTP Adapter configuration refers to this private key file.
10. Restart the OpenSSH server by running the following commands:

    ```
    net stop sshd
    net start sshd
    ```

4.4.4.3.4 Configuring Oracle FTP Adapter for Public Key Authentication with OpenSSH Running Inside a Firewall

To set up the Oracle FTP Adapter for public key authentication, you must specify the values of the parameters listed in Table 4–11 in the deployment descriptor. Ensure that the `authenticationType` parameter is set to `publickey` and the `transportProvider` parameter is set to `socket`. The `privateKeyFile` parameters should contain the location of the private key file.

A sample list of public key authentication properties and their values is shown in Table 4–13.

Table 4–13 Sample SFTP Properties and Values

Property	Value
`useSftp`	`true`
`authenticationType`	`publickey`
`preferredKey ExchangeAlgorithm`	`diffie-hellman-group1-sha1`
`preferred CompressionAlgorithm`	`none`
`preferred DataIntegrityAlgorithm`	`hmac-md5`
`preferredPKIAlgorithm`	`ssh-rsa`
`privateKeyFile`	C:\my-secured-folder\id_rsa
`preferredCipherSuite`	blowfish-cbc
`transportProvider`	`socket`

4.4.4.3.5 Configuring Oracle FTP Adapter for Public Key Authentication with OpenSSH Running Outside a Firewall

Perform the following steps to set up the Oracle FTP Adapter for public key authentication when OpenSSH is running outside the firewall:

1. In the deployment descriptor for Oracle FTP Adapter, you must specify the values of the properties listed in Table 4–11 in the deployment descriptor for Oracle FTP Adapter. Ensure that the `authenticationType` property is set to `publickey` and the `transportProvider` property is set to `HTTP`. The `privateKeyFile` property contains the location of the private key file.
2. In the deployment descriptor for Oracle FTP Adapter, also specify the following proxy-related properties:
   - `proxyHost`: The name of the proxy host.
   - `proxyPort`: The port number of the proxy.
   - `proxyUsername`: The user name for the proxy.
   - `proxyPassword`: The password for the proxy.
   - `useProxy`: Specify `true` to use proxy.

A sample list with public key authentication properties and proxy properties is shown in Table 4–14.

Table 4–14 Sample SFTP Properties and Values

Property	Value
proxyHost	proxy.host.com
proxyPort	80
proxyUsername	anonymous
proxyPassword	tiger@scott.com
useProxy	true
useSftp	true
authenticationType	publickey
preferredKey ExchangeAlgorithm	diffie-hellman-group1-sha1
preferred CompressionAlgorithm	none
preferred DataIntegrityAlgorithm	hmac-md5
preferredPKIAlgorithm	ssh-rsa
privateKeyFile	C:\my-secured-folder\id_rsa
preferredCipherSuite	blowfish-cbc
transportProvider	HTTP

4.4.5 Configuring Oracle FTP Adapter for HTTP Proxy

The Oracle FTP Adapter provides proxy support for HTTP proxy only. The HTTP proxy support is available in the following two modes, plain FTP mode and SFTP mode. This section explains how to configure the Oracle FTP Adapter for running in plain FTP mode and SFTP mode. It contains following sections:

- Section 4.4.5.1, "Configuring for Plain FTP Mode"
- Section 4.4.5.2, "Configuring for SFTP Mode"

4.4.5.1 Configuring for Plain FTP Mode

For running the Oracle FTP Adapter in plain FTP mode, you must specify the value of certain parameters in the Oracle FTP Adapter deployment descriptor. Table 4–15 lists the properties that you must modify.

Table 4–15 Plain FTP Mode Properties

Property	Description
host	The remote FTP server name.
port	The FTP control port number.
username	The FTP user name.
password	The FTP password.
proxyHost	The proxy host name.
proxyPort	The proxy port number.
proxyUsername	The proxy user name.
proxyPassword	The proxy password.

Table 4–15 (Cont.) Plain FTP Mode Properties

Property	Description
`proxyType`	The proxy type. Only HTTP proxy type is supported.
`proxyDefinitionFile`	The absolute path of the proxy definition file. This parameter is not mandatary. See Section 4.4.5.1.1, "Proxy Definition File" for more information.
`useProxy`	Specify `true` to use proxy.

A sample list of Oracle FTP Adapter descriptor properties and their values is shown in Table 4–16.

Table 4–16 Sample Plain FTP Mode Properties and Values

Property	Value
`host`	`my.host.com`
`port`	`21`
`username`	`user`
`password`	`password`
`proxyHost`	`proxy.host.com`
`proxyPort`	`80`
`proxyUsername`	`anonymous`
`proxyPassword`	`tiger@scott.com`
`proxyType`	`http`
`proxyDefinitionFile`	`c:\proxydefinitions.xml`
`useProxy`	`true`

4.4.5.1.1 Proxy Definition File You can specify all proxy-specific information in a proxy definition file and configure the adapter to use this file with the `proxyDefinitionFile` property of the Oracle FTP Adapter deployment descriptor file. A proxy definition file is written in XML format and is based on XML schema. The XML schema for the proxy definition file is shown in Example 4–5. Your proxy definition file must be based on this XML schema.

Example 4–5 Proxy Definition File XML Schema

```
<?xml version = \"1.0\" encoding = \"UTF-8\"?>
<schema targetNamespace = "http://ns.oracle.com/ip/af/ftp/proxy" xmlns =
"http://www.w3.org/2001/XMLSchema"
xmlns:proxy="http://ns.oracle.com/ip/af/ftp/proxy">

        <element name="ProxyDefinitions" type="proxy:ProxyDefinitionsType"/>
             <complexType name="ProxyDefinitionsType">
                     <sequence>
                          <element name="Proxy" type="proxy:ProxyDefinition"
                           minOccurs="0" maxOccurs="unbounded"/>
                     </sequence>
             </complexType>

             <complexType name="ProxyDefinition">
```

```
                    <sequence>
                         <element name="Step" type="proxy:StepType"
                          minOccurs="1" maxOccurs="unbounded"/>
                    </sequence>
                    <attribute name="key" type="ID" use="required"/>
                    <attribute name="description" type="string"
                     use="required"/>
                    <attribute name="type" type="proxy:Protocol"
                     use="optional"/>
               </complexType>

               <complexType name="StepType">
                   <simpleContent>
                      <extension base="string">
                          <attribute name="command" type="string"
use="required"/>
                          <attribute name="args" type="string" use="required"/>
                      </extension>
                   </simpleContent>
               </complexType>

                <simpleType name="Protocol">
                     <restriction base="string">
                          <enumeration value="ftp" />
                          <enumeration value="http" />
                     </restriction>
                </simpleType>
</schema>
```

A sample proxy definition file, based on the XML schema in Example 4–5, would look as shown in Example 4–6:

Example 4–6 Proxy Definition File

```
<?xml version = '1.0' standalone = 'yes'?>
<proxy:ProxyDefinitions xmlns:proxy="http://ns.oracle.com/ip/af/ftp/proxy">
<Proxy key="http" description="http" type="http">
<Step command="USER" args="remote_username" />
<Step command="PASS" args="remote_password" />
</Proxy>
</proxy:ProxyDefinitions>
```

When you use the file in Example 4–6, the Oracle FTP Adapter sends the following sequence of commands to log in:

1. USER remote_username
2. PASS remote_password

You can also direct the proxy definition file to pick values from the deployment descriptor for Oracle FTP Adapter. You can use the following expressions for this:

- `$proxy.user`: This corresponds to the value of the `proxyUsername` parameter in the Oracle FTP Adapter deployment descriptor.
- `$proxy.pass`: This corresponds to the value of the `proxyPassword` parameter in the Oracle FTP Adapter deployment descriptor.
- `$remote.user`: This corresponds to the value of the `username` parameter in the Oracle FTP Adapter deployment descriptor.

- `$remote.pass`: This corresponds to the value of the `password` parameter in the Oracle FTP Adapter deployment descriptor.
- `$remote.host`: This corresponds to the value of the `host` parameter in the Oracle FTP Adapter deployment descriptor.
- `$remote.port`: This corresponds to the value of the `port` parameter in the Oracle FTP Adapter deployment descriptor.

A sample proxy definition file based on the XML schema in Example 4–6 and taking values from the `weblogic-ra.xml` file is shown in Example 4–7:

Example 4–7 Proxy Definition File Taking Values from the Deployment Descriptor

```
<?xml version = '1.0' standalone = 'yes'?>
<proxy:ProxyDefinitions xmlns:proxy="http://ns.oracle.com/ip/af/ftp/proxy">
<Proxy key="http" description="http" type="http">
<Step command="USER" args="$remote.user" />
<Step command="PASS" args="$remote.pass" />
</Proxy>
</proxy:ProxyDefinitions>
```

4.4.5.2 Configuring for SFTP Mode

For running the Oracle FTP Adapter in SFTP mode, you must specify the value of certain properties in the Oracle FTP Adapter deployment descriptor. Table 4–17 lists the properties that you must modify.

Table 4–17 SFTP Mode Properties

Property	Description
`host`	The remote FTP server name.
`port`	The FTP control port number.
`username`	The SFTP user name.
`password`	The SFTP password.
`proxyHost`	The proxy server host name.
`proxyPort`	The proxy port number.
`proxyUsername`	The proxy user name.
`proxyPassword`	The proxy password.
`useSftp`	Specify `true` for SFTP mode. This value is required to use the SFTP feature.
`authenticationType`	Specify either `PASSWORD` or `PUBLICKEY.PASSWORD` See Section 4.4.4.3, "Set Up Oracle FTP Adapter for SFTP"
`transportProvider`	Specify `http` as value. Only HTTP transport provider is supported.

A sample list of deployment descriptor properties is shown in Table 4–18.

Table 4–18 Sample SFTP Mode Properties and Values

Property	Value
`host`	`my.host.com`
`port`	`22`

Table 4–18 (Cont.) Sample SFTP Mode Properties and Values

Property	Value
username	user
password	password
proxyHost	proxy.host.com
proxyPort	80
proxyUsername	anonymous
proxyPassword	password
useSFTP	true
authenticationType	password
transportProvider	http

4.5 Oracle File and FTP Adapters Use Cases

This section includes the following Oracle File and FTP Adapters use cases:

- Section 4.5.1, "Oracle File Adapter XML Debatching"
- Section 4.5.2, "Flat Structure for Oracle BPEL PM"
- Section 4.5.3, "Flat Structure for Mediator"
- Section 4.5.4, "Oracle File Adapter Scalable DOM"
- Section 4.5.5, "Oracle File Adapter ChunkedRead"
- Section 4.5.6, "Oracle File Adapter Read File As Attachments"
- Section 4.5.7, "Oracle File Adapter File Listing"
- Section 4.5.8, "Oracle File Adapter Complex Structure"
- Section 4.5.9, "Oracle FTP Adapter Debatching"
- Section 4.5.10, "Oracle FTP Adapter Dynamic Synchronous Read"
- Section 4.5.11, "Copying, Moving, and Deleting Files"

4.5.1 Oracle File Adapter XML Debatching

This is an Oracle File Adapter feature that debatches large XML documents into smaller individual XML fragments.

In this use case, the Debatching XML process uses the Oracle File Adapter to debatch an XML file containing a batch of employees occurring in the XML file as repeating nodes. The Adapter then processes the nodes and writes separate output files to every individual node.

This use case includes the following sections:

- Section 4.5.1.1, "Prerequisites"
- Section 4.5.1.2, "Designing the SOA Composite"
- Section 4.5.1.3, "Creating the Inbound Oracle File Adapter Service"
- Section 4.5.1.4, "Creating the Outbound File Adapter Service"
- Section 4.5.1.5, "Wiring Services and Activities"

- Section 4.5.1.6, "Deploying with JDeveloper"
- Section 4.5.1.7, "Monitoring Using Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console)"

4.5.1.1 Prerequisites

To perform debatching, you require the following files from the `artifacts.zip` file contained in the `Adapters-102FileAdapterXMLDebatching` sample:

- `artifacts/input/emps.xml`
- `artifacts/schemas/employees.xsd`

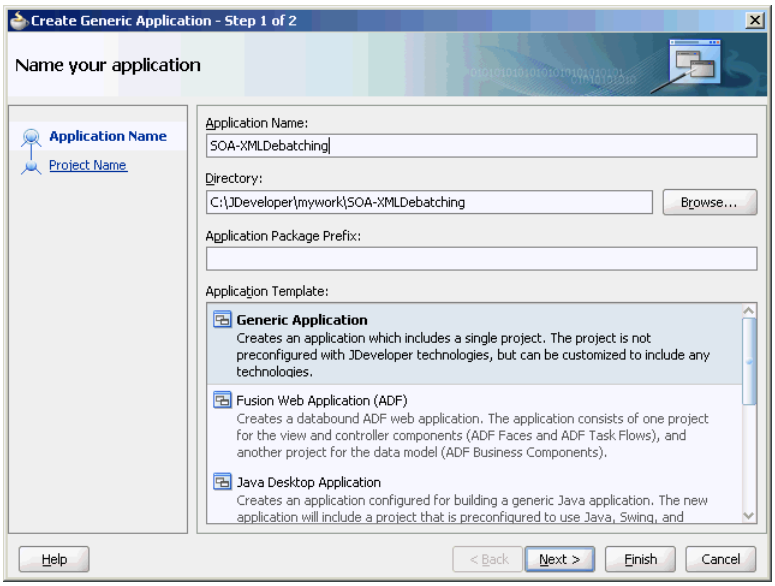
You can obtain the `Adapters-102FileAdapterXMLDebatching` sample by accessing the Oracle SOA Sample Code site.

4.5.1.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

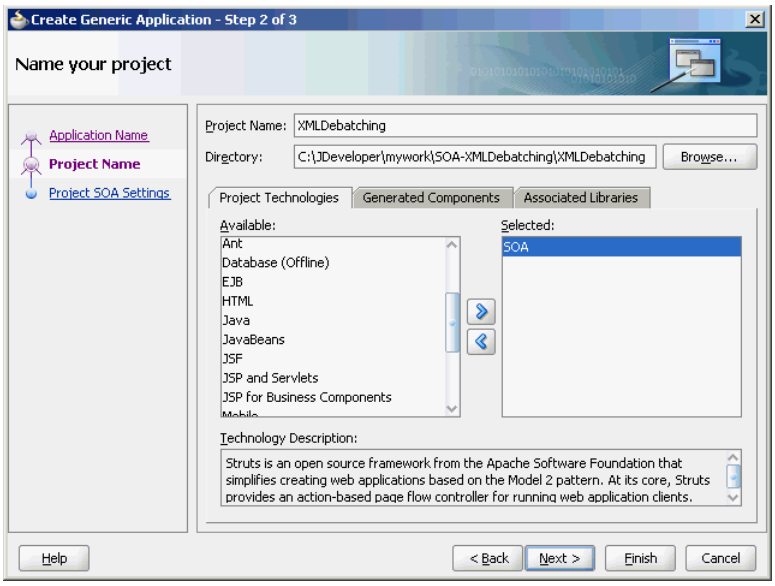
1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `SOA-XMLDebatching` in the **Application Name** field, as shown in Figure 4–48, and click **Next**. The Create Generic Application - Name your project page is displayed.

Figure 4–48 The Generic Create Application - Name your application Page



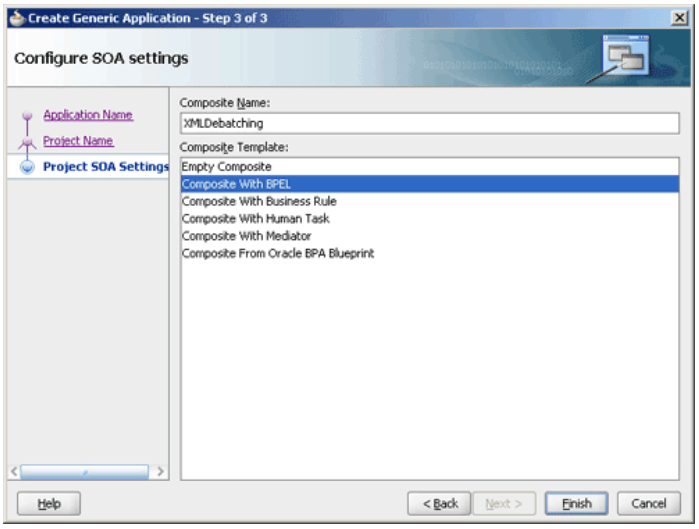
3. Enter `XMLDebatching` in the **Project Name** field.
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list, as shown in Figure 4–49.

Figure 4–49 The Create Generic Application - Name your project Page



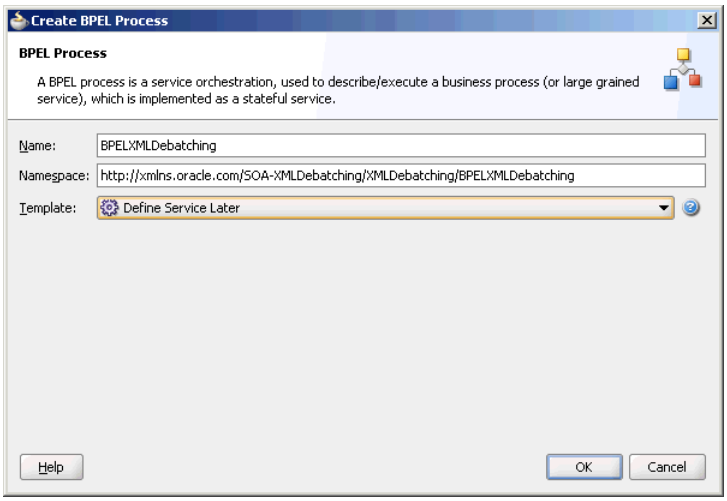
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, as shown in Figure 4–50, and click **Finish**. The Create BPEL Process - BPEL Process page is displayed.

Figure 4–50 The Create Generic Application - Configure SOA settings Page



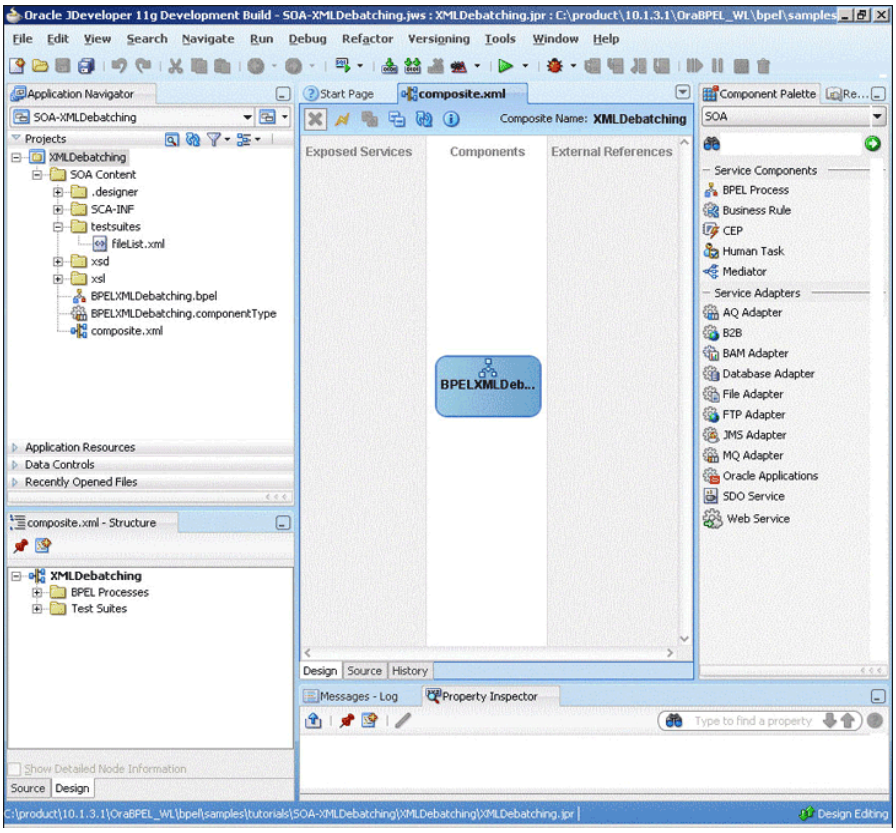
7. Enter `BPELXMLDebatching` in the **Name** field, select **Define Service Later** from the Template box, as shown in Figure 4–51.

Figure 4–51 The Create BPEL Process - BPEL Process Page



8. Click **OK**. The SOA-XMLDebatching application and the XMLDebatching project appear in the design area, as shown in Figure 4–52.

Figure 4–52 The JDeveloper - Composite.xml



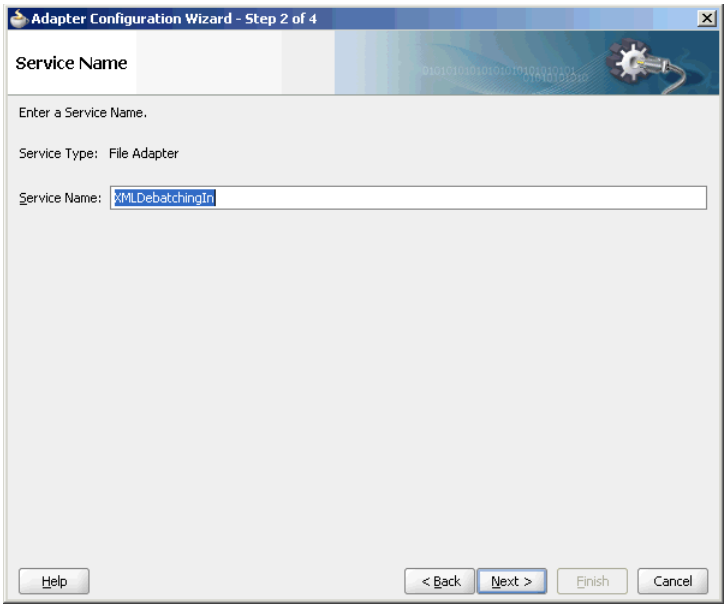
9. Copy the `employees.xsd` file to the xsd directory in your project (see Section 4.5.1.1, "Prerequisites" for the location of this file).

4.5.1.3 Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory:

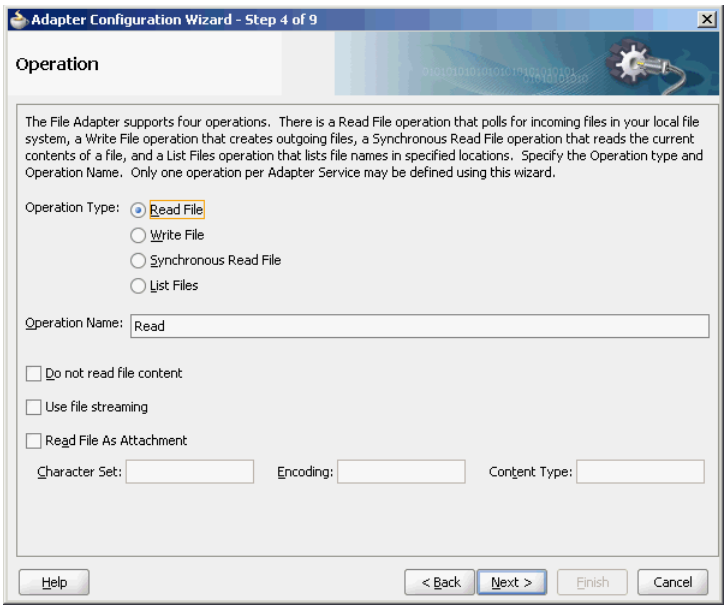
1. Drag and drop the Oracle File Adapter from the Component Palette to the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `XMLDebatchingIn` in the **Service Name** field and, as shown in Figure 4–53.

Figure 4–53 The Adapter Configuration Wizard - Service Name Page



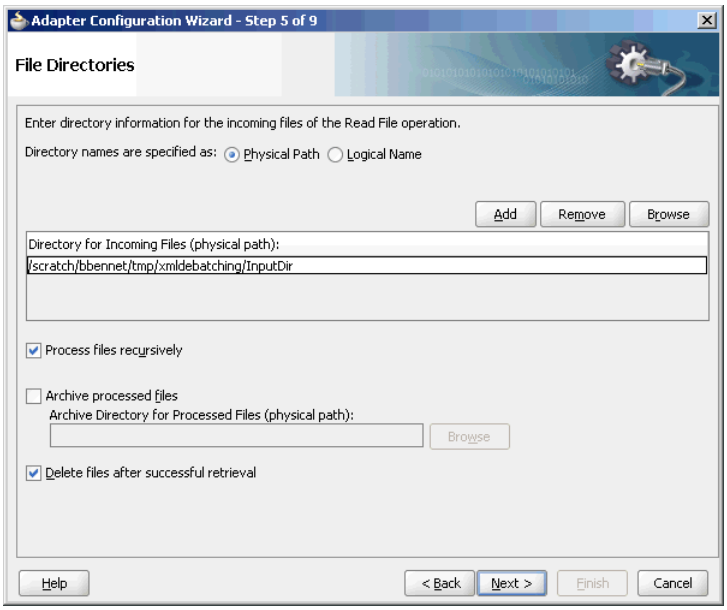
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Read File**, as shown in Figure 4–54, and click **Next**. The File Directories page is displayed.

Figure 4–54 The Adapter Configuration Wizard Operation Page



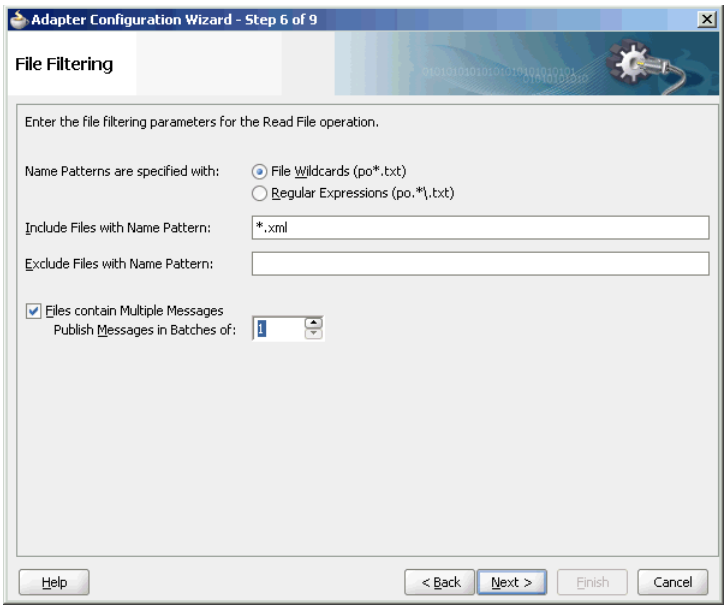
7. Enter the physical path for the input directory, as shown in Figure 4–55. The File Filtering page is displayed.

Figure 4–55 The Adapter Configuration Wizard - File Directories Page



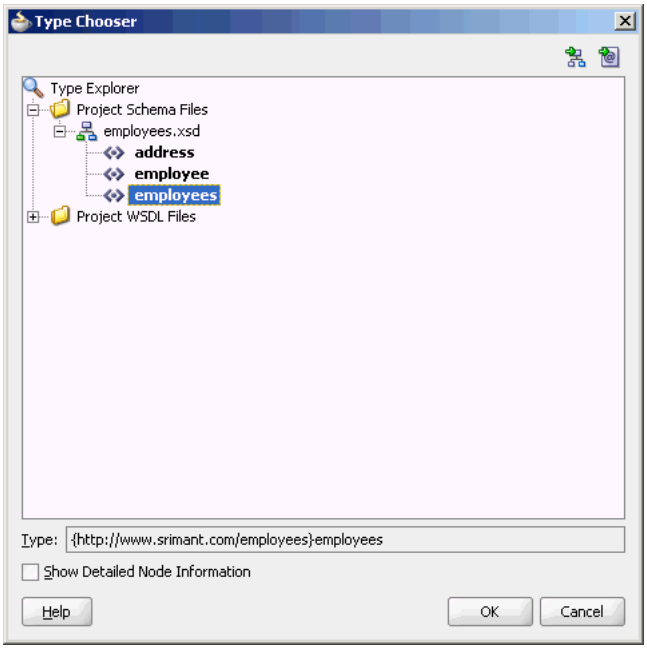
8. Enter `*.xml` in the **Include Files With Name Pattern** field, select **Files Contain Multiple Messages** check box, specify `1` as the value for Publish Messages in Batches Of box, as shown in Figure 4–56.

Figure 4–56 The Adapter Configuration Wizard File Filtering Page



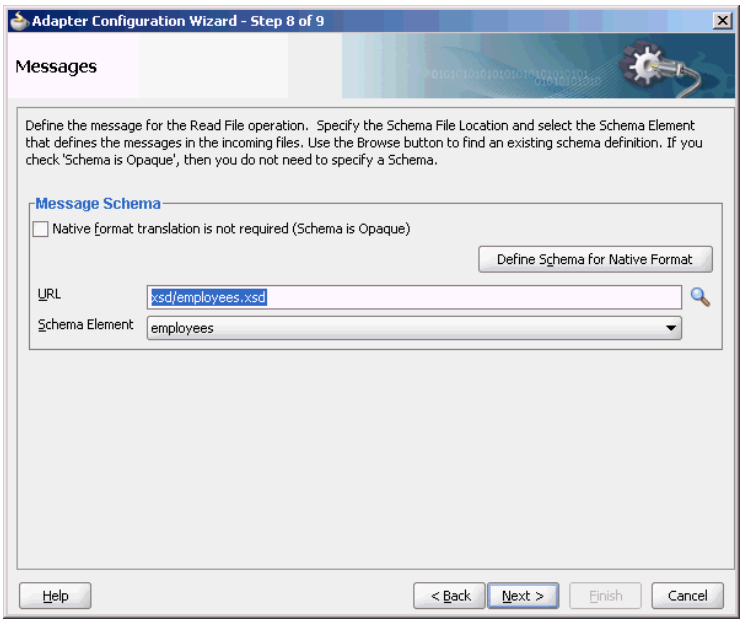
9. Click **Next**. The File Polling page is displayed.
10. Click **Next**. The Messages page is displayed.
11. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
12. Click **Project Schema Files**, **employees.xsd**, and **employees**, as shown in Figure 4–57.

Figure 4–57 The Type Chooser Dialog



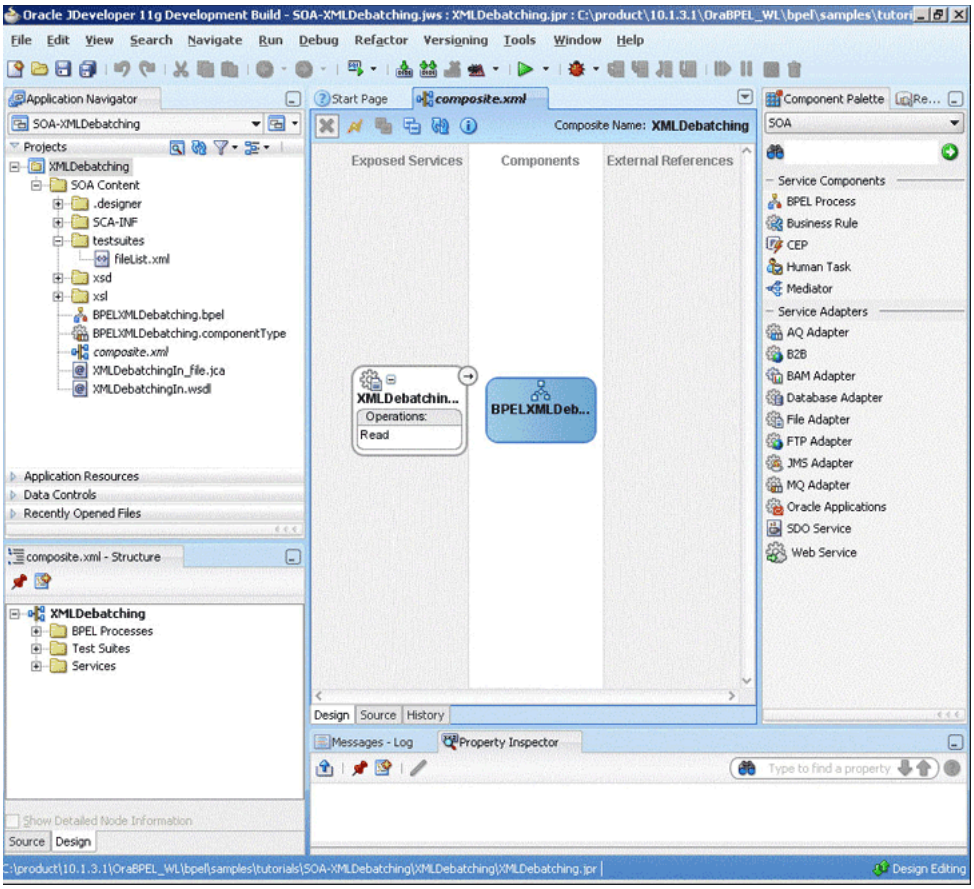
13. Click **OK**. The URL field in the Messages page is populated with the **employees.xsd** file, as shown in Figure 4–58.

Figure 4–58 The Adapter Configuration Wizard File Messages Page



14. Click **Next**. The Finish page is displayed.
15. Click **Finish**. The inbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–59.

Figure 4–59 The JDeveloper - Composite.xml

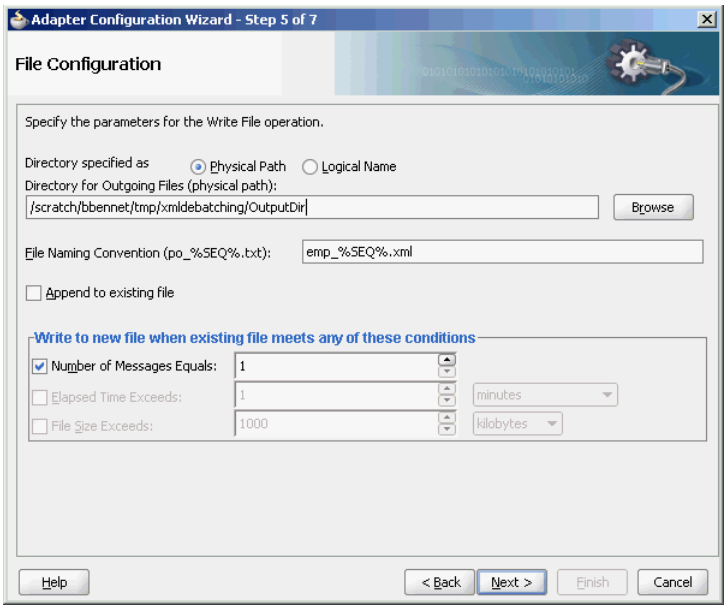


4.5.1.4 Creating the Outbound File Adapter Service

Perform the following steps to create an outbound file adapter service to write the file from a local directory to the FTP server:

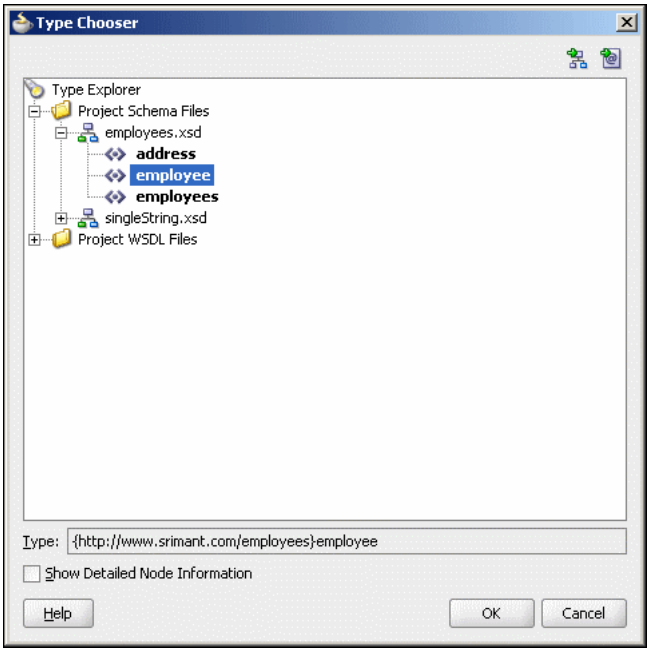
1. Drag and drop the File Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `XMLOut` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, and click **Next**. The File Configuration page is displayed.
7. Enter the physical path for the output directory and enter `emp_%SEQ%.xml` in the **File Naming Convention (po_%SEQ%.txt)** field, as shown in Figure 4–60.
8. Select **Number of Messages Equals** option, if not selected. The default value is `1`.

Figure 4–60 The Adapter Configuration Wizard - File Configuration Page



9. Click **Next**. The Messages page is displayed.
10. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
11. Click **Project Schema Files**, **employees.xsd**, and **employee**, as shown in Figure 4–61.

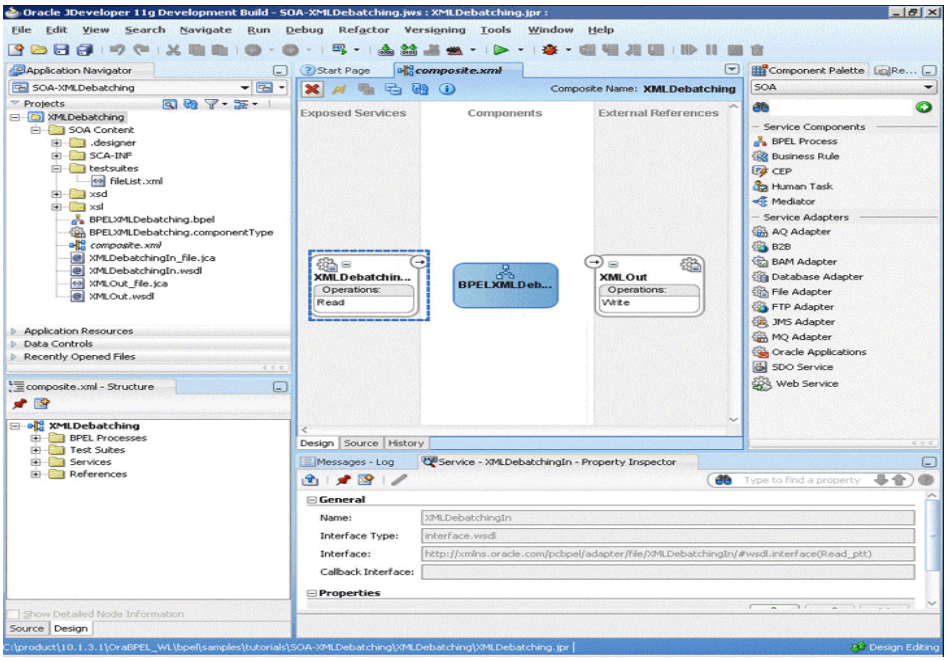
Figure 4–61 The Type Chooser Dialog



12. Click **OK**. The URL field in the Messages page is populated with the employees.xsd file, as shown in Figure 4–58.
13. Click **Next**. The Finish page is displayed.

14. Click **Finish**. The outbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–62.

Figure 4–62 The JDeveloper - Composite.xml



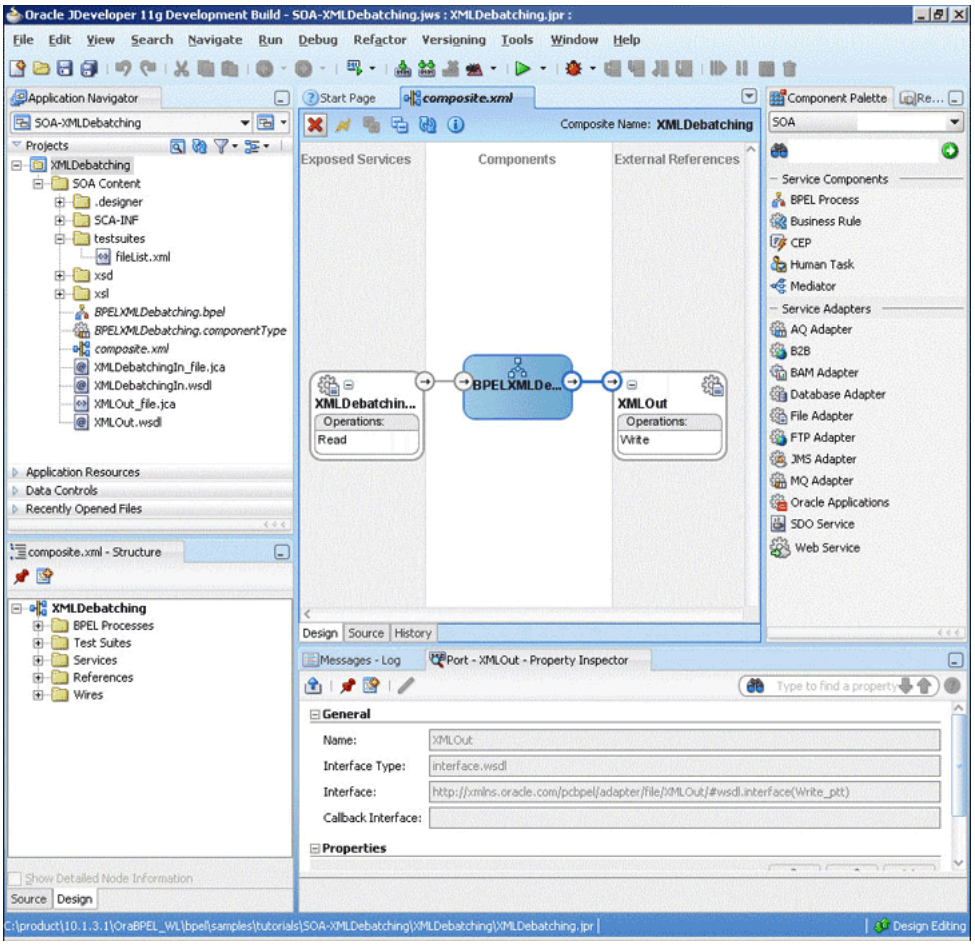
4.5.1.5 Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the XMLDebatchingIn in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the XMLOut in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 4–63.

Figure 4–63 The JDeveloper - Composite.xml

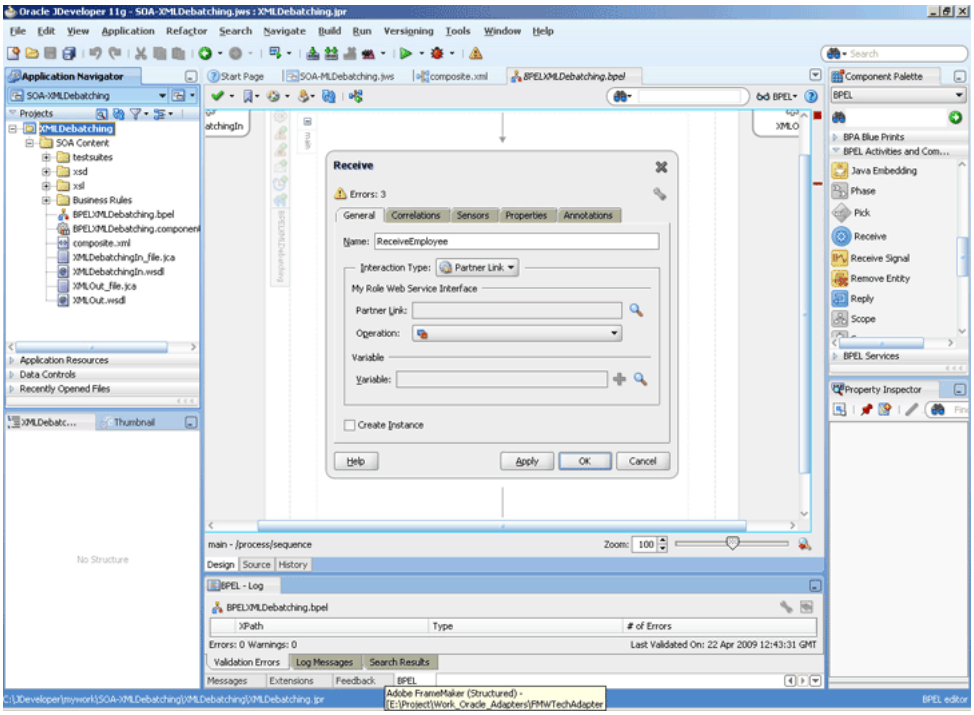


3. Click **File**, **Save All**.

Add a Receive Activity

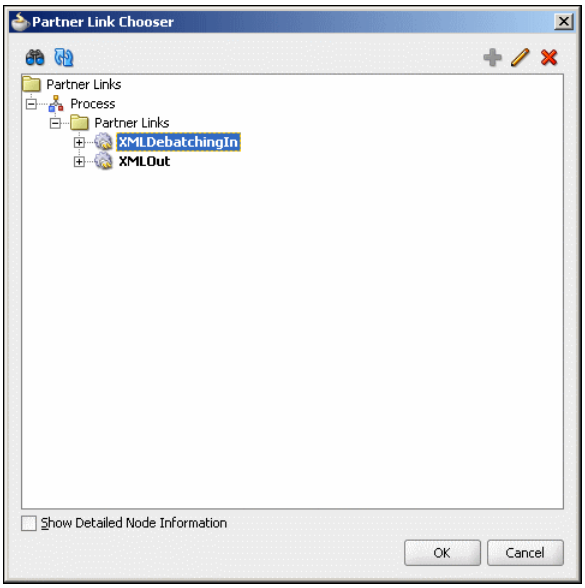
1. Double-click **BPELXMLDebatching**. The BPELXMLDebatching.bpel page is displayed.
2. Drag and drop a **Receive** activity from the Component Palette to the design area.
3. Double-click the **Receive** activity. The Receive dialog is displayed.
4. Enter `ReceiveEmployee` in the **Name** field, as shown in Figure 4–64.

Figure 4–64 The JDeveloper - BPELXMLDebatching.bpel



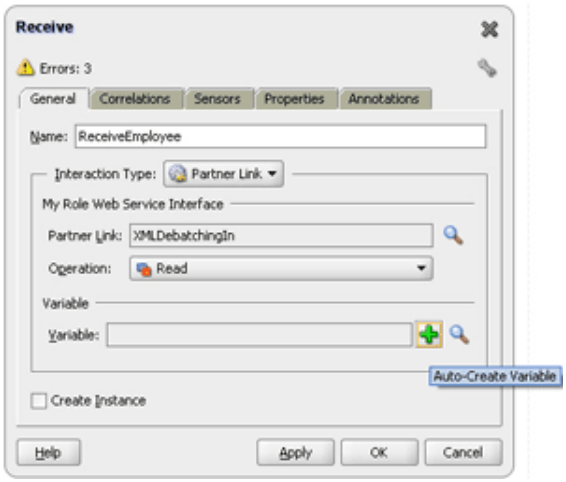
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
6. Select **XMLDebatchingIn**, as shown in Figure 4–65, and click **OK**.

Figure 4–65 The Partner Link Chooser Dialog



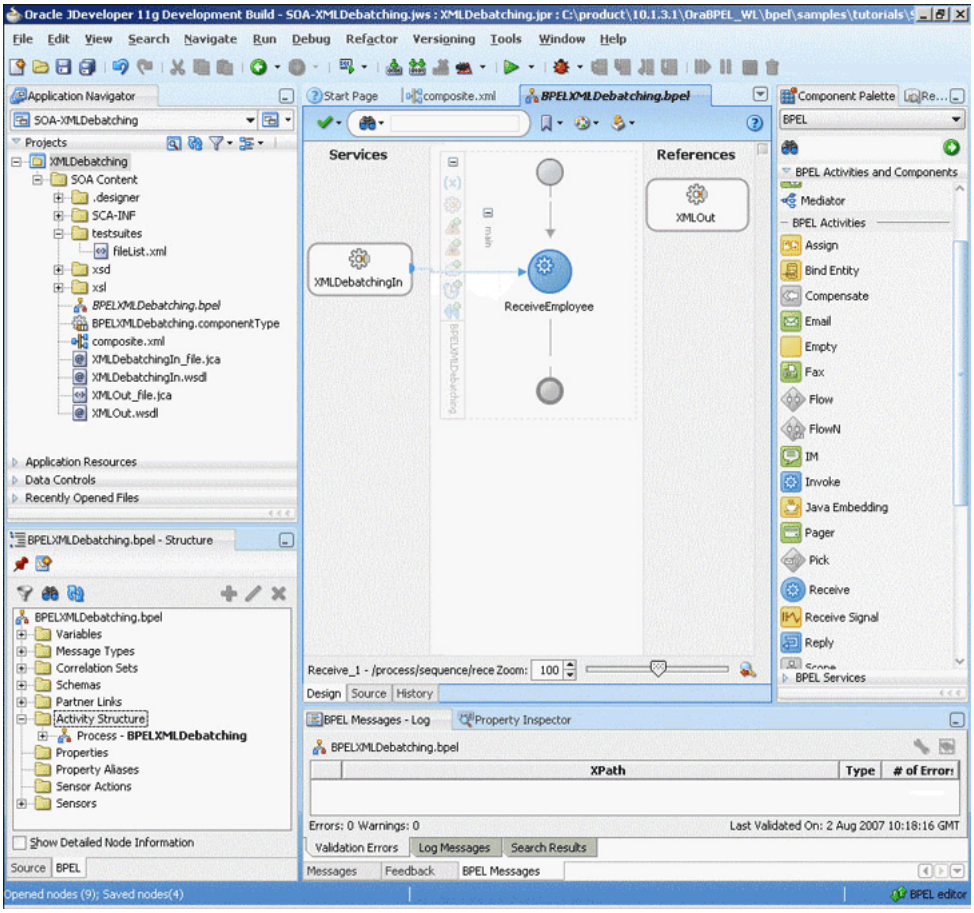
7. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog, as shown in Figure 4–66. The Create Variable dialog is displayed.

Figure 4–66 The Receive Dialog



8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper BPELXMLDebatching.bpel page appears, as shown in Figure 4–67.

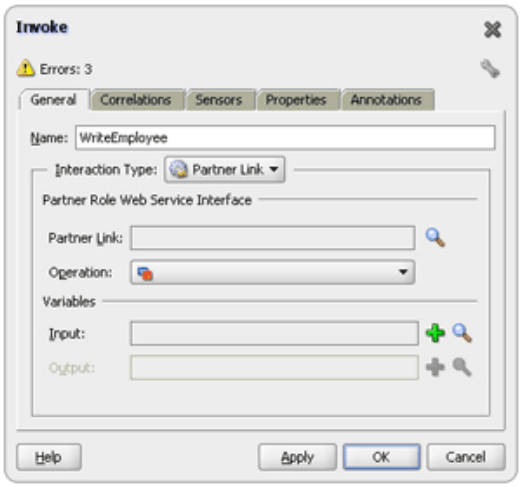
Figure 4–67 The JDeveloper - BPELXMLDebatching.bpel



Add an Invoke Activity

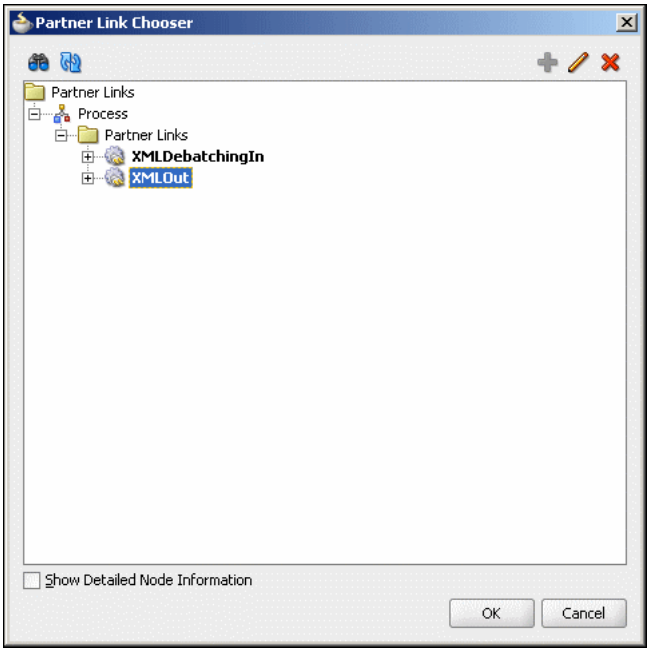
1. Drag and drop an **Invoke** activity from the Component Palette to the design area.
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `WriteEmployee` in the **Name** field, as shown in Figure 4–68.

Figure 4–68 The Invoke Dialog



4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **XMLOut**, as shown in Figure 4–69, and click **OK**.

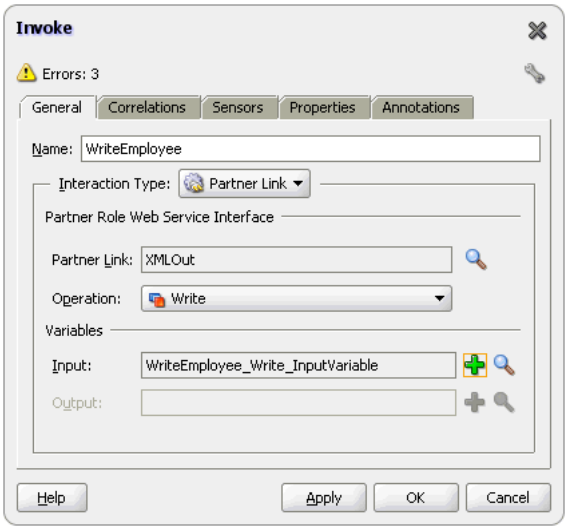
Figure 4–69 The Partner Link Chooser Dialog



6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.

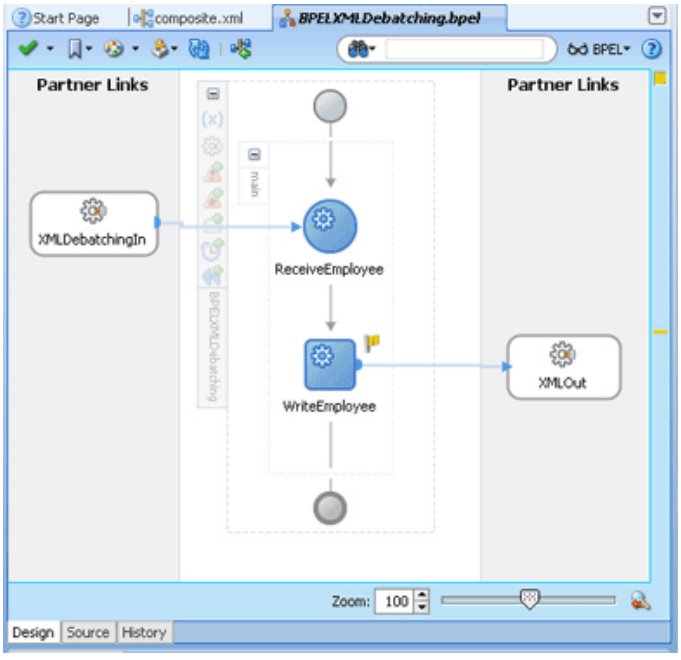
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name. The Invoke dialog is displayed, as shown in Figure 4–70.

Figure 4–70 The Invoke Dialog



8. Click **OK**. The JDeveloper BPELXMLDebatching.bpel page appears, as shown in Figure 4–71.

Figure 4–71 The JDeveloper - BPELXMLDebatching.bpel

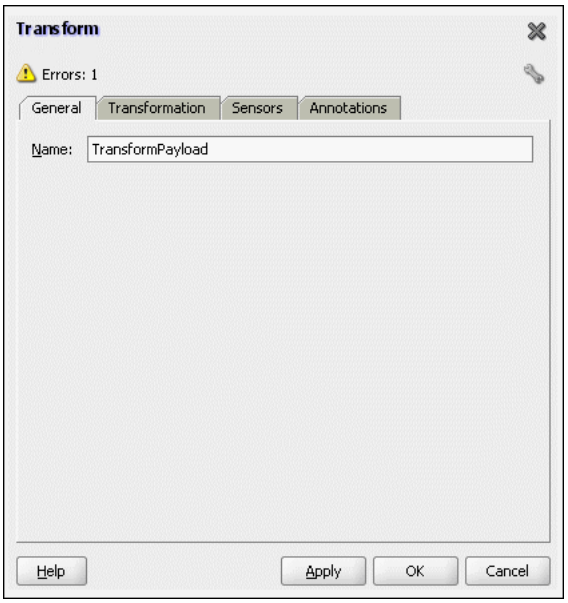


Add a Transform Activity

1. Drag and drop a **Transform** activity from the Component Palette in between the Receive and Invoke activities in the design area.
2. Double-click the **Transform** activity. The Transform dialog is displayed.

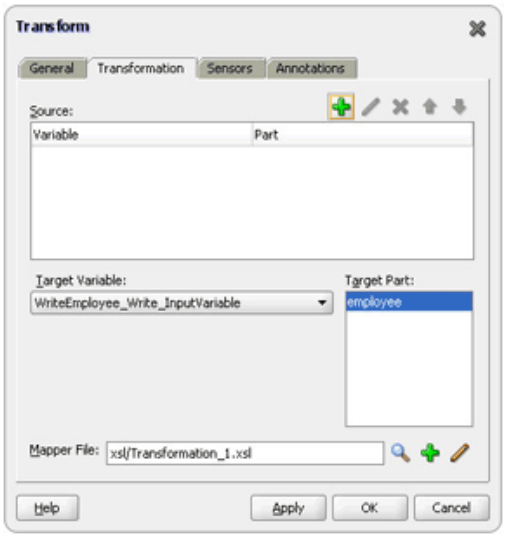
3. Enter `TransformPayload` in the Name field, as shown in Figure 4–72.

Figure 4–72 The Transform Dialog



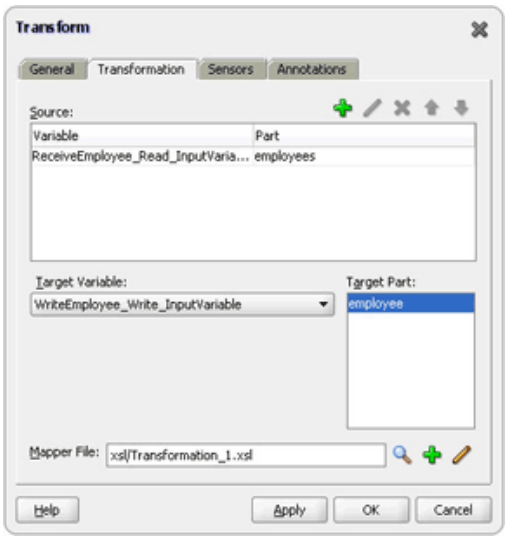
4. Click the **Transformation** tab. The Transform dialog is displayed, as shown in Figure 4–73.

Figure 4–73 The Transform Dialog - Transformation Tab



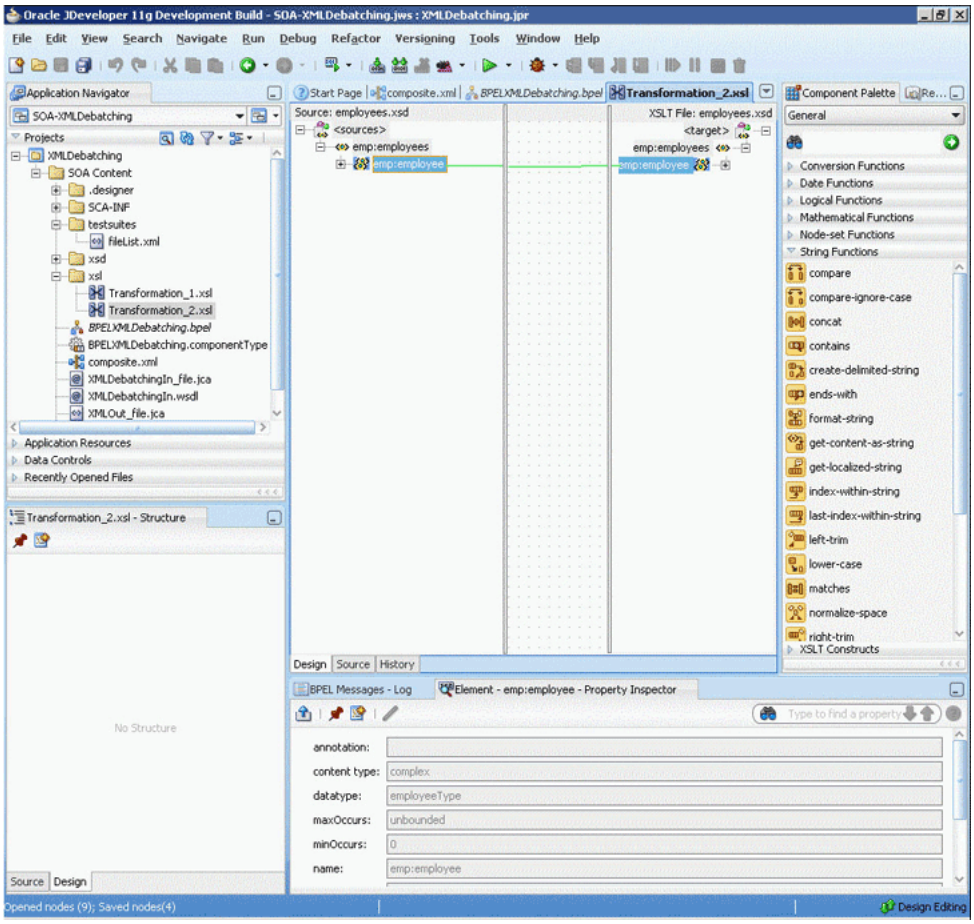
5. Click the **Create...** icon. The Source Variable dialog is displayed.
6. Select **ReceiveEmployee_Read_InputVariable** in the Source Variable box, and select **employees** in the Source Part box, and then click **OK**. The Transform dialog is displayed with the Source and Part selected.
7. Select **WriteEmployee_Write_InputVariable** in the Target Variable list, select **employee** in the Target Part, as shown in Figure 4–74.

Figure 4–74 The Transform Dialog



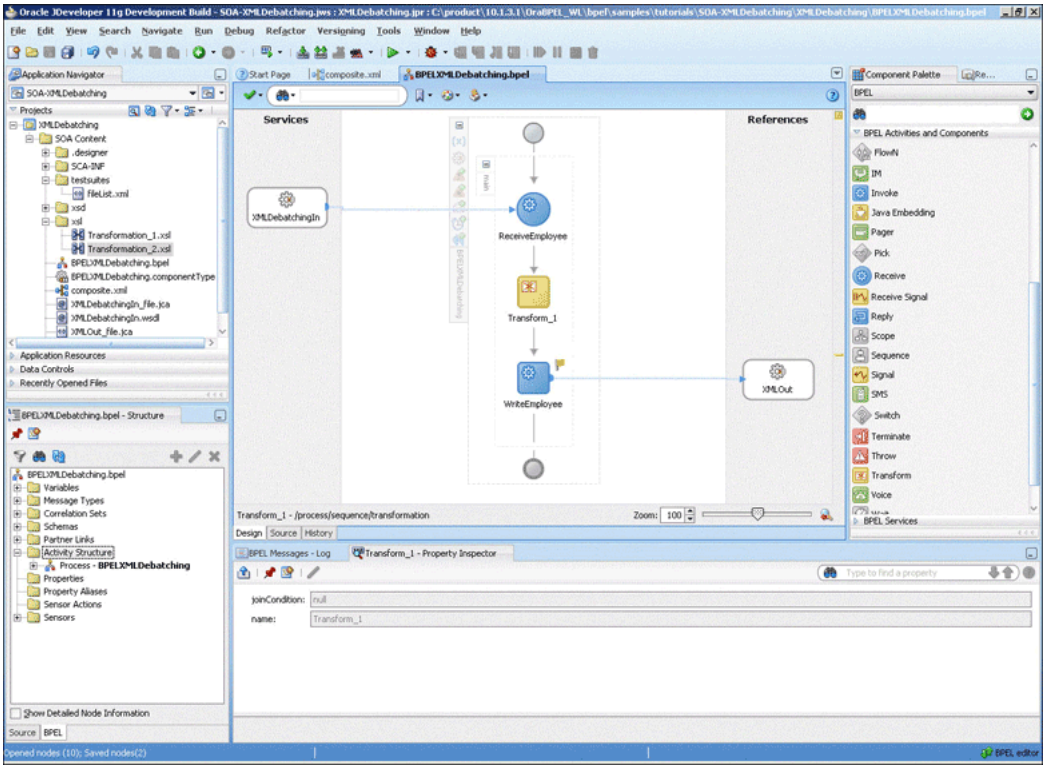
8. Click the **Create Mapping** icon. The XSL Editor page is displayed.
9. Drag employees from sources to employee in the target, as shown in Figure 4–75. The Auto Map Preferences dialog is displayed.

Figure 4–75 The JDeveloper - Transformation_2.xsl



10. Click **OK**.
11. Click **File**, **Save All**.
12. Close the XSL Editor page. The BPELXMLDebatching.bpel page is displayed, as shown in Figure 4–76.

Figure 4–76 The JDeveloper - XML Debatching Complete



4.5.1.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters".
2. Deploy the application. For more information, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper"

4.5.1.7 Monitoring Using Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console)

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the **emps.xml** file to the input directory and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The Dashboard is displayed.

Note your Instance ID in the Recent Instances area.

4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The Flow Trace page is displayed.
6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow.
9. Click an activity to display the activity details.

4.5.2 Flat Structure for Oracle BPEL PM

This use case demonstrates how a flat structure business process uses the Oracle File Adapter to process address book entries from a Comma Separated Value (CSV) file. This is then transformed and written to another file in a Fixed Length format.

This use case includes the following sections:

- Section 4.5.2.1, "Prerequisites"
- Section 4.5.2.2, "Designing the SOA Composite"
- Section 4.5.2.3, "Creating the Inbound Oracle File Adapter Service"
- Section 4.5.2.4, "Creating the Outbound Oracle File Adapter Service"
- Section 4.5.2.5, "Wiring Services and Activities"
- Section 4.5.2.6, "Deploying with JDeveloper"
- Section 4.5.2.7, "Monitoring Using Oracle Fusion Middleware Control Console"

4.5.2.1 Prerequisites

To perform the flat structure business process, you require the following files from the `artifacts.zip` file contained in the `Adapters-101FileAdapterFlatStructure` sample:

- `artifacts/input/address-csv.txt`
- `artifacts/schemas/address-csv.xsd`
- `artifacts/schemas/address-fixedLength.xsd`
- `artifacts/xsl/addr1Toaddr2.xsl`

You can obtain the `Adapters-101FileAdapterFlatStructure` sample by accessing the Oracle SOA Sample Code site.

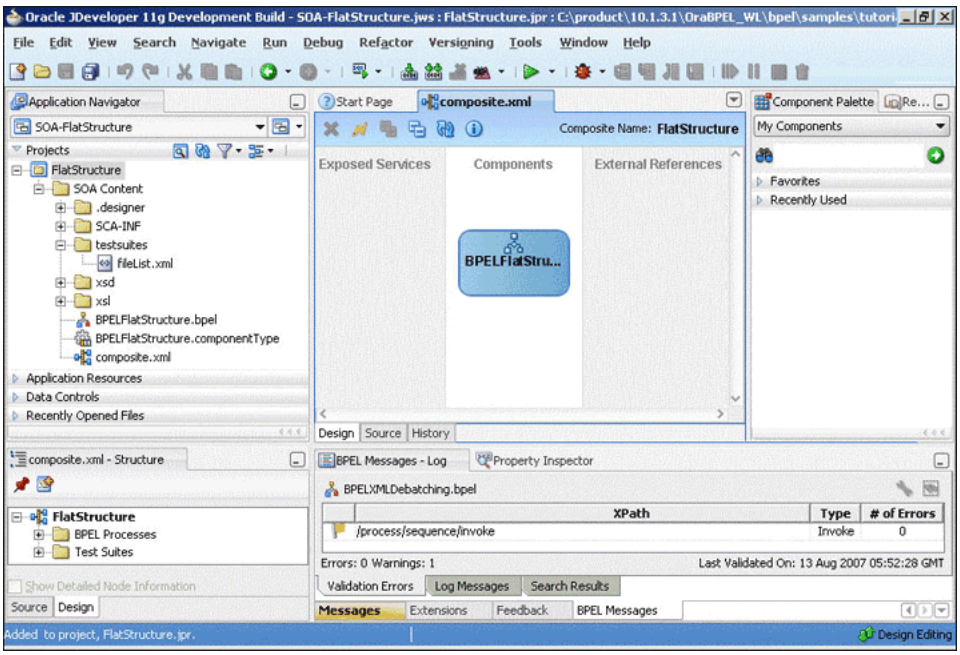
4.5.2.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `SOA-FlatStructure` in the **Application Name** field, and click **OK**. The Create Generic Application - Name your project page is displayed.
3. Enter `FlatStructure` in the **Project Name**.

4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process - BPEL Process page is displayed.
7. Enter `BPELFlatStructure` in the **Name** field, select **Define Service Later** from the Template box.
8. Click **OK**. The SOA-FlatStructure application and the FlatStructure project appear in the design area, as shown in Figure 4–77.

Figure 4–77 The JDeveloper - Composite.xml



9. Copy the **address-csv.xsd** and **address-fixedLength.xsd** files to the schema directory in your project (see Section 4.5.2.1, "Prerequisites" for the location of this file).
10. Copy **addr1Toaddr2.xsl** to the xsl directory of your project (see Section 4.5.2.1, "Prerequisites" for the location of this file).

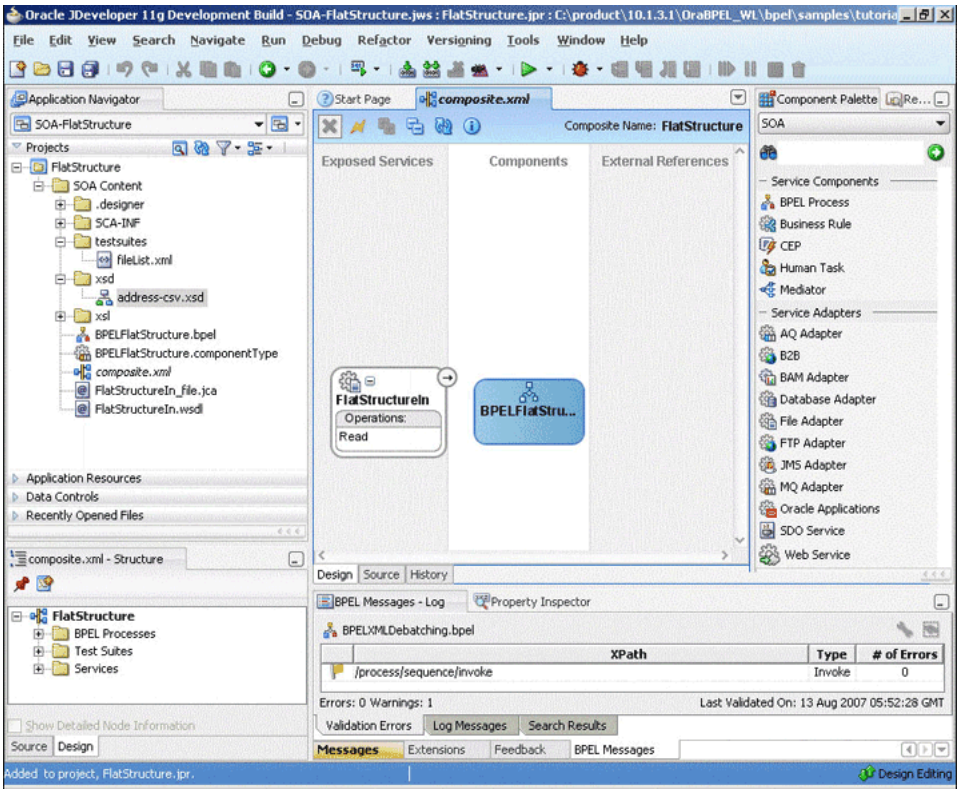
4.5.2.3 Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory:

1. Drag and drop File Adapter from the Component Palette to the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `FlatStructureIn` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Click **Next**. The Operation page is displayed.
6. Select **Read File**, and click **Next**. The File Directories page is displayed.

7. Enter the physical path for the input directory. Check **Archive Processed Files**.
8. Enter the physical path for the archive directory for processed files.
9. Click **Next**. The File Filtering page is displayed.
10. Enter `*.txt` in the **Include Files With Name Pattern** field, click **Next**. The File Polling page is displayed.
11. Click **Next**. The Messages page is displayed.
12. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
13. Click **Project Schema Files**, **address-csv.xsd**, and **Root-Element**.
14. Click **OK**. The URL field in the Messages page is populated with the `address-csv.xsd` file.
15. Click **Next**. The Finish page is displayed.
16. Click **Finish**. The inbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4–78.

Figure 4–78 The JDeveloper - Composite.xml



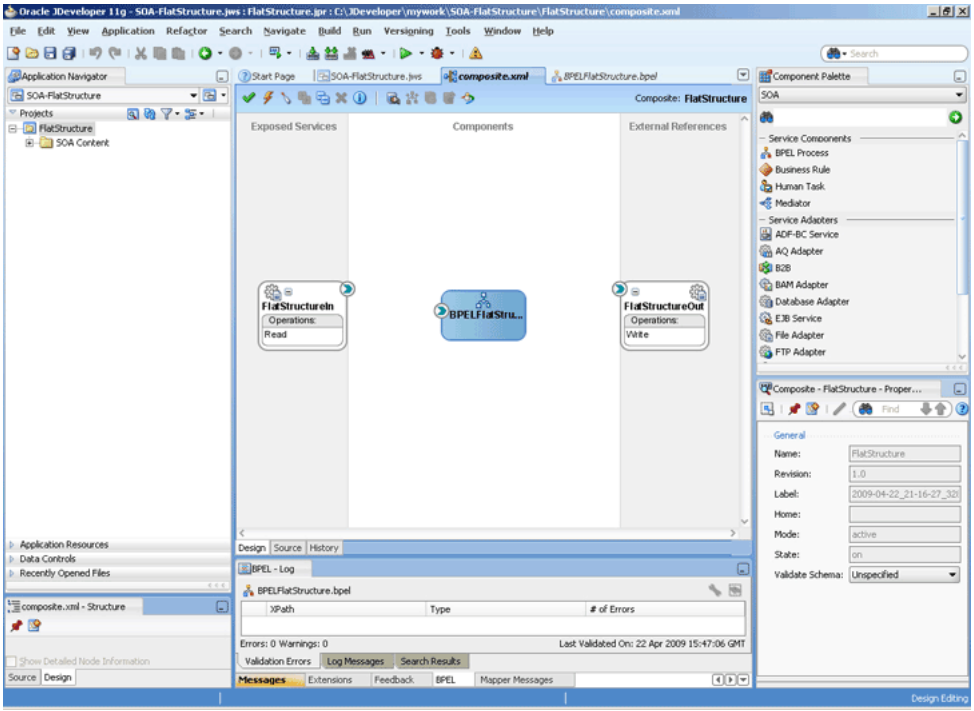
4.5.2.4 Creating the Outbound Oracle File Adapter Service

Perform the following steps to create an outbound Oracle File Adapter service to write the file from a local directory to the FTP server:

1. Drag and drop File Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.

2. Click **Next**. The Service Name page is displayed.
3. Enter `FlatStructureOut` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, and click **Next**. The File Configuration page is displayed.
7. Enter the physical path for the output directory and enter `address_%SEQ%.data` in the **File Naming Convention(po_%SEQ%.txt)** field.
8. Click **Next**. The Messages page is displayed.
9. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
10. Click **Project Schema Files**, **address-fixedLength.xsd**, and **Root-Element**.
11. Click **OK**. The URL field in the Messages page is populated with the address-fixedLength.xsd file.
12. Click **Next**. The Finish page is displayed.
13. Click **Finish**. The outbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–79.

Figure 4–79 The JDeveloper - Composite.xml



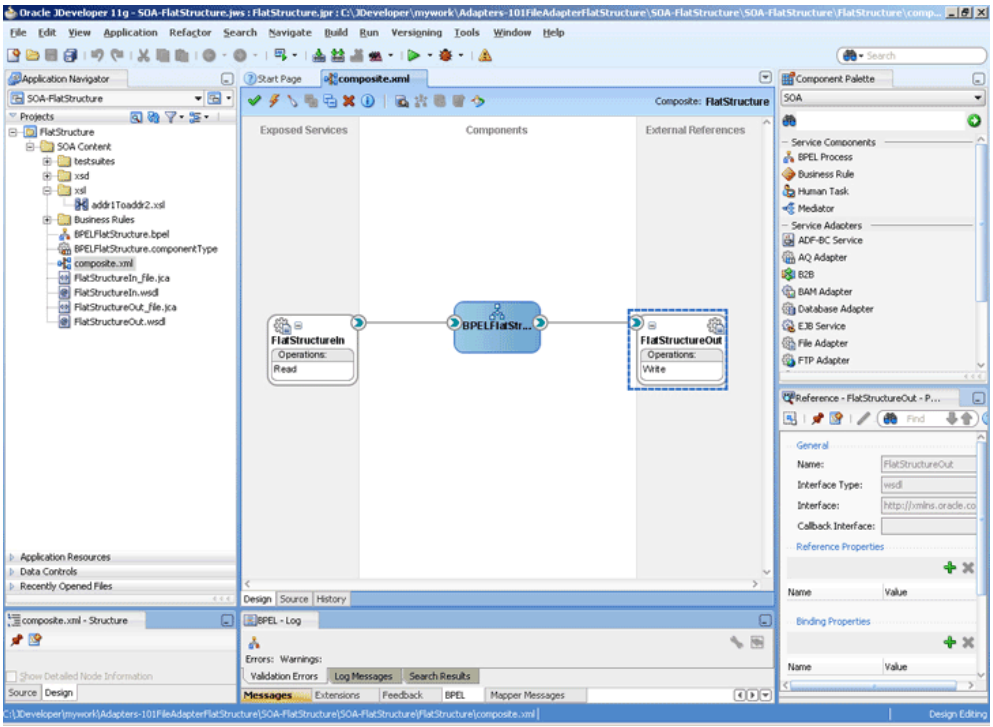
4.5.2.5 Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the FlatStructureIn in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the FlatStructureOut in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 4–80.

Figure 4–80 The JDeveloper - Composite.xml

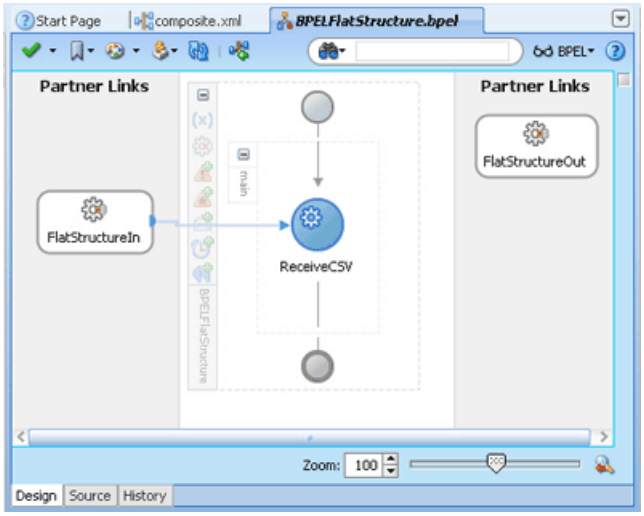


3. Click **File**, **Save All**.

Add a Receive Activity

1. Double-click **BPELFlatStructure**. The BPELFlatStructure.bpel page is displayed.
2. Drag and drop a **Receive** activity from the Component Palette to the design area.
3. Double-click the **Receive** activity. The Receive dialog is displayed.
4. Enter `ReceiveCSV` in the **Name** field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
6. Select **FlatStructureIn**, and click **OK**.
7. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog. The Create Variable dialog is displayed.
8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper BPELFlatStructure.bpel page appears, as shown in Figure 4–81.

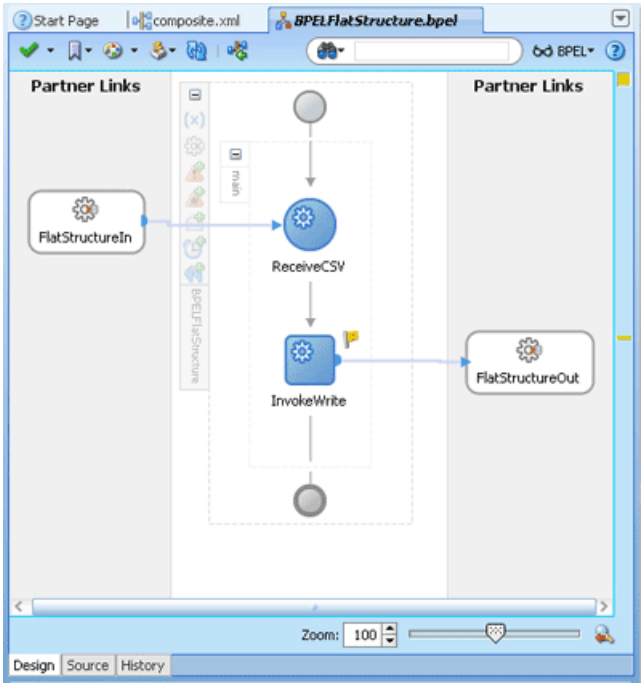
Figure 4–81 The JDeveloper - BPELFlatStructure.bpel



Add an Invoke Activity

1. Drag and drop an **Invoke** activity from the Component Palette to the design area.
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `InvokeWrite` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **FlatStructureOut**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
8. Click **OK**. The JDeveloper BPELFlatStructure.bpel page appears, as shown in Figure 4–82.

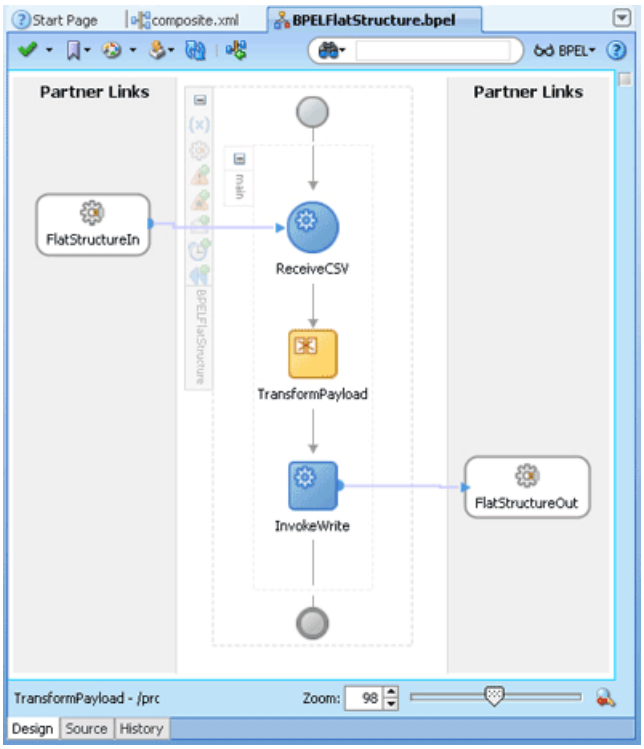
Figure 4–82 The JDeveloper - BPELFlatStructure.bpel



Add a Transform Activity

1. Drag and drop a **Transform** activity from the Component Palette in between the Receive and Invoke activities in the design area.
2. Double-click the **Transform** activity. The Transform dialog is displayed.
3. Enter `TransformPayload` in the **Name** field.
4. Click the **Transformation** tab. The Transform dialog is displayed.
5. Click the **Create...** icon. The Source Variable dialog is displayed.
6. Select **ReceiveCSV_Read_InputVariable** in the Source Variable box, and select **body** in the Source Part box, and then click **OK**. The Transform dialog is displayed with the Source and Part selected.
7. Select **InvokeWrite_Write_InputVariable** in the Target Variable list, select **body** in the Target Part.
8. Click the **Browse** button at the end of the Mapper File field and select **addr1Toaddr2.xsl** file from the xsl directory in your project.
9. Click **OK**.
10. Click **File**, **Save All**. The BPELFlatStructure.bpel page is displayed, as shown in Figure 4–83.

Figure 4–83 The JDeveloper - BPELFlatStructure.bpel



4.5.2.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application. For more information, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

4.5.2.7 Monitoring Using Oracle Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the address-csv.txt file to the input directory and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the Recent Instances area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The Flow Trace page is displayed.
6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.

8. Click the **Flow** tab to view the process flow. Additionally, click an activity (such as invoke, receive) to view the details of an activity.
9. Click **ReceiveCSV** to display the activity details.

4.5.3 Flat Structure for Mediator

In this use case, Mediator receives the customer data from a file system as a text file, through an inbound Oracle File Adapter service named `ReadFile`. The `ReadFile` adapter service sends the message to a routing service named `ReadFile_RS`. The `ReadFile_RS` sends the message to the outbound adapter service `WriteFTP`. The `WriteFTP` service delivers the message to its associated external application.

This use case includes the following sections:

- Section 4.5.3.1, "Prerequisites"
- Section 4.5.3.2, "Creating a Mediator Application and Project"
- Section 4.5.3.3, "Importing the Schema Definition (.XSD) Files"
- Section 4.5.3.4, "Creating the Inbound Oracle File Adapter Service"
- Section 4.5.3.5, "Creating the Outbound Oracle FTP Adapter Service"
- Section 4.5.3.6, "Wiring Services"
- Section 4.5.3.7, "Creating the Routing Rule"
- Section 4.5.3.8, "Deploying with JDeveloper"
- Section 4.5.3.9, "Run-Time Task"

4.5.3.1 Prerequisites

This example assumes that you are familiar with basic Mediator constructs, such as services, routing service, and JDeveloper environment for creating and deploying Mediator services.

To perform the flat structure for Mediator business process, you require the following files from the `artifacts.zip` file contained in the `Adapters-101FileAdapterFlatStructure` sample:

- `artifacts/schemas/address-csv.xsd`

You can see the `Adapters-101FileAdapterFlatStructure` sample by accessing the Oracle SOA Sample Code site.

4.5.3.2 Creating a Mediator Application and Project

To create an application and a project for the use case, follow these steps:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `FileFTP_RW` in the **Application Name** field and click **Next**. The Create Generic Application - Name your project page is displayed.
3. Enter `FileRead_FTPWrite` in the **Project Name** field.
4. In the Available list in the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Create Generic Application - Configure SOA settings page is displayed.

6. Select **Composite With Mediator** in the Composite Template box.
7. Click **Finish**. The Create Mediator - Mediator Component page is displayed.
8. Enter `FileRead_RS` in the Name field.
9. Select **Define Interface Later** in the Template list, and then click **OK**. The `FileFTP_RW` application and the `FileRead_FTPWrite` project appear in the design area.

4.5.3.3 Importing the Schema Definition (.XSD) Files

Perform the following steps to import the XSD files that define the structure of the messages:

1. Create a `Schema` directory and copy the `address-csv.xsd` file to this directory (see Section 4.5.3.1, "Prerequisites" for the location of this file).
2. In the **Application Navigator**, select **FileRead_FTPWrite**.
3. From the **File** menu, select **Import**. The Import dialog is displayed.
4. From the **Select What You Want to Import** list, select **Web Source,** and then click **OK.** The Web Source dialog is displayed.
5. To the right of the **Copy From** field, click **Browse.** The Choose Directory dialog is displayed.
6. Navigate to the **Schema** directory and click **Select**. The Web Source dialog with the directory is displayed.
7. Click **OK**.

4.5.3.4 Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory

1. Drag a File Adapter service from Components Palette to the design area. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `ReadFile` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Click **Next**. The Operation page is displayed.
6. Select **Read File** and click **Next**. The File Directories page is displayed.
7. Select **Physical Path** option, and click **Browse** and select a polling directory.
8. Click **Next**. The File Filtering page is displayed.
9. Enter `*.txt` in the **Include Files with Name Pattern** field and click **Next**. The File Polling page is displayed.
10. Click **Next**. The Messages page is displayed.
11. Click the **Browse For Schema File** button at the end of the URL field. The Type Chooser dialog is displayed.
12. Select **Project Schema Files**, **address-csv.xsd**, and then **Root-Element**.
13. Click **OK**.
14. Click **Next** in the Messages page. The Finish page is displayed.

15. Click **Finish**. A ReadFile adapter service is created.

4.5.3.5 Creating the Outbound Oracle FTP Adapter Service

Perform the following steps to create an outbound Oracle FTP Adapter service to write the file to an FTP server:

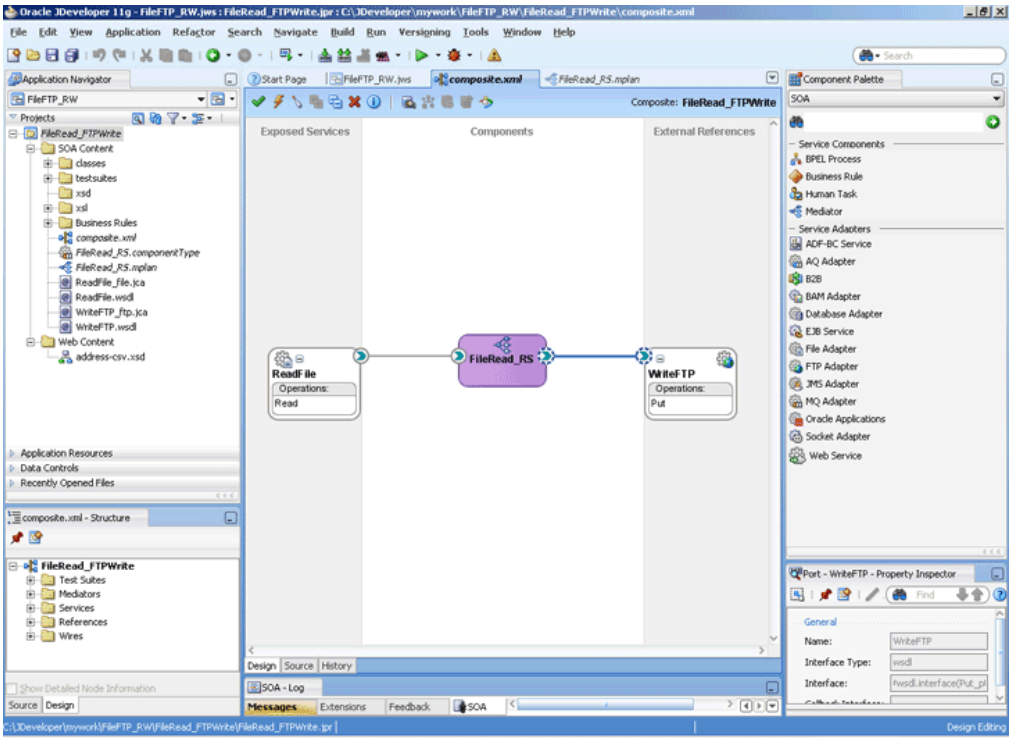
1. Drag an FTP Adapter service from Components Palette to the design area. The Adapter Configuration Wizard Welcome page is displayed.
2. Click Next. The Service Name page is displayed.
3. Enter `WriteFTP` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The FTP Server Connection page is displayed.
6. Specify the JNDI Name of the FTP Server in the **FTP Server JNDI Name** field and click **Next**. The Operation page is displayed.
7. Select **Ascii** option as File Type.
8. Select **Put File** option as the Operation Type and click **Next**. The File Configuration page is displayed.
9. Specify the directory to which file must be written in the **Directory for Outgoing Files (physical path)** field.
10. Specify the naming convention for the output file name in the File Naming Convention field. For example, `po_%SEQ%.txt`.
11. Click **Next**. The Messages page is displayed.
12. Click the **Browse For Schema File** button at the end of the URL field. The Type Chooser dialog is displayed.
13. Select **Project Schema Files**, **address-csv.xsd**, and then **Root-Element**.
14. Click **OK**.
15. Click **Next** in the Messages page. The Finish page is displayed.
16. Click **Finish**. A WriteFTP adapter service is created.

4.5.3.6 Wiring Services

You have to assemble or wire the three components that you have created: Inbound Oracle File Adapter service, Mediator component, Outbound Oracle FTP Adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the ReadFile in the Exposed Services area to the drop zone that appears as a green triangle in the Mediator component in the Components area.
2. Drag the small triangle in the Mediator component in the Components area to the drop zone that appears as a green triangle in the WriteFTP in the External References area. The JDeveloper composite.xml appears, as shown in Figure 4–84.

Figure 4–84 The JDeveloper - Composite.xml

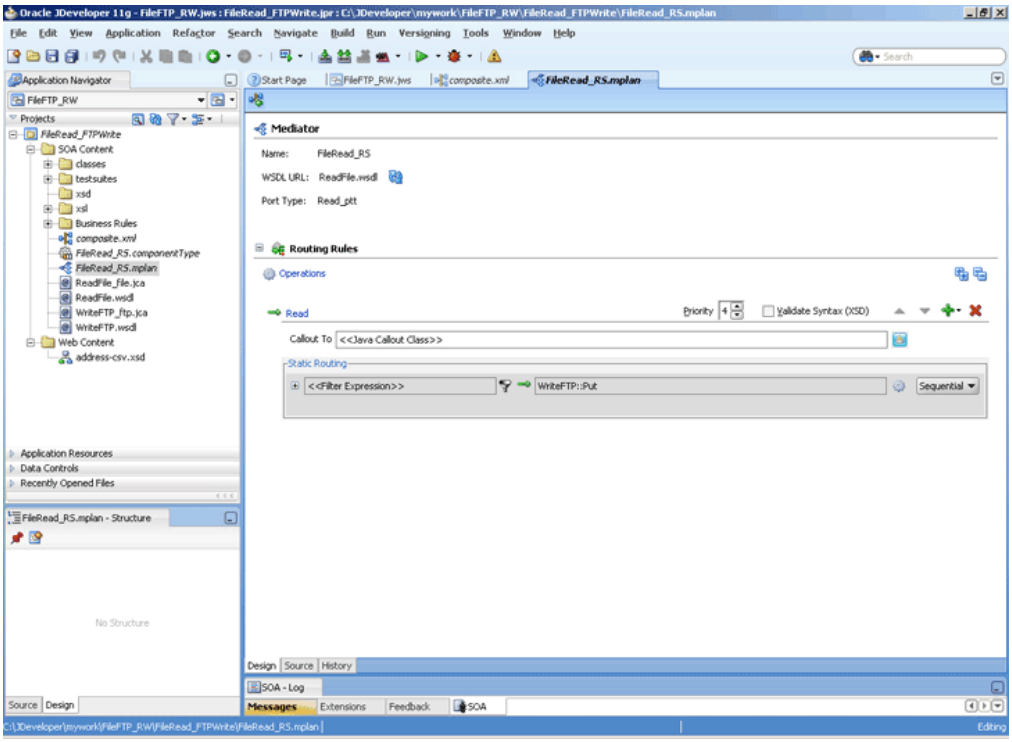


4.5.3.7 Creating the Routing Rule

Perform the following steps to create a routing service:

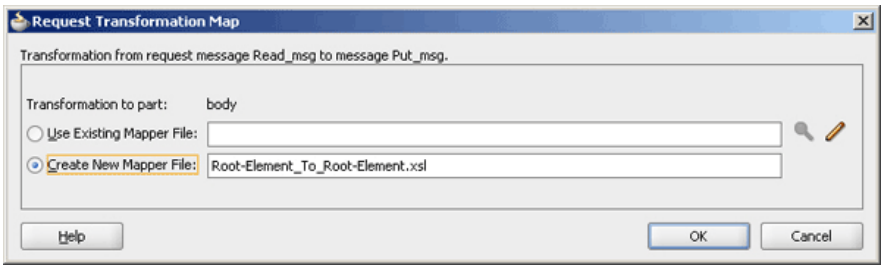
1. Double-click the **ReadFile_RS** routing service. The Read operation is listed in the Operations pane, as shown in Figure 4–85.

Figure 4–85 The JDeveloper - ReadFile_RS Routing Service Page



2. Click the + sign to the left of <<Filter Expression>>to expand the routing rule.
3. Click the icon that appears at the end of the Transform Using field. The Request Transformation Map dialog is displayed, as shown in Figure 4–86.

Figure 4–86 The Request Transformation Map Dialog



4. Select **Create New Mapper File** and click **OK**.

   A Root-Element_To_Root-Element.xsl tab is added to JDeveloper. This tab enables you to graphically create a document transformation file to convert the structure of the file data to a canonical data structure.
5. Drag and drop the **imp1:Address** source element to the **imp1:Address** target element. The Auto Map Preferences dialog is displayed.
6. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names.**
7. Click **OK**.
8. From the **File** menu, click **Save**.

4.5.3.8 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application. For more information, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

4.5.3.9 Run-Time Task

At run time, copy a text file to the polling directory. Once the Oracle File Adapter picks the file,it writes the file to the directory that you specified at design time.

4.5.4 Oracle File Adapter Scalable DOM

This use case demonstrates how a scalable DOM process uses the streaming feature to copy/move huge files from one directory to another.

The streaming option is not supported with DB2 hydration store.

This use case includes the following sections:

- Section 4.5.4.1, "Prerequisites"
- Section 4.5.4.2, "Designing the SOA Composite"
- Section 4.5.4.3, "Creating the Inbound Oracle File Adapter Service"
- Section 4.5.4.4, "Creating the Outbound Oracle File Adapter Service"
- Section 4.5.4.5, "Wiring Services and Activities"
- Section 4.5.4.6, "Deploying with JDeveloper"
- Section 4.5.4.7, "Monitoring Using Fusion Middleware Control Console"

4.5.4.1 Prerequisites

To perform the streaming large payload process, you require the following files from the `artifacts.zip` file contained in the `Adapters-103FileAdapterScalableDOM` sample:

- `artifacts/schemas/address-csv.xsd`
- `artifacts/input/address-csv-large.txt`

You can obtain the `Adapters-103FileAdapterScalableDOM` sample by accessing the Oracle SOA Sample Code site.

4.5.4.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `SOA-ScalableDOM` in the **Application Name** field, and click **Next**. The Create Generic Application - Name your project page is displayed.
3. Enter `ScalableDOM` in the **Project Name** field.

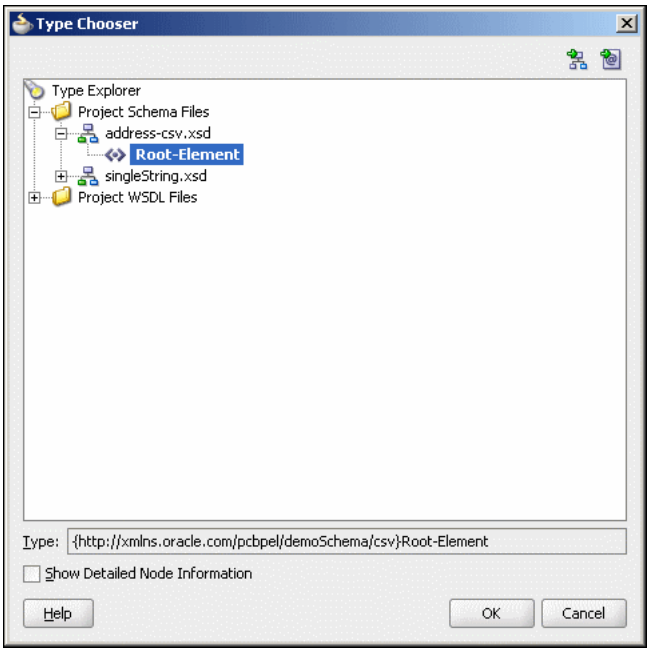
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The SOA-ScalableDOM application and ScalableDOM project appears in the Application Navigator and the Create BPEL Process - BPEL Process page is displayed.
7. Enter `BPELScalableDOM` in the **Name** field, select **Define Service Later** from the Template box.
8. Click **OK**. The SOA-ScalableDOM application and the ScalableDOM project appears in the design area.
9. Copy the **address-csv.xsd** file to the xsd directory in your project (see Section 4.5.4.1, "Prerequisites" for the location of this file).

4.5.4.3 Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory:

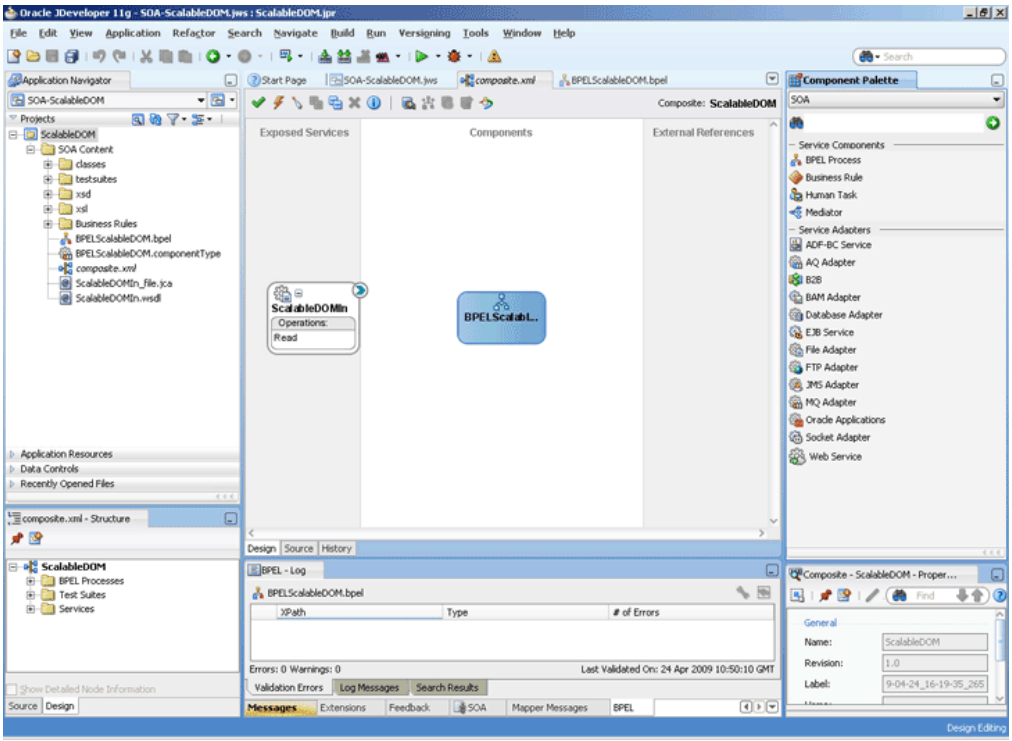
1. Drag and drop File Adapter from the Component Palette to the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `ScalableDOMIn` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Read File**, and check **Use File Streaming**, and click **Next**. The File Directories page is displayed.
7. Enter the physical path for the input directory. The File Filtering page is displayed.
8. Enter `*.txt` in the **Include Files With Name Pattern** field, click **Next**. The File Polling page is displayed.
9. Click **Next**. The Messages page is displayed.
10. Click **Browse For Schema File** that appears after the URL field. The Type Chooser dialog is displayed.
11. Click **Project Schema Files**, **address-csv.xsd**, and **Root-Element**, as shown in Figure 4–87.

Figure 4–87 The Type Chooser Dialog



12. Click **OK**. The URL field in the Messages page is populated with the `address-csv.xsd` file.
13. Click **Next**. The Finish page is displayed.
14. Click **Finish**. The inbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–88.

Figure 4–88 The JDeveloper - Composite.xml

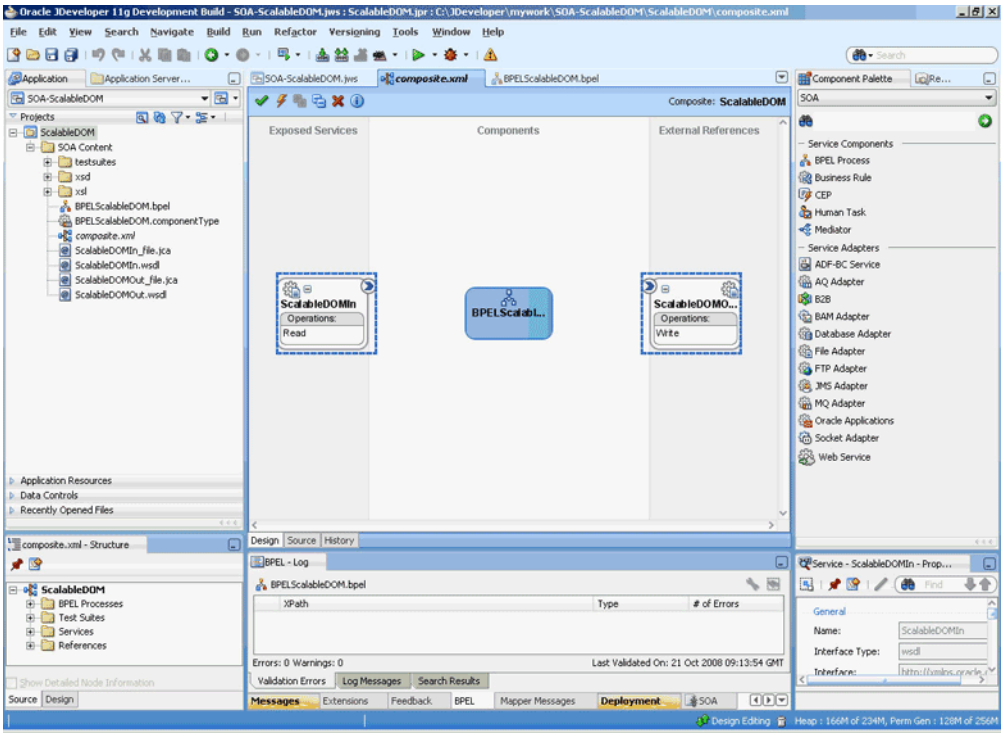


4.5.4.4 Creating the Outbound Oracle File Adapter Service

Perform the following steps to create an outbound Oracle File Adapter service to write the file from a local directory to the FTP server:

1. Drag and drop File Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `ScalableDOMOut` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, and click **Next**. The File Configuration page is displayed.
7. Enter the physical path for the output directory and enter `address-csv_%SEQ%.xml` in the **File Naming Convention (po_%SEQ%.txt)** field.
8. Click **Next**. The Messages page is displayed.
9. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
10. Click **Project Schema Files**, **address-csv.xsd**, and **Root-Element**.
11. Click **OK**. The URL field in the Messages page is populated with the `address-csv.xsd` file.
12. Click **Next**. The Finish page is displayed.
13. Click **Finish**. The outbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–89.

Figure 4–89 The JDeveloper - Composite.xml



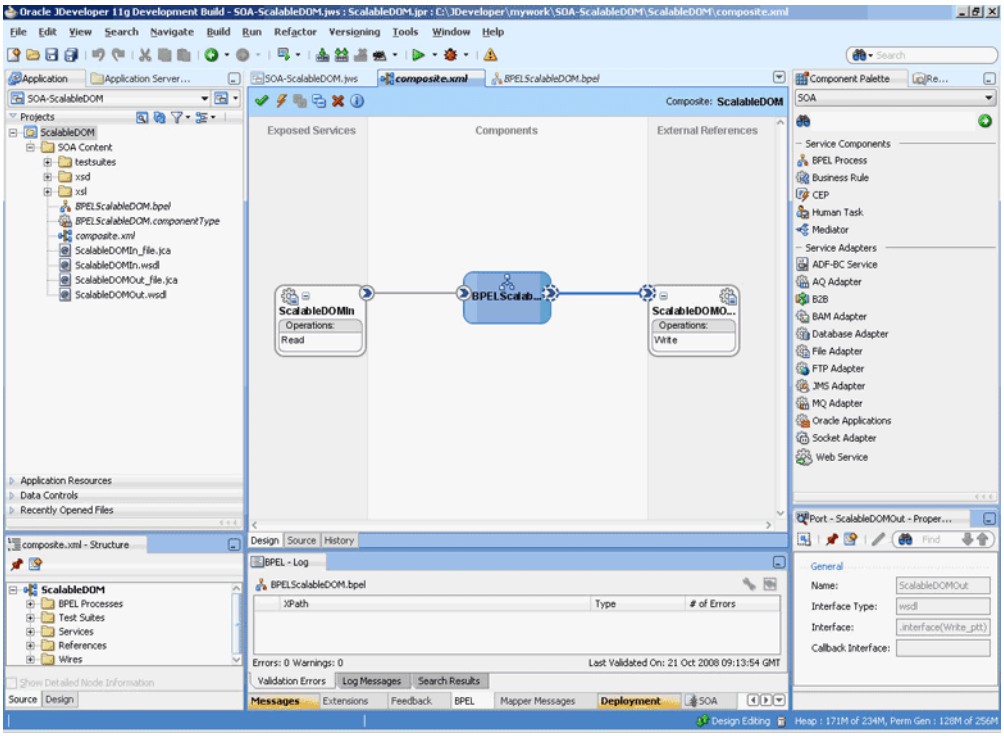
4.5.4.5 Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the ScalableDOMIn in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the ScalableDOMOut in the External References area.

   The JDeveloper composite.xml appears, as shown in Figure 4–90.

Figure 4–90 The JDeveloper - Composite.xml

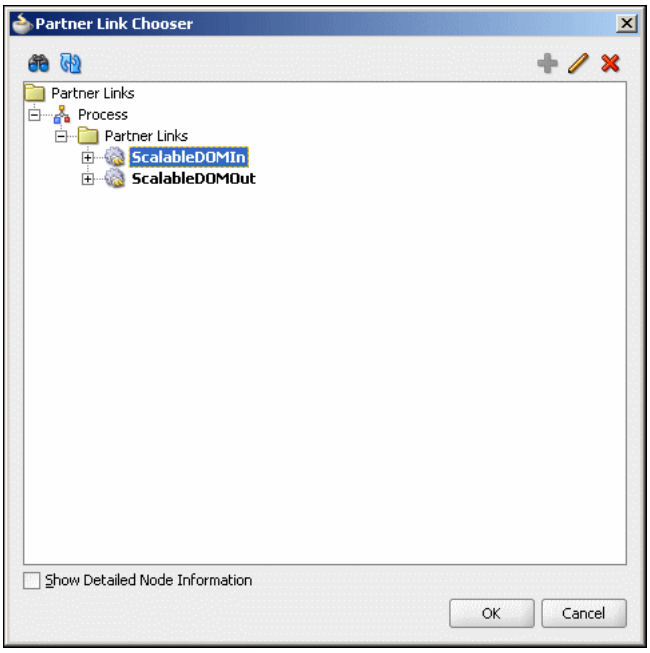


3. Click **File**, **Save All**.

Add a Receive Activity

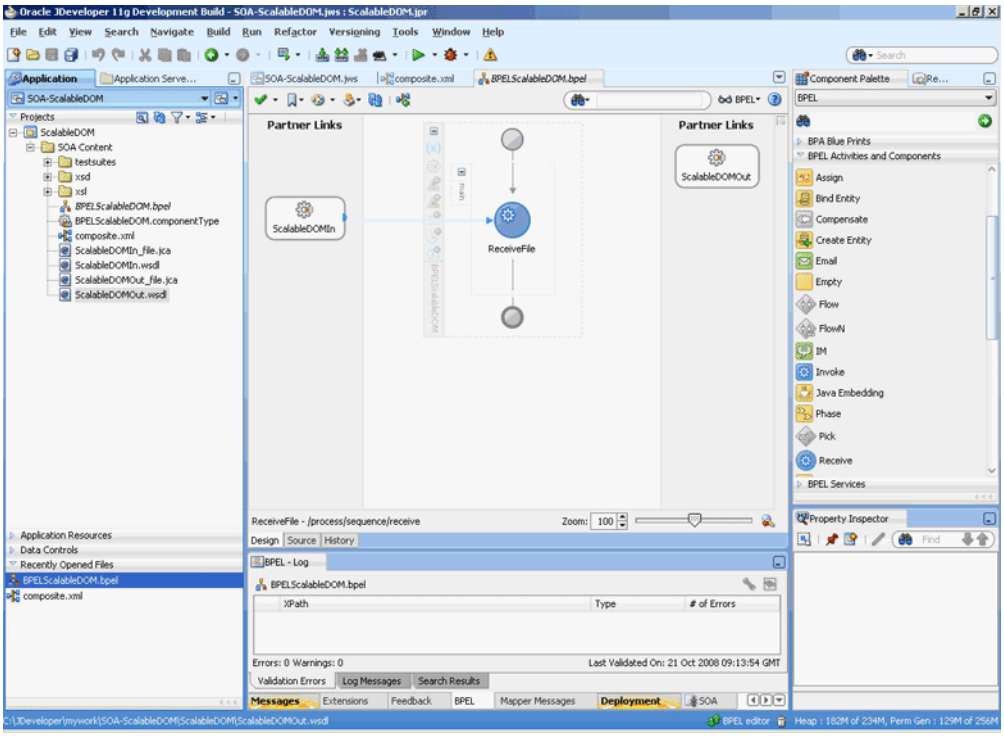
1. Double-click **BPELScalableDOM**. The BPELScalableDOM.bpel page is displayed.
2. Drag and drop a **Receive** activity from the Component Palette to the design area.
3. Double-click the **Receive** activity. The Receive dialog is displayed.
4. Enter `ReceiveFile` in the Name field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
6. Select **ScalableDOMIn**, as shown in Figure 4–91, and click **OK**.

Figure 4–91 The Partner Link Chooser Dialog



7. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog. The Create Variable dialog is displayed.
8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper composite.xml page appears, as shown in Figure 4–92.

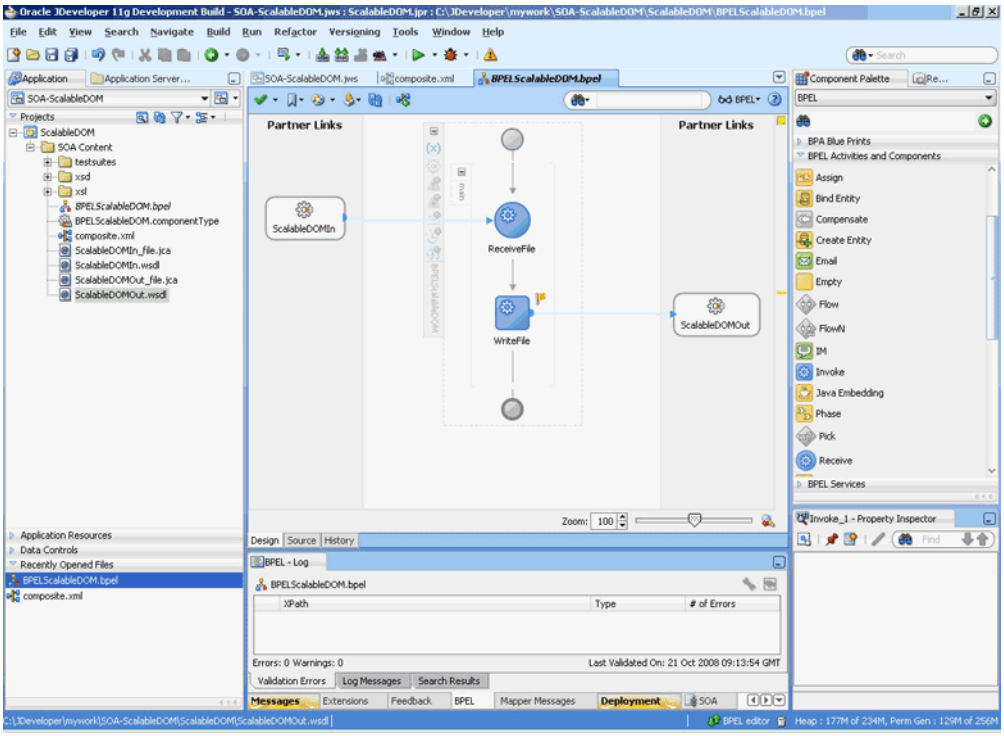
Figure 4–92 The JDeveloper - BPELScalableDOM.bpel



Add an Invoke Activity

1. Drag and drop an **Invoke** activity from the Component Palette to the design area.
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `WriteFile` in the Name field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **ScalableDOMOut**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Input variable field is populated with the default variable name. The Invoke dialog is displayed.
8. Click **OK**. The JDeveloper BPELScalableDOM.bpel page appears, as shown in Figure 4–93.

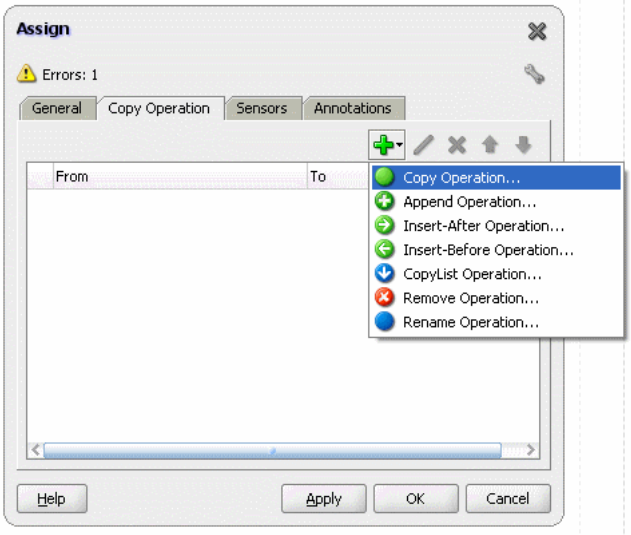
Figure 4–93 The JDeveloper - BPELScalableDOM.bpel Page



Add an Assign Activity

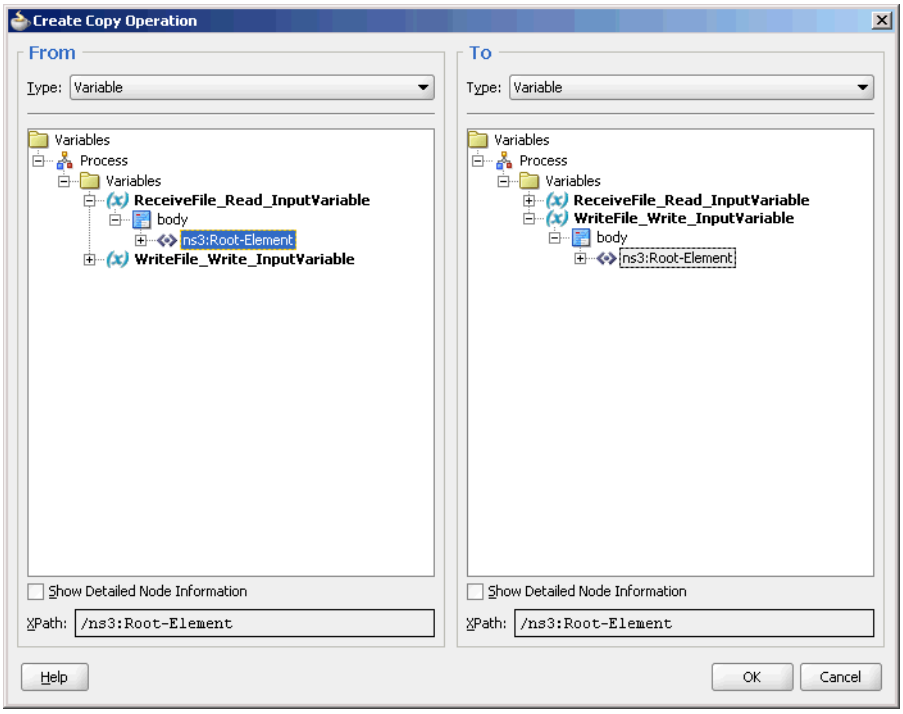
1. Drag and drop an **Assign** activity from the Component Palette in between the Receive and Invoke activities in the design area.
2. Double-click the **Assign** activity. The Assign dialog is displayed.
3. Enter `AssignPayload` in the **Name** field.
4. Click the **Copy Operation** tab. The Assign dialog is displayed, as shown in Figure 4–94.

Figure 4–94 The Assign Dialog - Copy Operation Tab



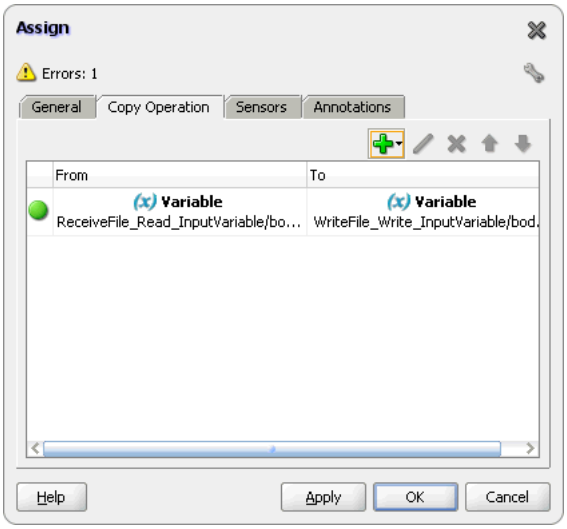
5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
6. Expand the variables in the From and To panes, as shown in Figure 4–95.

Figure 4–95 The Create Copy Operation Dialog



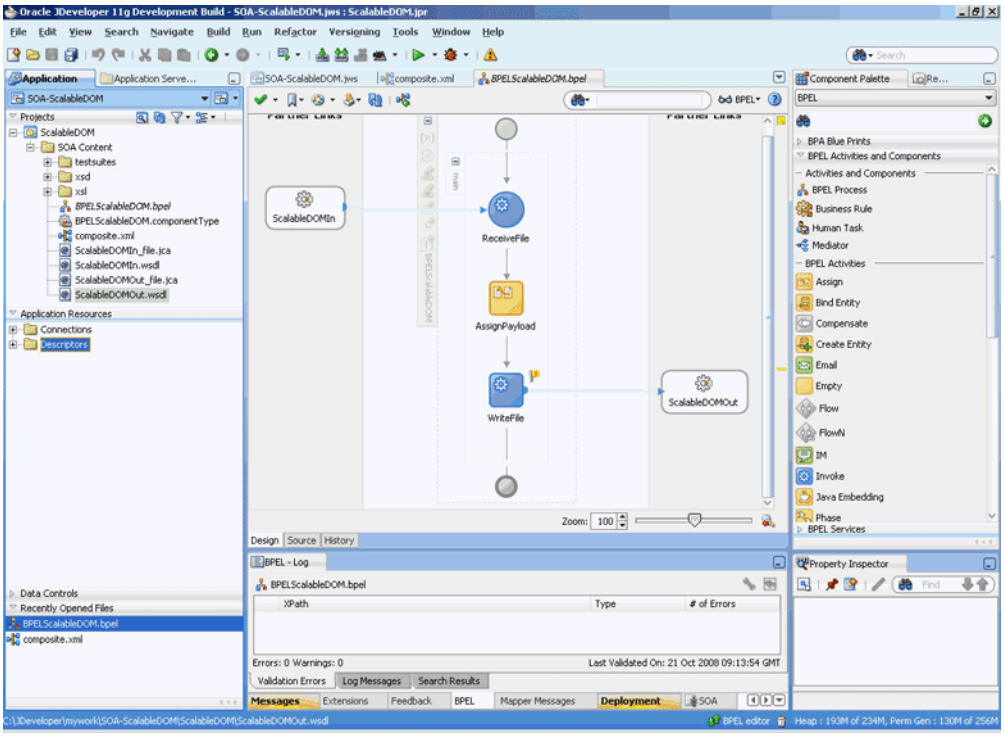
7. Click **OK**. The Assign dialog is displayed, as shown in Figure 4–96.

Figure 4–96 The Assign Dialog



8. Click **OK**, the JDeveloper BPELScalableDOM.bpel page is displayed, as shown in Figure 4–97.

Figure 4–97 The JDeveloper - BPELScalableDOM.bpel



9. Click **File**, **Save All**.

4.5.4.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application. For more information, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

4.5.4.7 Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the `address-csv-large.txt` file to the input directory and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the Recent Instances area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The Flow Trace page is displayed.

6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow. Additionally, click an activity to view the details of an activity.

4.5.5 Oracle File Adapter ChunkedRead

This is an Oracle File Adapter feature that uses an invoke activity within a while loop to process the target file. This feature enables you to process arbitrarily large files.

This use case includes the following sections:

- Section 4.5.5.1, "Prerequisites"
- Section 4.5.5.2, "Designing the SOA Composite"
- Section 4.5.5.3, "Creating the Inbound Oracle File Adapter Service"
- Section 4.5.5.4, "Creating the Outbound Oracle File Adapter Service"
- Section 4.5.5.5, "Wiring Services and Activities"
- Section 4.5.5.6, "Deploying with JDeveloper"
- Section 4.5.5.7, "Monitoring Using Fusion Middleware Control Console"

4.5.5.1 Prerequisites

To perform the Oracle File Adapter ChunkRead, you require the following files from the `artifacts.zip` file contained in the `Adapters-106FileAdapterChunkedRead` sample:

- `artifacts/schemas/address-csv.xsd`
- `artifacts/schemas/address-fixedLength.xsd`
- `artifacts/xsl/addr1Toaddr2.xsl`
- `artifacts/input/address-csv.txt`

You can obtain the `Adapters-106FileAdapterChunkedRead` sample by accessing the Oracle SOA Sample Code site.

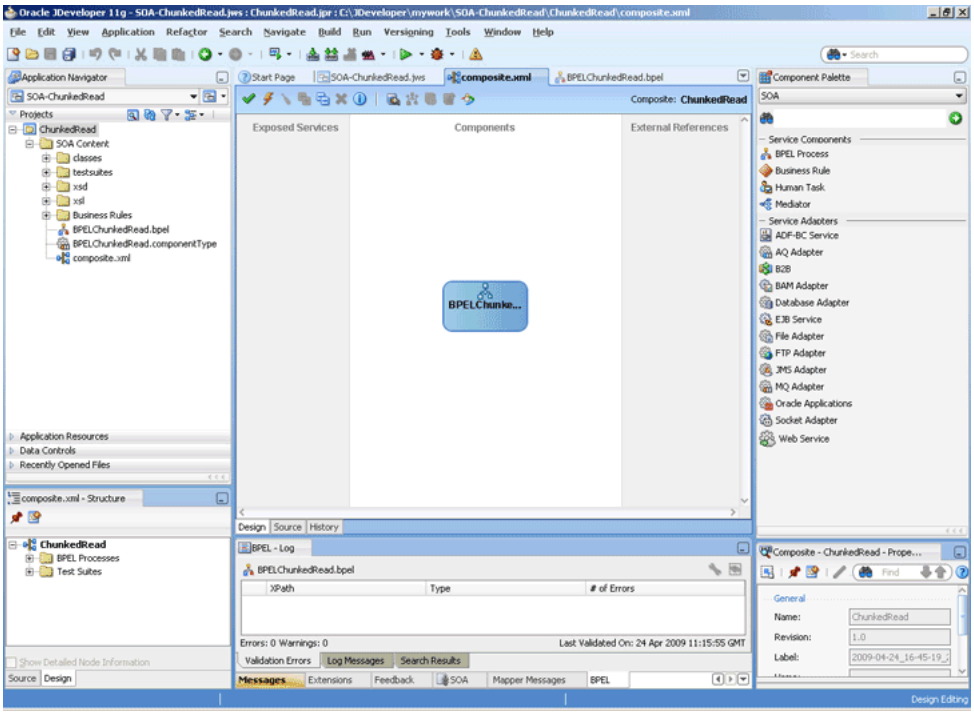
4.5.5.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `SOA-ChunkedRead` in the **Application Name** field, and click **Next**. The Create Generic Application - Name your project page is displayed.
3. Enter `ChunkedRead` in the **Project Name** field.
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process - BPEL Process page is displayed.

7. Enter `BPELChunkedRead` in the **Name** field, select **Define Service Later** from the Template box.
8. Click **OK**. The SOA-ChunkedRead application and the ChunkedRead project appears in the design area, as shown in Figure 4–98.

Figure 4–98 The JDeveloper - Composite.xml



9. Copy the **address-csv.xsd** and **address-fixedLength.xsd** files to the xsd directory in your project (see Section 4.5.5.1, "Prerequisites" for the location of these files).
10. Copy **addr1Toaddr2.xsl** to the xsl directory of your project (see Section 4.5.5.1, "Prerequisites" for the location of these files).

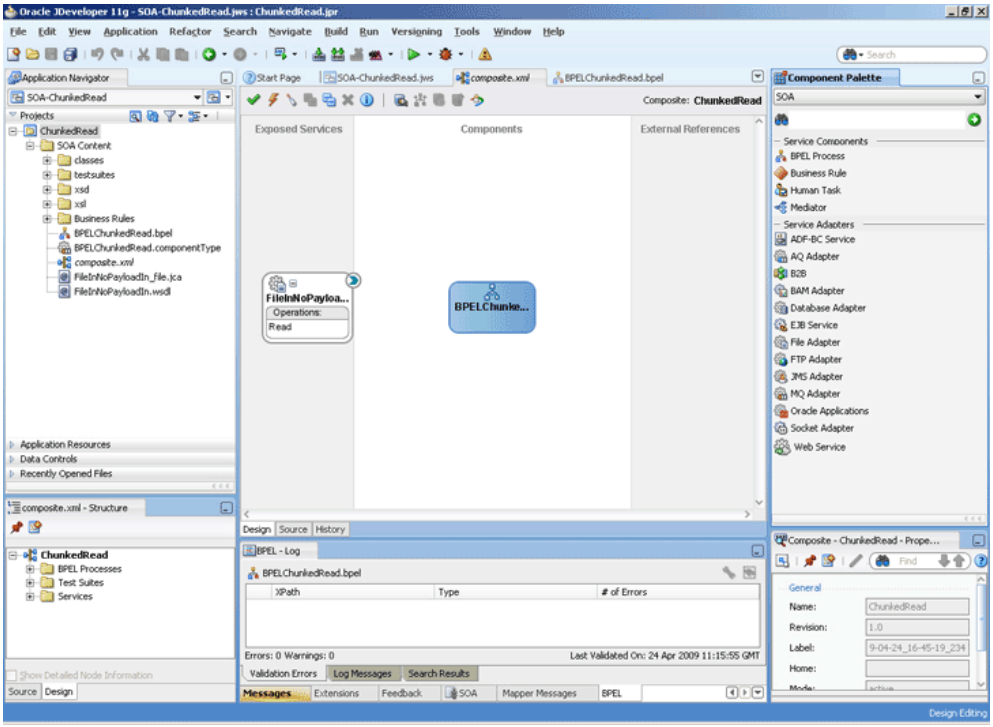
4.5.5.3 Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory:

1. Drag and drop File Adapter from the Component Palette to the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `FileInNoPayloadIn` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Read File**, check **Do Not Read File Content box**, and then click **Next**. The File Directories page is displayed.
7. Enter the physical path for the input directory. Check **Process Files Recursively**.
8. Click **Next**. The File Filtering page is displayed.

9. Enter `*.txt` in the **Include Files With Name Pattern** field, click **Next**. The File Polling page is displayed.
10. Click **Next**. The Finish page is displayed.
11. Click **Finish**. The inbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–99.

Figure 4–99 The JDeveloper - Composite.xml



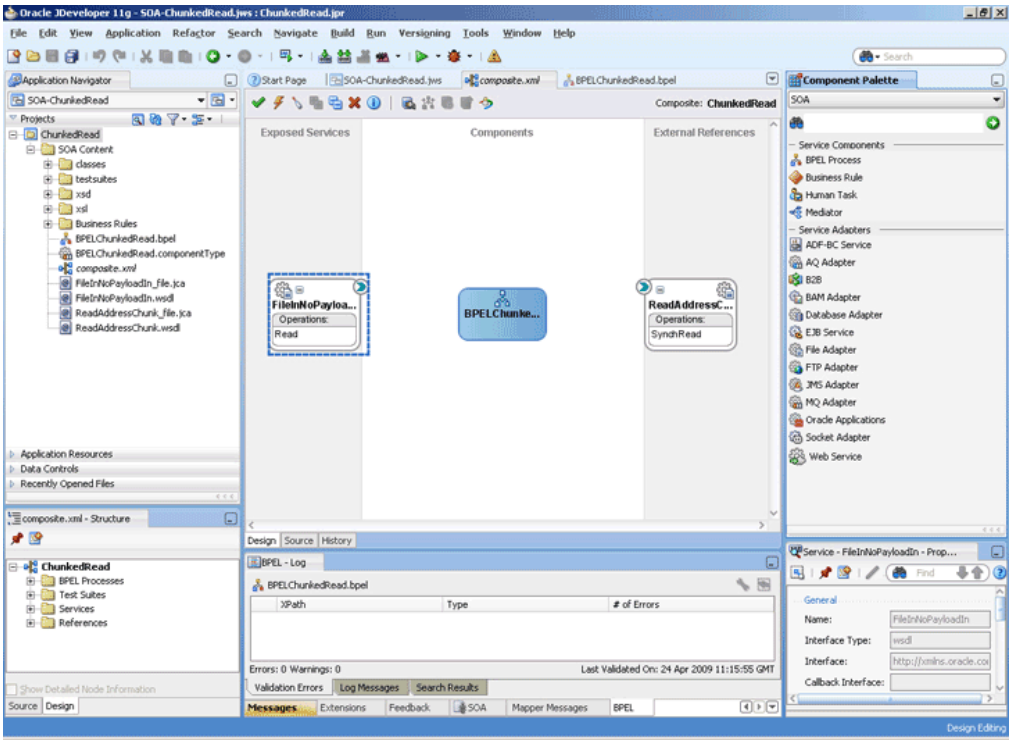
4.5.5.4 Creating the Outbound Oracle File Adapter Service

Perform the following steps to create an outbound Oracle File Adapter service to write the file from a local directory to the FTP server:

1. Drag and drop File Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `ReadAddressChunk` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Synchronous Read File**, enter `ChunkedRead` in the **Operation Name** field, and then click **Next**. The File Directories page is displayed.
7. Enter the physical path for the output directory and select **Delete Files After Successful Retrieval**.
8. Click **Next**. The File Name page is displayed.
9. Enter `dummy.txt` in the **File Name** field.

10. Click **Next**. The Messages page is displayed.
11. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
12. Click **Project Schema Files**, **address-csv.xsd**, and **Root-Element**.
13. Click **OK**. The URL field in the Messages page is populated with the `address-csv.xsd` file.
14. Click **Next**. The Finish page is displayed.
15. Click **Finish**. The outbound Oracle File Adapter is now configured and `composite.xml` appears, as shown in Figure 4–100.

Figure 4–100 The JDeveloper - Composite.xml



16. Manually edit the metadata to incorporate the chunked read feature.

    Open `ReadAddressChunk_file.jca` file and modify the metadata as shown below:

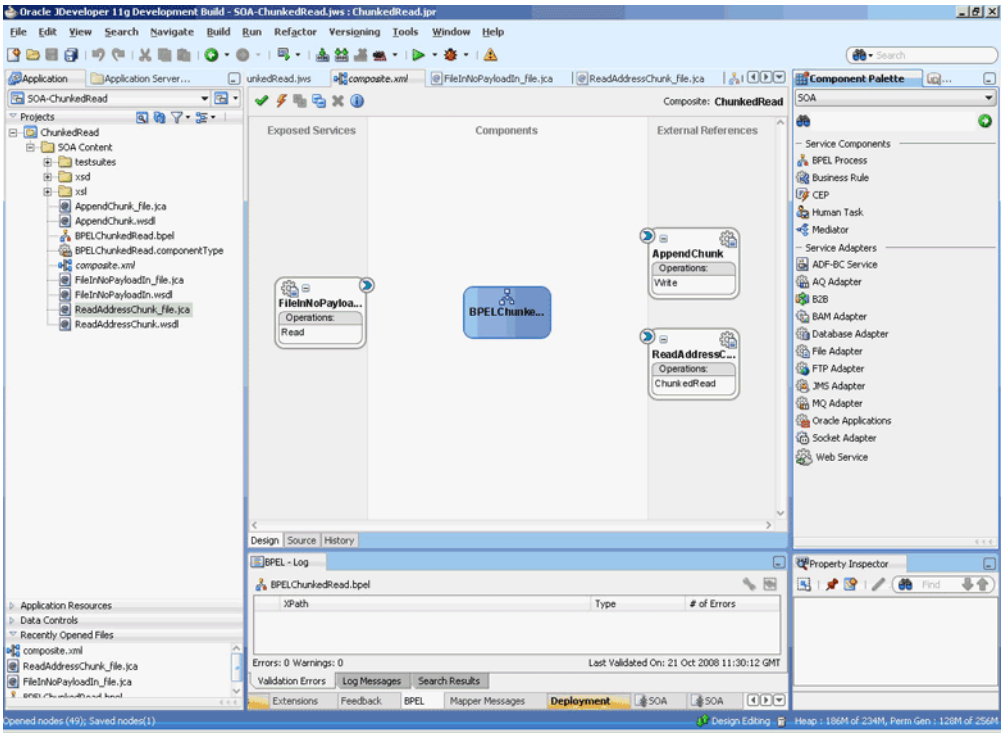
```
<?xml version="1.0" encoding="UTF-8"?>
<adapter-config name="ReadAddressChunk" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter" adapterRef=""/>
  <endpoint-interaction portType="ChunkedRead_ptt" operation="ChunkedRead">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.ChunkedInteractionSpec">
      <property name="PhysicalDirectory" value="/tmp/chunked/in"/>
      <property name="FileName" value="dummy.txt"/>
      <property name="ChunkSize" value="1"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

17. Click **File**, **Save All**.

Add Another Outbound Oracle File Adapter Service

1. Drag and drop File Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `AppendChunk` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, enter `Write` in the **Operation Name** field, and then click **Next**. The File Configuration page is displayed.
7. Enter the physical path for the output directory, enter `dummy.txt` in the **File Naming Convention (po_%SEQ%.txt)** and select **Append to Existing File**.
8. Click **Next**. The Messages page is displayed.
9. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
10. Click **Project Schema Files**, **address-fixedLength.xsd**, and **Root-Element**.
11. Click **OK**. The URL field in the Messages page is populated with the `address-fixedLength.xsd` file.
12. Click **Next**. The Finish page is displayed.
13. Click **Finish**. The outbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–101.

Figure 4–101 The JDeveloper - Composite.xml



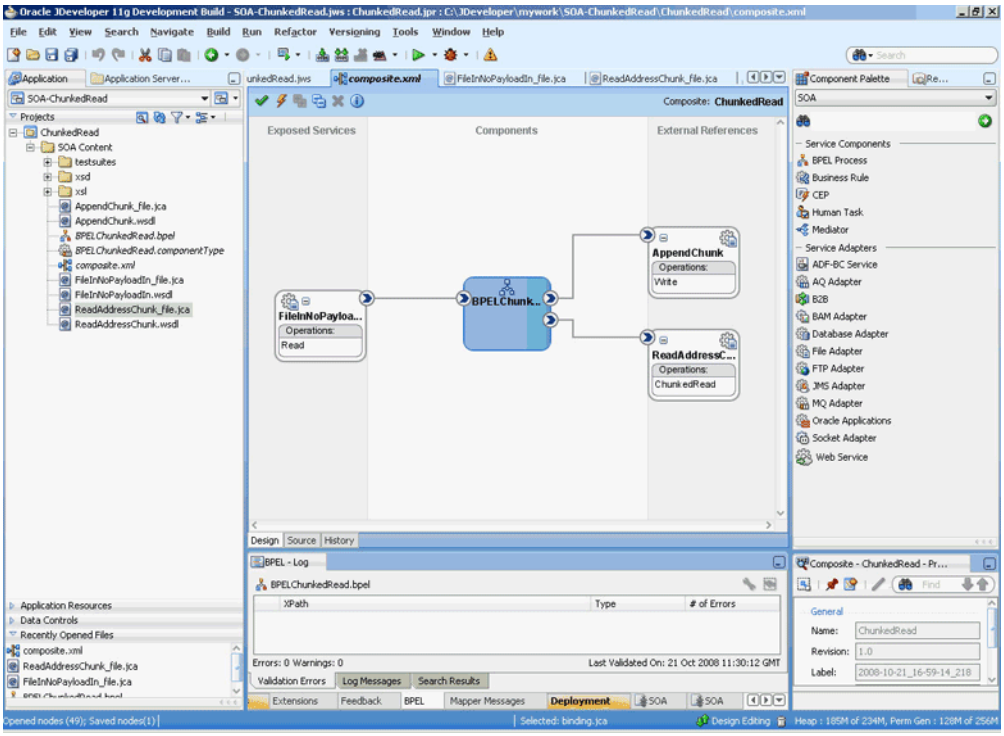
4.5.5.5 Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, two Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the FileInNoPayloadIn in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the ReadAddressChunk in the External References area and also to the green triangle in the AppendChunk in the External References area.

   The JDeveloper `composite.xml` appears, as shown in Figure 4–102.

Figure 4–102 The JDeveloper - Composite.xml



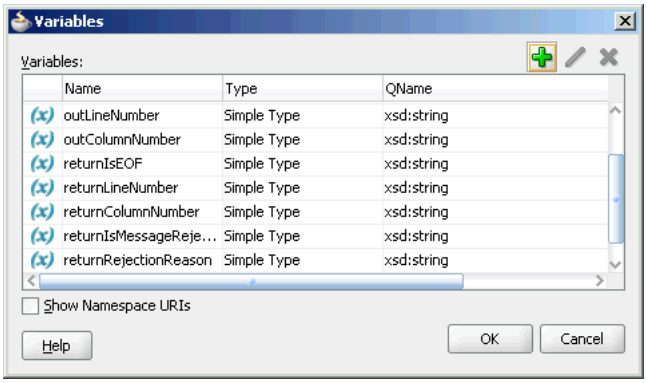
3. Click **File**, **Save All**.

Add a Receive Activity

1. Double-click **BPELChunkedRead**. The BPELChunkedRead.bpel page is displayed.
2. Click the **Variables...** icon represented by (x). The Variables dialog is displayed.
3. Click the **Create...** icon. The Create Variable dialog is displayed.
4. Create the following variables, as shown in Figure 4–103, for later use:

```
<variable name="dir" type="xsd:string"/>
   <variable name="file" type="xsd:string"/>
   <variable name="outIsEOF" type="xsd:string"/>
   <variable name="outLineNumber" type="xsd:string"/>
   <variable name="outColumnNumber" type="xsd:string"/>
   <variable name="returnIsEOF" type="xsd:string"/>
   <variable name="returnLineNumber" type="xsd:string"/>
   <variable name="returnColumnNumber" type="xsd:string"/>
   <variable name="returnIsMessageRejected" type="xsd:string"/>
   <variable name="returnRejectionReason" type="xsd:string"/>
   <variable name="returnNoDataFound" type="xsd:string"/>
```

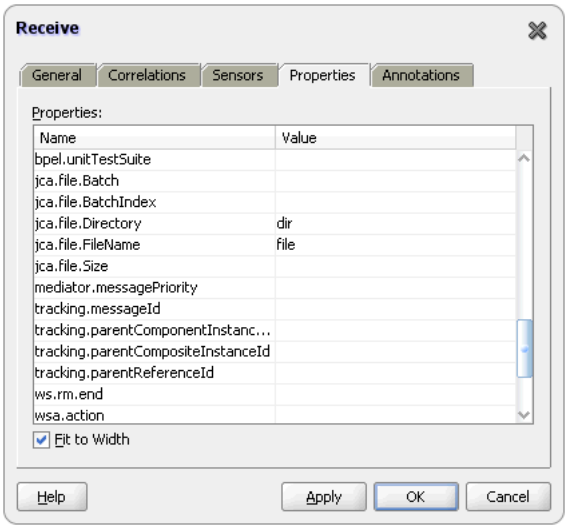
Figure 4–103 The Variables Dialog



Note: All variables are Simple Types of type xsd:string.

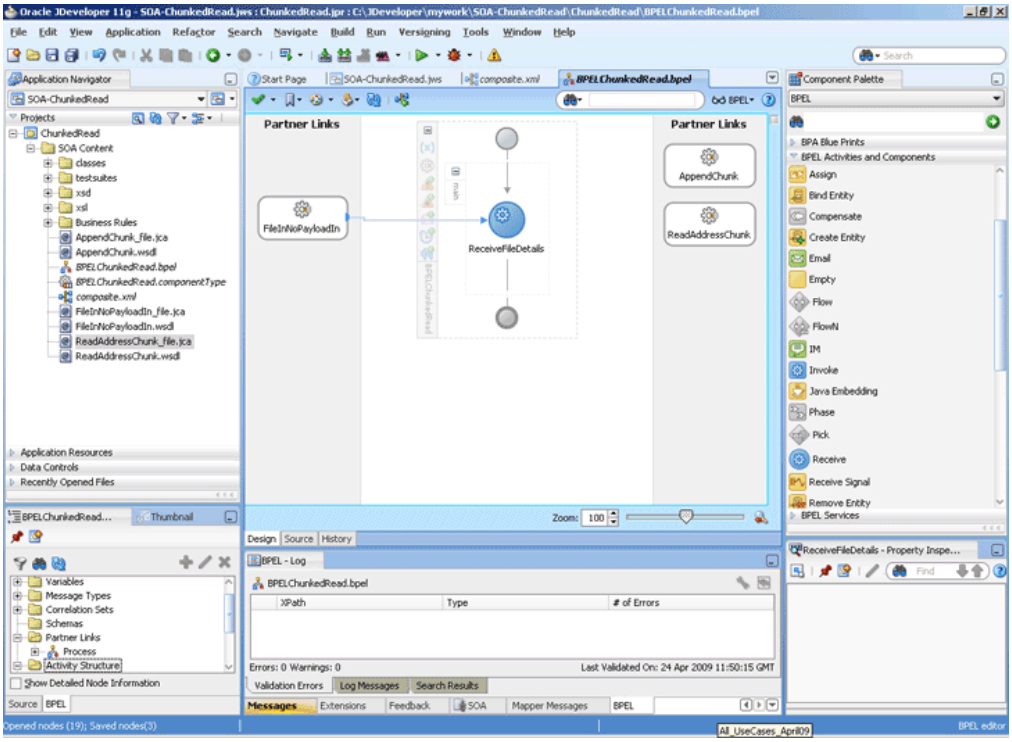
5. Drag and drop a **Receive** activity from the Component Palette to the design area.
6. Double-click the **Receive** activity. The Receive dialog is displayed.
7. Enter `ReceiveFileDetails` in the **Name** field.
8. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
9. Select **FileInNoPayloadIn**, and click **OK**.
10. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog. The Create Variable dialog is displayed.
11. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
12. Check **Create Instance**.
13. Click the **Properties** tab. The properties and the corresponding value column is displayed.
14. Select jca.file.Directory property. Double-click in the corresponding value column. The Adapter Property value dialog is displayed.
15. Click the **Browse Variables** icon. The Variable XPath Builder dialog is displayed.
16. Expand **Variables**, select **dir**, and then click **OK**. The value of the `jca.file.Directory` is set to `dir`.
17. Repeat the same for `jca.file.FileName` property and set the value to **file**. The Receive dialog is displayed, as shown in Figure 4–104.

Figure 4–104 The Receive Dialog - Adapters Tab



18. Click **OK**. The JDeveloper BPELChunkedRead.bpel page appears, as shown in Figure 4–105

Figure 4–105 The JDeveloper - BPELChunkedRead.bpel

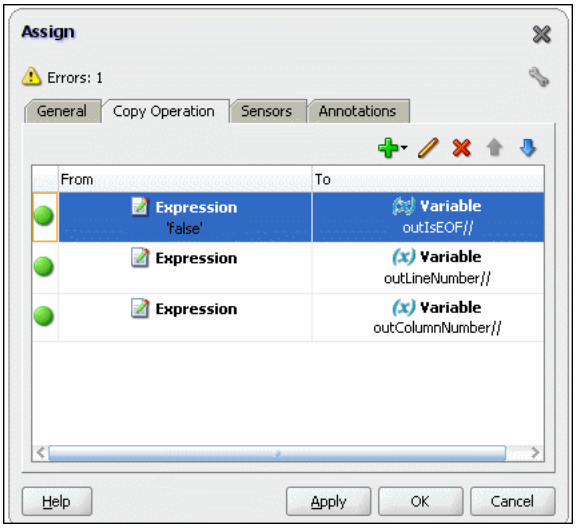


Add an Assign Activity

1. Drag and drop an **Assign** activity from the Component Palette after the Receive activity in the design area.
2. Double-click the **Assign** activity. The Assign dialog is displayed.
3. Enter `AssignChunkedRead` in the **Name** field.

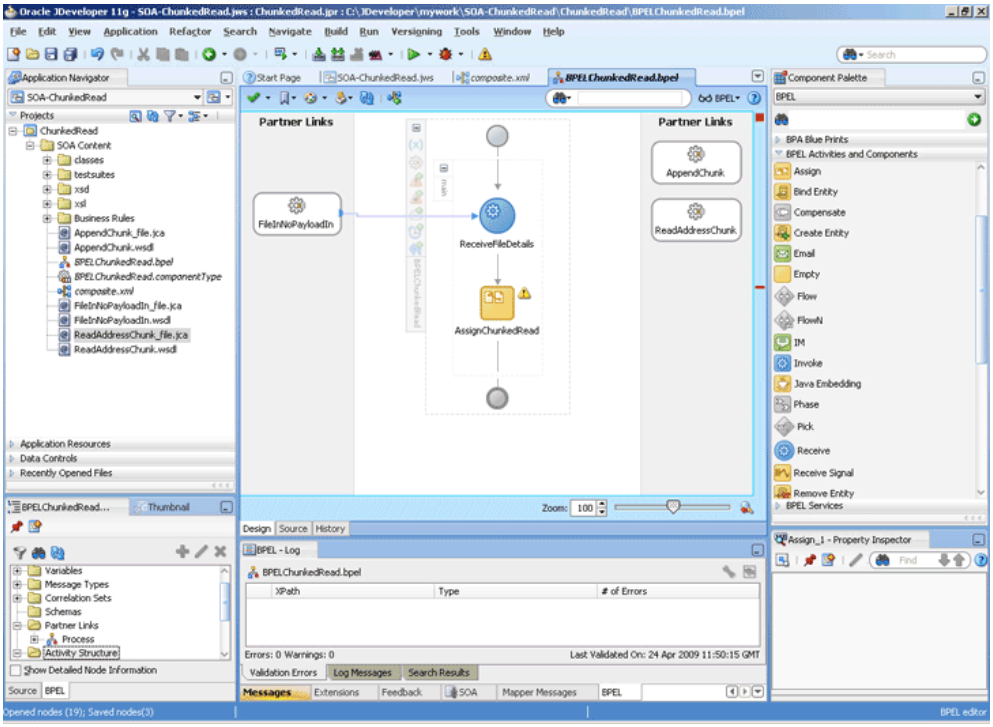
4. Click the **Copy Operation** tab. The Assign dialog is displayed, as shown in Figure 4–94.
5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
6. Set the default values for the headers, as shown in Figure 4–106.

Figure 4–106 The Assign Dialog



7. Click **OK**, the JDeveloper BPELChunkedRead.bpel page is displayed, as shown in Figure 4–107.

Figure 4–107 The JDeveloper - BPELChunkedRead.bpel

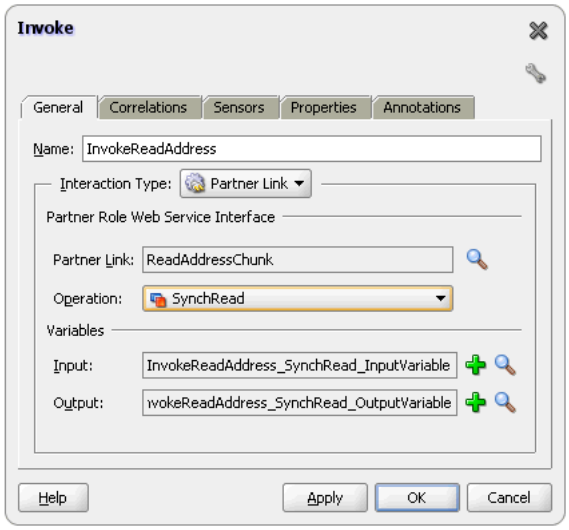


8. Click **File**, **Save All**.

Add an Invoke Activity

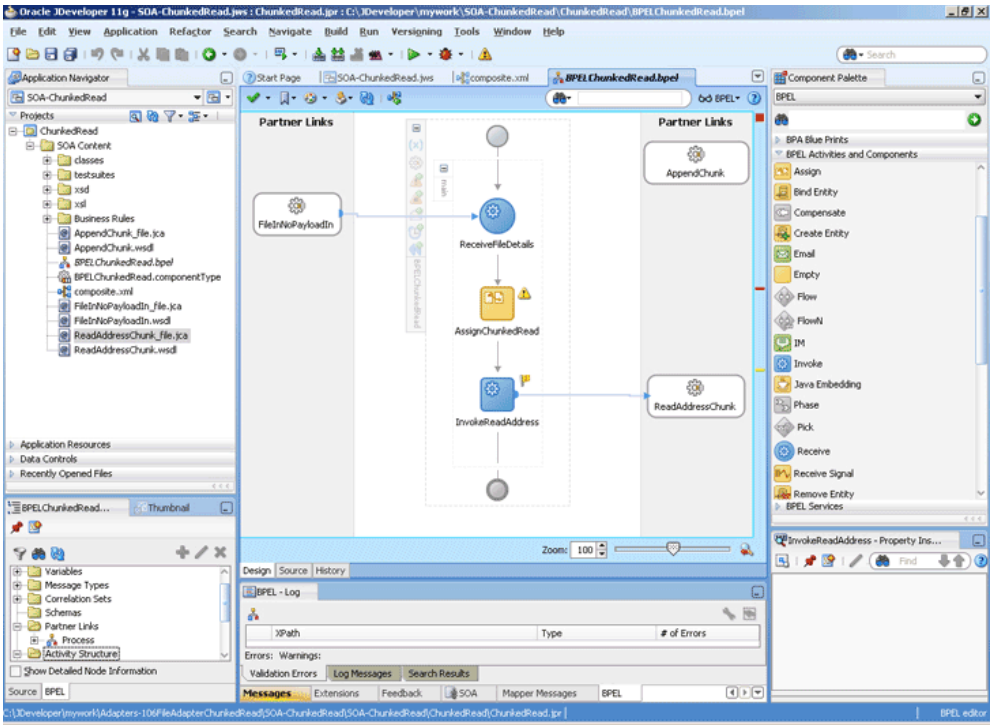
1. Drag and drop an **Invoke** activity below the Assign Activity from the Component Palette to the design area.
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `InvokeReadAddress` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **ReadAddressChunk**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name. The Invoke dialog is displayed with input variable populated.
8. Repeat the same to select the output variable. The Invoke dialog is displayed, as shown in Figure 4–108.

Figure 4–108 The Invoke Dialog



9. Click **OK**. The JDeveloper BPELChunkedRead.bpel page appears, as shown in Figure 4–109.

Figure 4–109 The JDeveloper - BPELChunkedRead.bpel



10. Click the **Source** tab for the BPELChunkedRead.bpel page, and add the following properties for the invoke activity that you just created:

```
<bpelx:inputProperty name="jca.file.Directory" variable="dir"/>
<bpelx:inputProperty name="jca.file.FileName" variable="file"/>
<bpelx:inputProperty name="jca.file.LineNumber" ="outLineNumber"/>
<bpelx:inputProperty name="jca.file.ColumnNumber"
variable="outColumnNumber"/>
<bpelx:inputProperty name="jca.file.IsEOF" variable="outIsEOF"/>
<bpelx:outputProperty name="jca.file.LineNumber"
variable="returnLineNumber"/>
<bpelx:outputProperty name="jca.file.ColumnNumber"
variable="returnColumnNumber"/>
<bpelx:outputProperty name="jca.file.IsEOF" variable="returnIsEOF"/>
<bpelx:outputProperty name="jca.file.IsMessageRejected"
variable="returnIsMessageRejected"/>
<bpelx:outputProperty name="jca.file.RejectionReason"
variable="returnRejectionReason"/>
<bpelx:outputProperty name="jca.file.NoDataFound"
variable="returnNoDataFound"/>
```

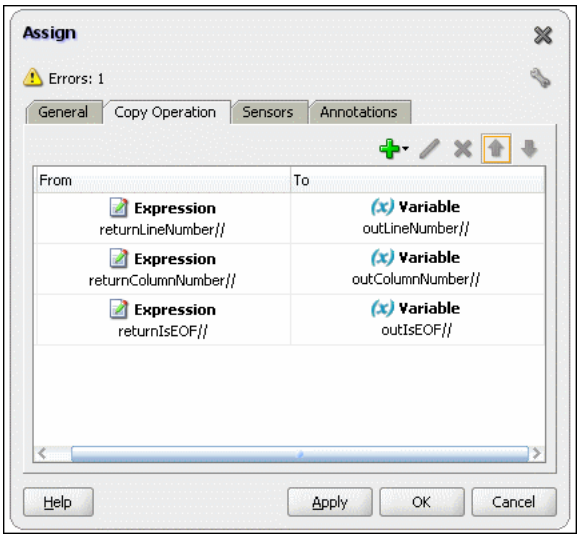
The invoke activity appears as follows:

```
<invoke name="InvokeReadAddress"
inputVariable="InvokeReadAddress_SynchRead_InputVariable"
outputVariable="InvokeReadAddress_SynchRead_OutputVariable"
partnerLink="ReadAddressChunk" portType="ns3:SynchRead_ptt"
operation="SynchRead">
<bpelx:inputProperty name="jca.file.Directory" variable="dir"/>
<bpelx:inputProperty name="jca.file.FileName" variable="file"/>
<bpelx:inputProperty name="jca.file.LineNumber" variable="outLineNumber"/>
<bpelx:inputProperty name="jca.file.ColumnNumber"
variable="outColumnNumber"/>
```

```
<bpelx:inputProperty name="jca.file.IsEOF" variable="outIsEOF"/>
<bpelx:outputProperty name="jca.file.LineNumber"
variable="returnLineNumber"/>
<bpelx:outputProperty name="jca.file.ColumnNumber"
variable="returnColumnNumber"/>
<bpelx:outputProperty name="jca.file.IsEOF" variable="returnIsEOF"/>
<bpelx:outputProperty name="jca.file.IsMessageRejected"
variable="returnIsMessageRejected"/>
<bpelx:outputProperty name="jca.file.RejectionReason"
variable="returnRejectionReason"/>
<bpelx:outputProperty name="jca.file.NoDataFound"
variable="returnNoDataFound"/>
</invoke>
```

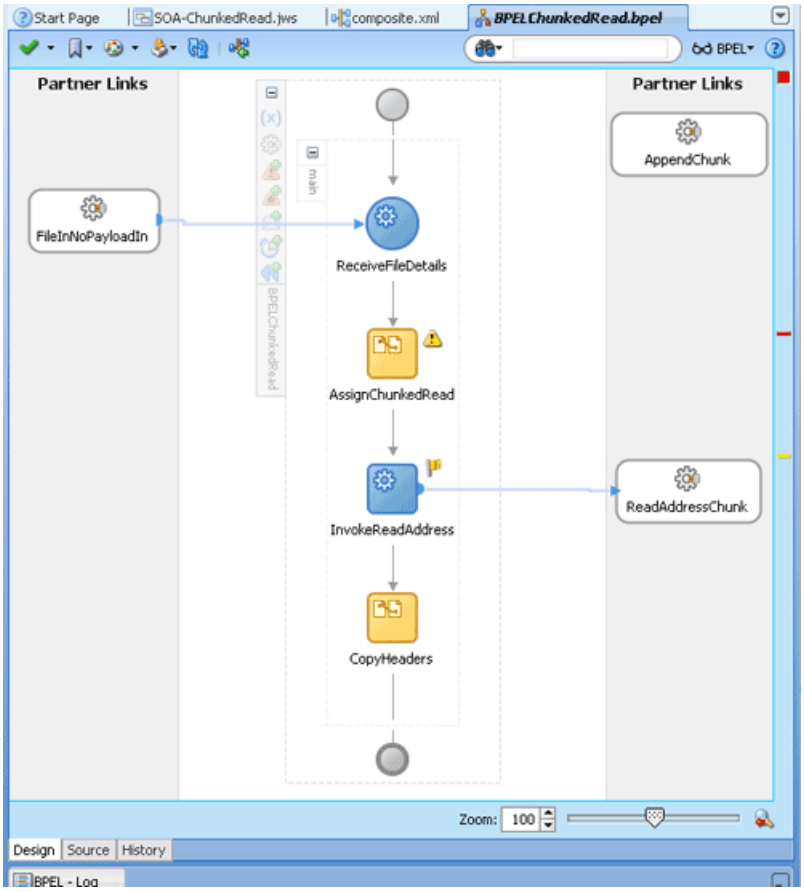
11. Add an assign activity called `CopyHeaders`, as given in Add an Assign Activity, to copy the return parameters from the invoke activity. The Assign dialog is displayed, as shown in Figure 4–110.

Figure 4–110 The Assign Dialog



12. Click **OK**. The JDeveloper BPELChunkedRead.bpel page is displayed, as shown in Figure 4–111.

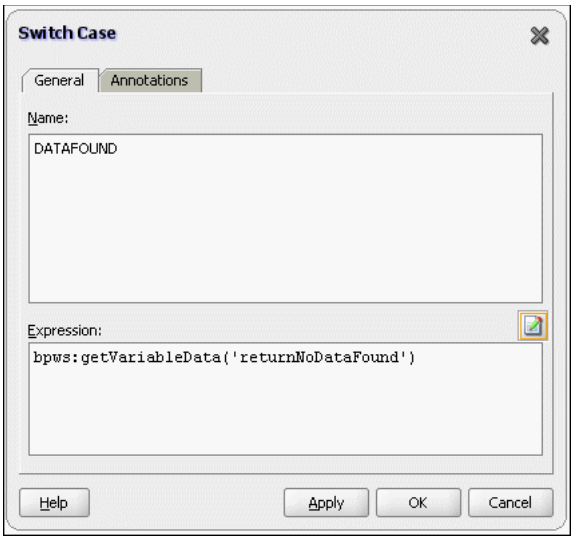
Figure 4–111 The JDeveloper - BPELChunkedRead.bpel



Add a Switch Activity

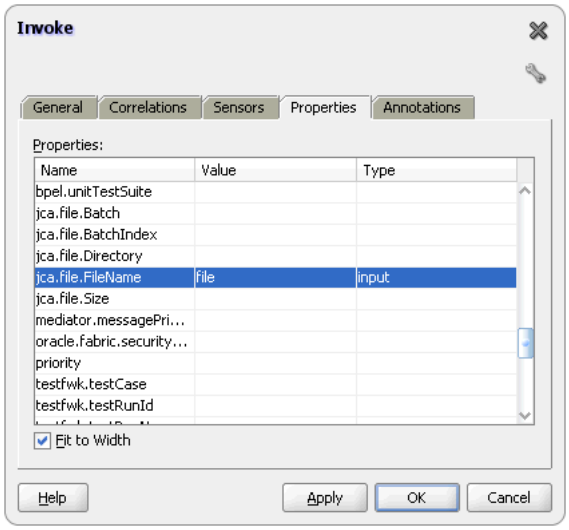
1. Drag and drop a **Switch** activity below the CopyHeaders Assign activity.
2. Double-click <case> in the Switch activity. The Switch Case dialog is displayed.
3. Enter `DATA FOUND` in the **Name** field and select the returnNoDataFound expression in the Expression box. The Switch Case dialog is displayed, as shown in Figure 4–112.

Figure 4–112 The Switch Case Dialog



4. Drag and drop an **Invoke** activity in the <Case DATA FOUND> for Switch Activity.
5. Double-click the **Invoke** activity. The Invoke dialog is displayed.
6. Enter `InvokeAppend` in the **Name** field.
7. Select AppendChunk in the Partner Link field.
8. Click the **Automatically Create Input Variable** icon to the right of the **Input** variable field in the Invoke dialog. The Create Variable dialog is displayed.
9. Select the default variable name and click **OK**. The Variable field is populated with the default variable name. The Invoke dialog is displayed with input variable populated.
10. Click the **Properties** tab and select file variable, as shown in Figure 4–113.

Figure 4–113 The Invoke Dialog

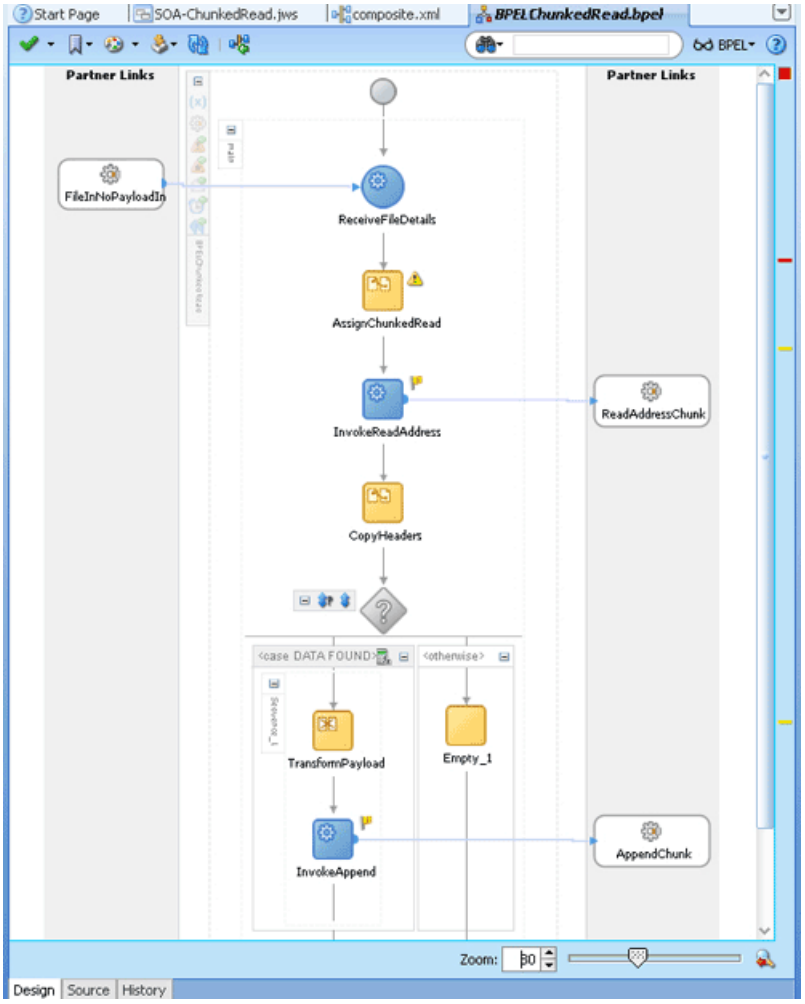


11. Click **OK**.

Add a Transform Activity

1. Drag and drop a **Transform** activity in the <case DATA FOUND> section just before the InvokeAppend activity.
2. Double-click the **Transform** activity.
3. Enter `TransformPayload` in the **Name** field.
4. Click the **Transformation** tab.
5. Click the **Create...** icon. The Source Variable dialog is displayed.
6. Select **InvokeReadAddress_SyncRead_InputVariable**, and click **OK**.
7. Select **InvokeAppend_Write_InputVariable** from the Target Variable list.
8. Click **Browse** at the end of the Mapper File field, and select the **addr1Toaddr2.xsl** file.
9. Click **OK**.
10. Drag and drop an **Empty** activity in the <otherwise> section in the Switch activity. The BPELChunkedRead.bpel page is displayed, as shown in Figure 4–114.

Figure 4–114 The JDeveloper - BPELChunkedRead.bpel



11. Click **File**, **Save All**.

4.5.5.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application. For more information, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

4.5.5.7 Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the `address-csv.txt` file to the input directory (see Section 4.5.5.1, "Prerequisites" for the location of this file) and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the Recent Instances area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The Flow Trace page is displayed.
6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow. Additionally, click an activity to view the details of an activity.

4.5.6 Oracle File Adapter Read File As Attachments

This is an Oracle File Adapter feature to opaquely copy or move large amount of data, from a source directory on your file system to a destination directory, as attachments. For example, you can transfer large MS Word documents, images, and PDFs without processing their content within the composite application. The read file as attachment feature is available only when the Read File option is chosen.

This use case demonstrates the ability of the Oracle File Adapter to process a large `*.doc` file as an attachment. This feature of reading files as attachments is very similar to `Opaque` translation. However, attachments can be of the order of gigabytes depending on database limitations.

- Section 4.5.6.1, "Prerequisites"
- Section 4.5.6.2, "Designing the SOA Composite"
- Section 4.5.6.3, "Creating the Inbound Oracle File Adapter Service"
- Section 4.5.6.4, "Creating the Outbound Oracle File Adapter Service"
- Section 4.5.6.5, "Wiring Services and Activities"
- Section 4.5.6.6, "Deploying with JDeveloper"
- Section 4.5.6.7, "Monitoring Using Fusion Middleware Control Console"

4.5.6.1 Prerequisites

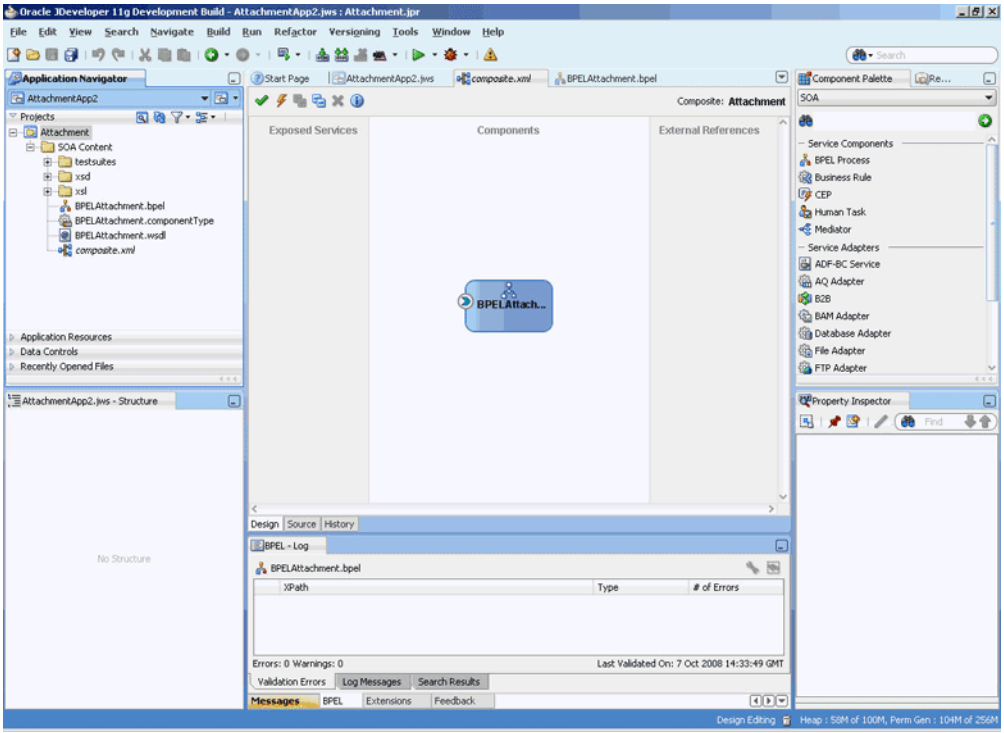
To perform Oracle File Adapter read file as attachments, you require a large MS Word document (`*.doc` file).

4.5.6.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `AttachmentApp` in the **Application Name** field, and click **Next**. The Create Generic Application - Name your project page is displayed.
3. Enter `Attachment` in the **Project Name** field.
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process - BPEL Process page is displayed.
7. Enter `BPELAttachment` in the **Name** field, select **Define Service Later** from the Template list.
8. Click **OK**. The `AttachmentApp` application and the `Attachment` project appear in the design area, as shown in Figure 4–115.

Figure 4–115 The JDeveloper - Composite.xml



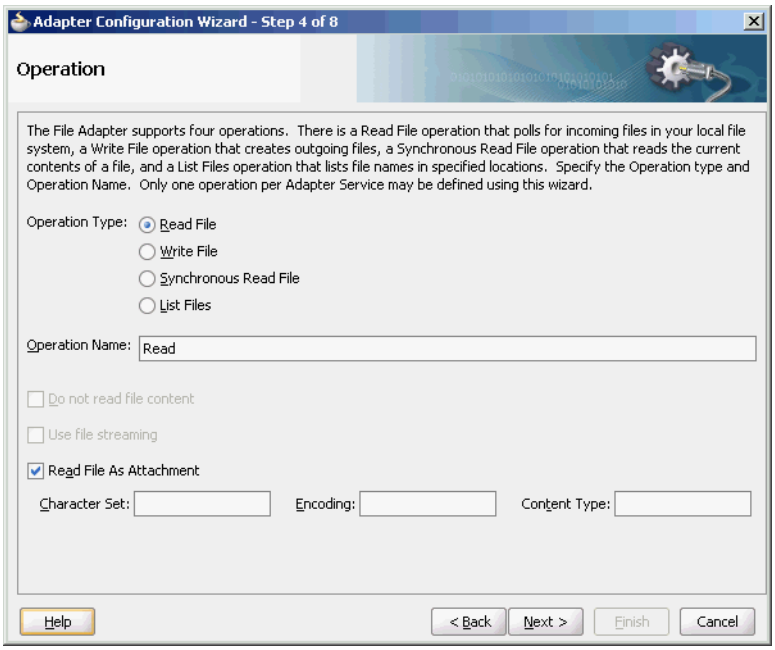
4.5.6.3 Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read a large file from a local directory:

1. Drag and drop File Adapter from the Component Palette to the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `AttachmentIn` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Read File** as the Operation Type and select **Read File As Attachment**, as shown in Figure 4–116, and then click **Next**. The File Directories page is displayed.

> **Note:** You must ignore Character Set, Encoding, and Content Type fields. These fields must be populated with values only if you are using third-party applications that must read this attachment. The attachment in this use case is finally consumed by an outbound Oracle File Adapter, hence these values are not required.

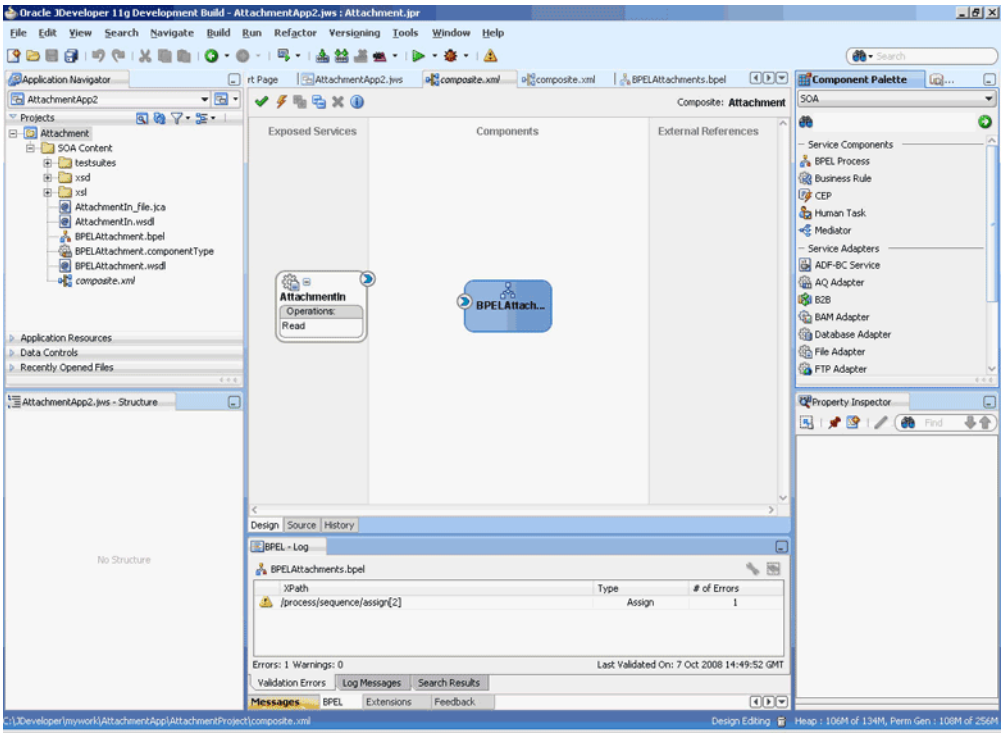
Figure 4–116 The Adapter Configuration Wizard Operation Page



7. Enter the physical path for the input directory, as shown in Figure 4–55 and click **Next**. The File Filtering page is displayed.
8. Enter `*.doc` in the **Include Files With Name Pattern** field, as shown in Figure 4–56.
9. Click **Next**. The File Polling page is displayed.
10. Click **Next**. The Finish page is displayed.

11. Click **Finish**. The inbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–117.

Figure 4–117 The JDeveloper - Composite.xml

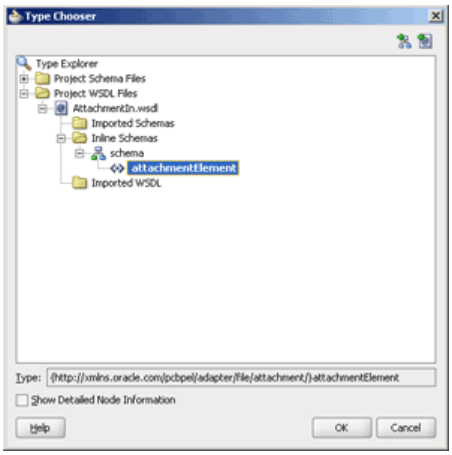


4.5.6.4 Creating the Outbound Oracle File Adapter Service

Perform the following steps to create an outbound Oracle File Adapter service to write the file from a local directory to the FTP server:

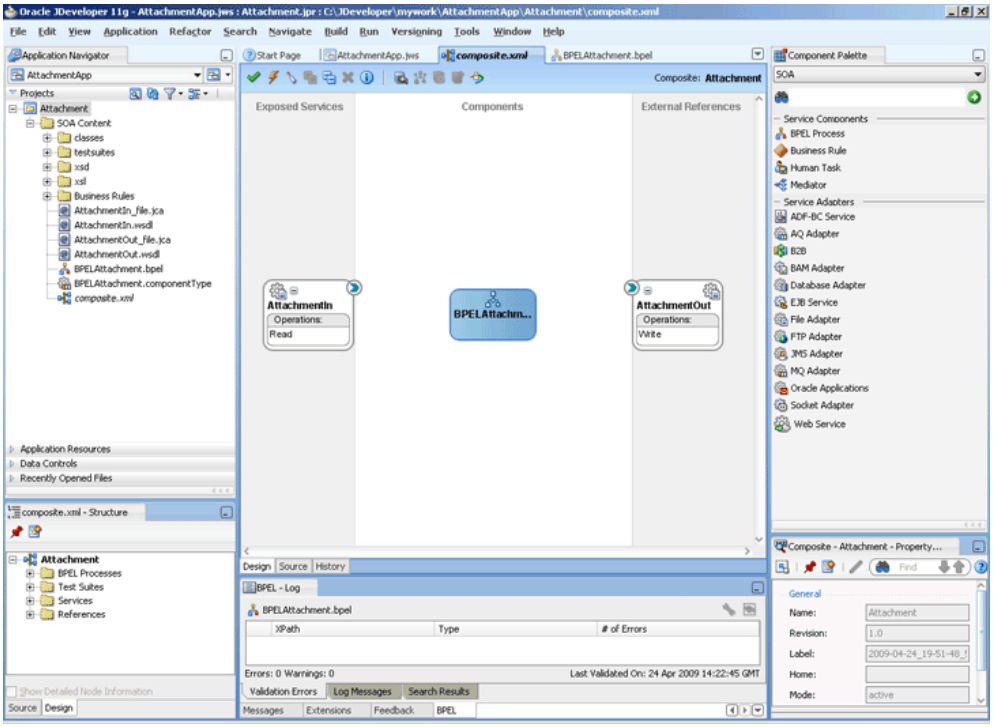
1. Drag and drop File Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `AttachmentOut` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, and click **Next**. The File Configuration page is displayed.
7. Enter the physical path for the output directory and enter `attachment_%SEQ%.doc` in the **File Naming Convention(po_%SEQ%.txt)** field, as shown in Figure 4–60.
8. Click **Next**. The Messages page is displayed.
9. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
10. Click **Project WSDL Files**, **AttachmentIn.wsdl**, **Inline Schemas**, and **attachmentElement**, as shown in Figure 4–118.

Figure 4–118 The Type Chooser Dialog



11. Click **OK**. The URL field in the Messages page is populated with AttachmentIn.wsdl.
12. Click **Next**. The Finish page is displayed.
13. Click **Finish**. The outbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–119.

Figure 4–119 The JDeveloper - Composite.xml



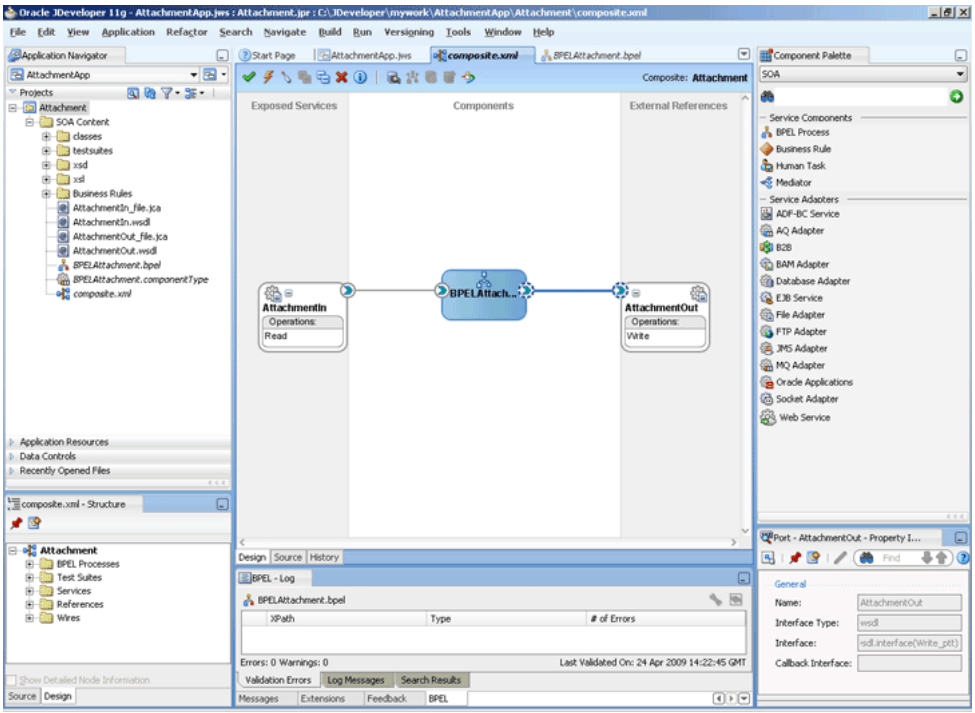
4.5.6.5 Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the AttachmentIn in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the AttachmentOut in the External References area.

   The JDeveloper composite.xml appears, as shown in Figure 4–120.

Figure 4–120 The JDeveloper - Composite.xml

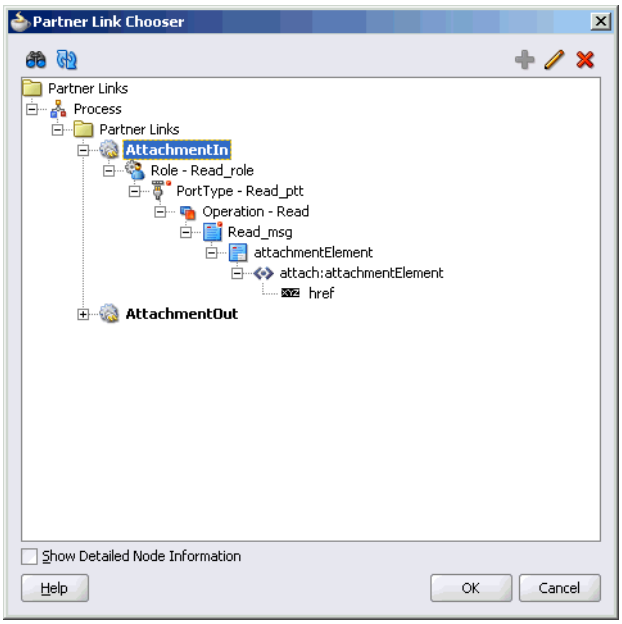


3. Click **File**, **Save All**.

Add a Receive Activity

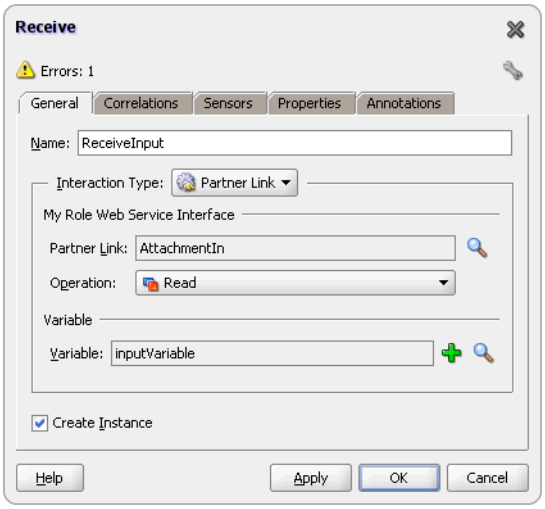
1. Double-click **BPELAttachment**. The BPELAttachment.bpel page is displayed.
2. Drag and drop a **Receive** activity from the Component Palette to the design area.
3. Double-click the **Receive** activity. The Receive dialog is displayed.
4. Enter `ReceiveInput` in the **Name** field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
6. Select **AttachmentIn**, as shown in Figure 4–121 and click **OK**.

Figure 4–121 The Partner Link Chooser Dialog



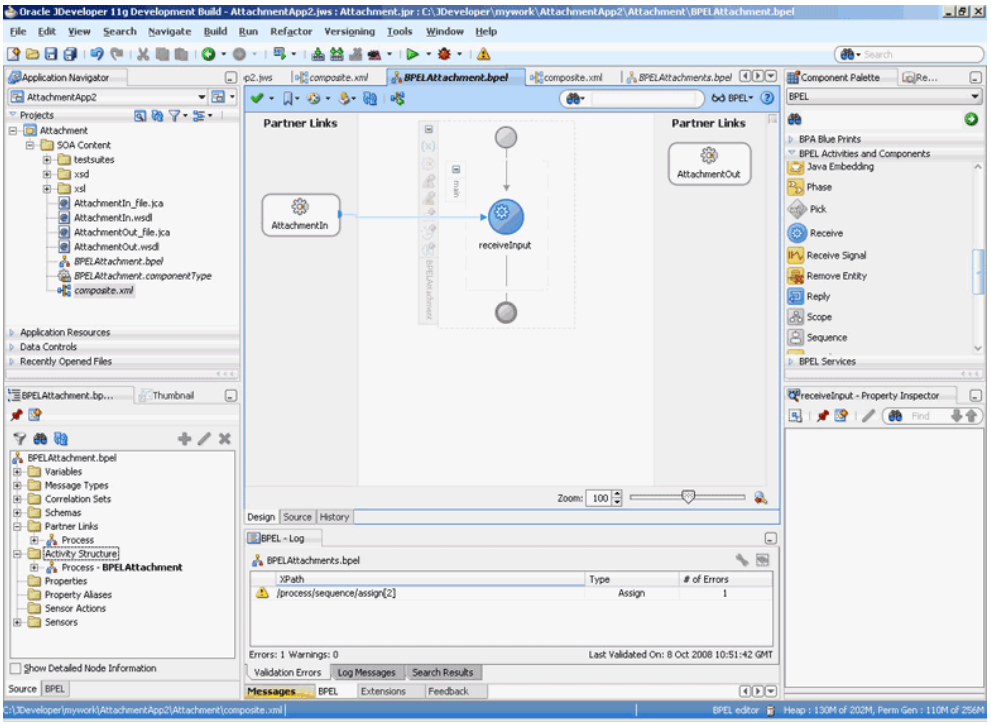
7. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog, as shown in Figure 4–122. The Create Variable dialog is displayed.

Figure 4–122 The Receive Dialog



8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper BPELAttachment.bpel page appears, as shown in Figure 4–123.

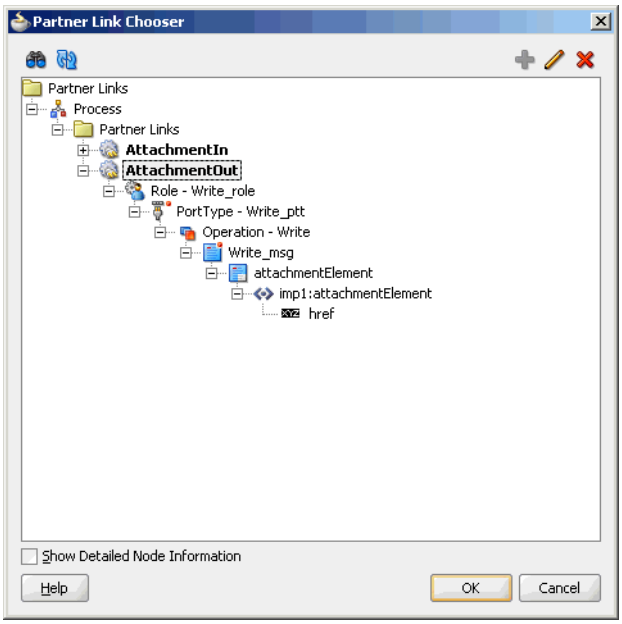
Figure 4–123 The JDeveloper - BPELXMLDebatching.bpel



Add an Invoke Activity

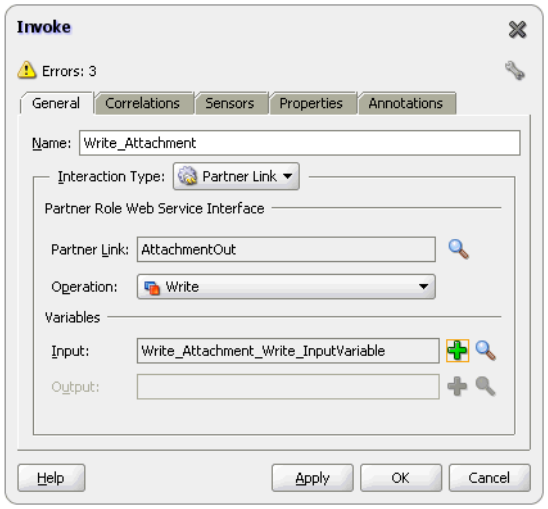
1. Drag and drop an **Invoke** activity from the Component Palette to the design area.
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `Write_Attachment` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **AttachmentOut**, as shown in Figure 4–124, and click **OK**.

Figure 4–124 The Partner Link Chooser Dialog



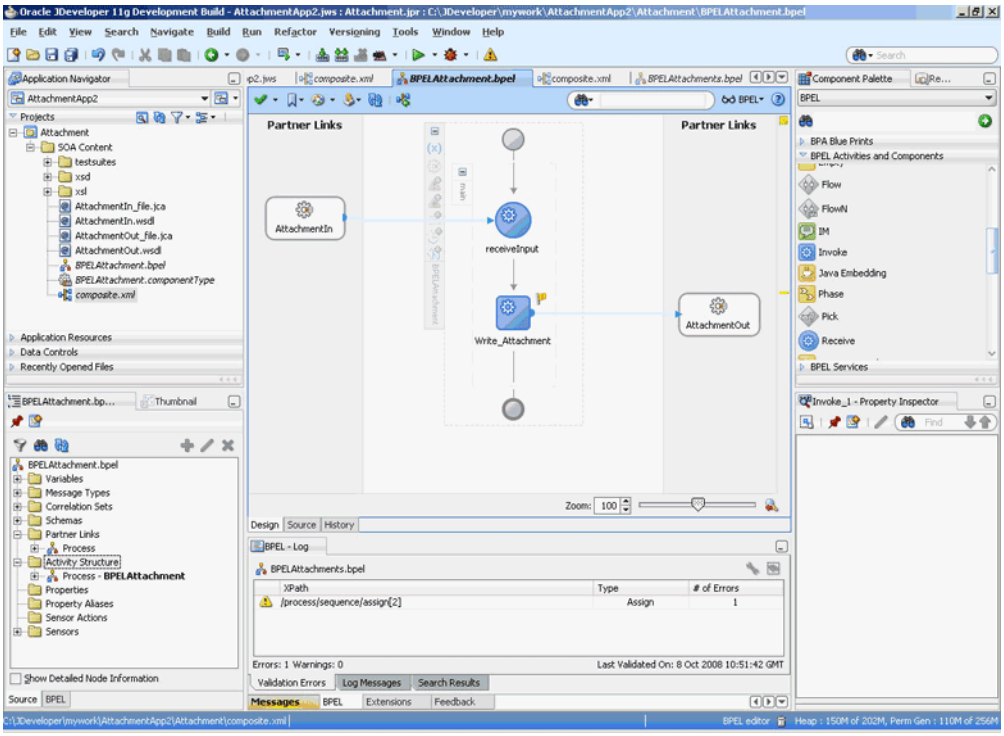
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name. The Invoke dialog is displayed, as shown in Figure 4–125.

Figure 4–125 The Invoke Dialog



8. Click **OK**. The JDeveloper BPELAttachment.bpel page appears, as shown in Figure 4–126.

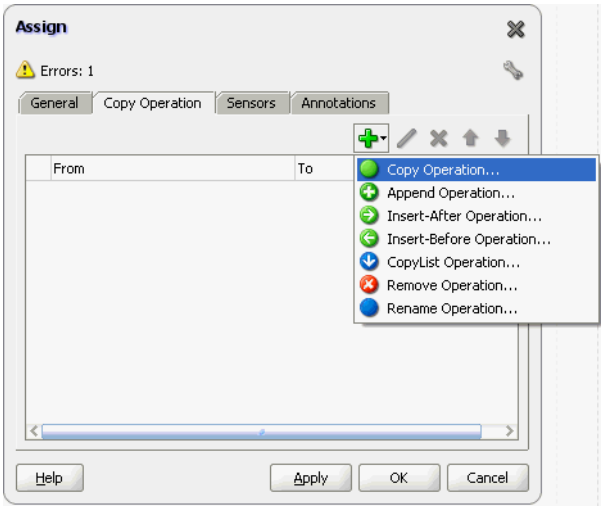
Figure 4–126 The JDeveloper - BPELXMLDebatching.bpel



Add an Assign Activity

1. Drag and drop an **Assign** activity from the Component Palette in between the Receive and Invoke activities in the design area.
2. Double-click the **Assign** activity. The Assign dialog is displayed.
3. Enter `AssignReference` in the **Name** field.
4. Click the **Copy Operation** tab. The Assign dialog is displayed, as shown in Figure 4–127.

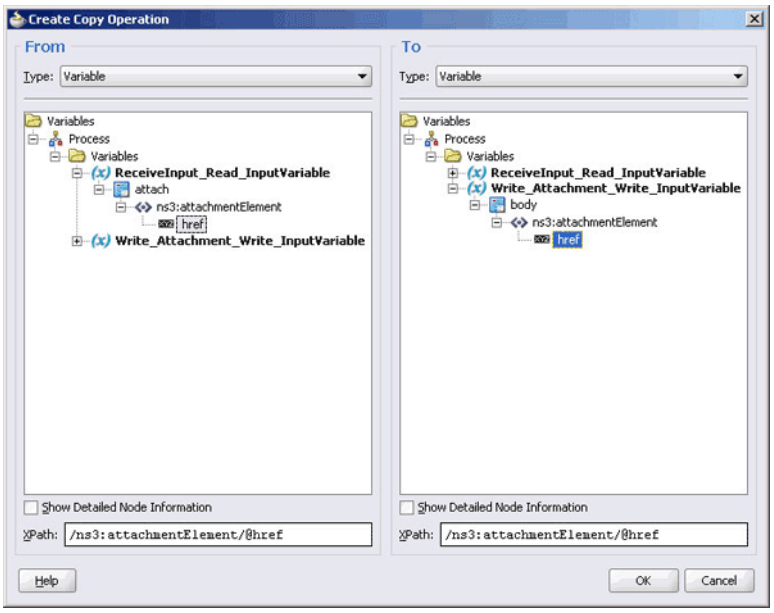
Figure 4–127 The Assign Dialog - Copy Operation Tab



5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.

6. Expand the variables in the From and To panes, as shown in Figure 4–128.

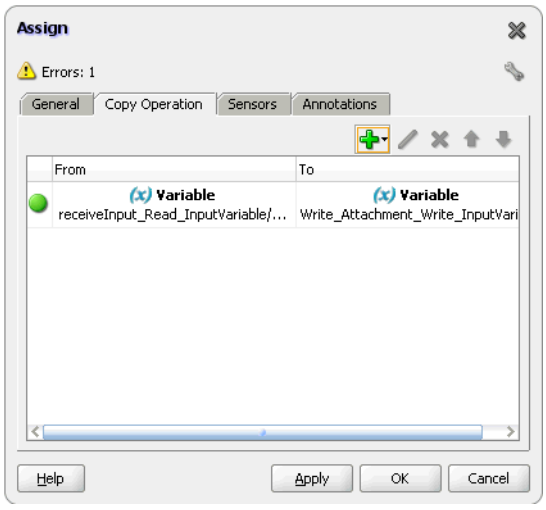
Figure 4–128 The Create Copy Operation Dialog



Note: In the case of variables defined by reference to an element, both the source and the target must be the same element.

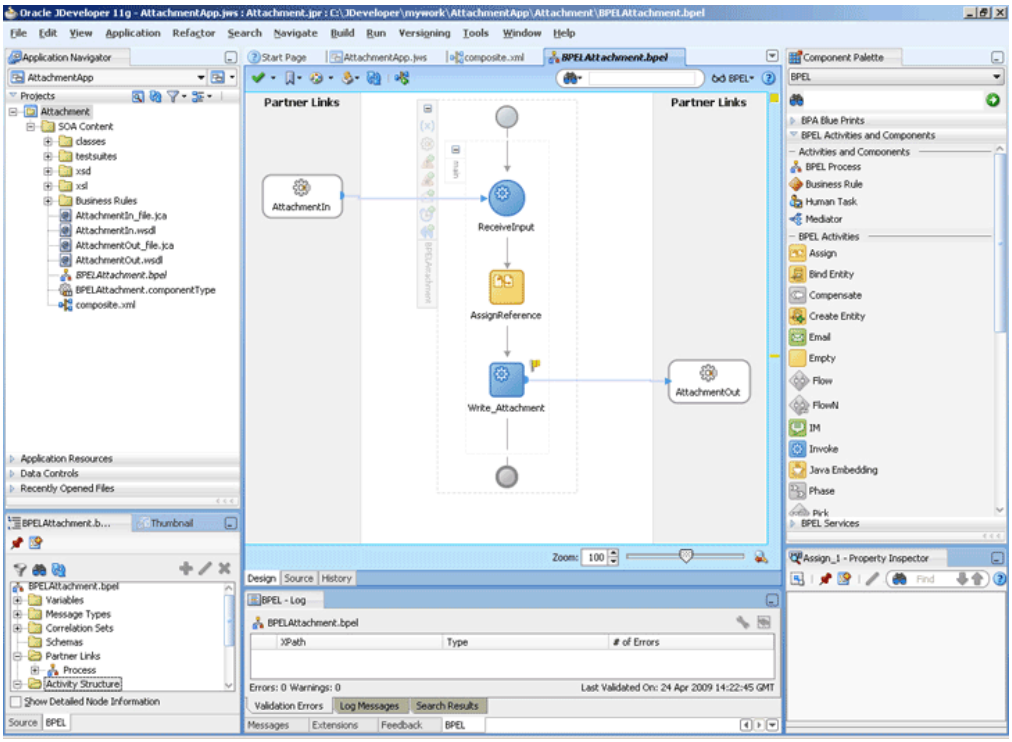
7. Click **OK**. The Assign dialog is displayed, as shown in Figure 4–129.

Figure 4–129 The Assign Dialog



8. Click **OK**, the JDeveloper BPELAttachment.bpel page is displayed, as shown in Figure 4–130.

Figure 4–130 The JDeveloper - BPELScalableDOM.bpel



9. Click **File**, **Save All**.

4.5.6.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application. For more information, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

4.5.6.7 Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://`*`servername`*`:`*`portnumber`*`/em`. The composite you deployed appears in the application navigator.
2. Copy the `attachment.doc` file to the input directory (see Section 4.5.6.1, "Prerequisites" for details) and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the Recent Instances area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The Flow Trace page is displayed.

6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow. Additionally, click an activity to view the details of an activity.

4.5.7 Oracle File Adapter File Listing

This is an Oracle File Adapter feature that lets you use an invoke activity to retrieve a list of files from a target directory. This list of files is returned as an XML document and contains information such as file name, directory name, file size, and last modified time.

This use case includes the following sections:

- Section 4.5.7.1, "Prerequisites"
- Section 4.5.7.2, "Designing the SOA Composite"
- Section 4.5.7.3, "Creating the Outbound Oracle File Adapter Service"
- Section 4.5.7.4, "Wiring Services and Activities"
- Section 4.5.7.5, "Deploying with JDeveloper"
- Section 4.5.7.6, "Monitoring Using Fusion Middleware Control Console"

4.5.7.1 Prerequisites

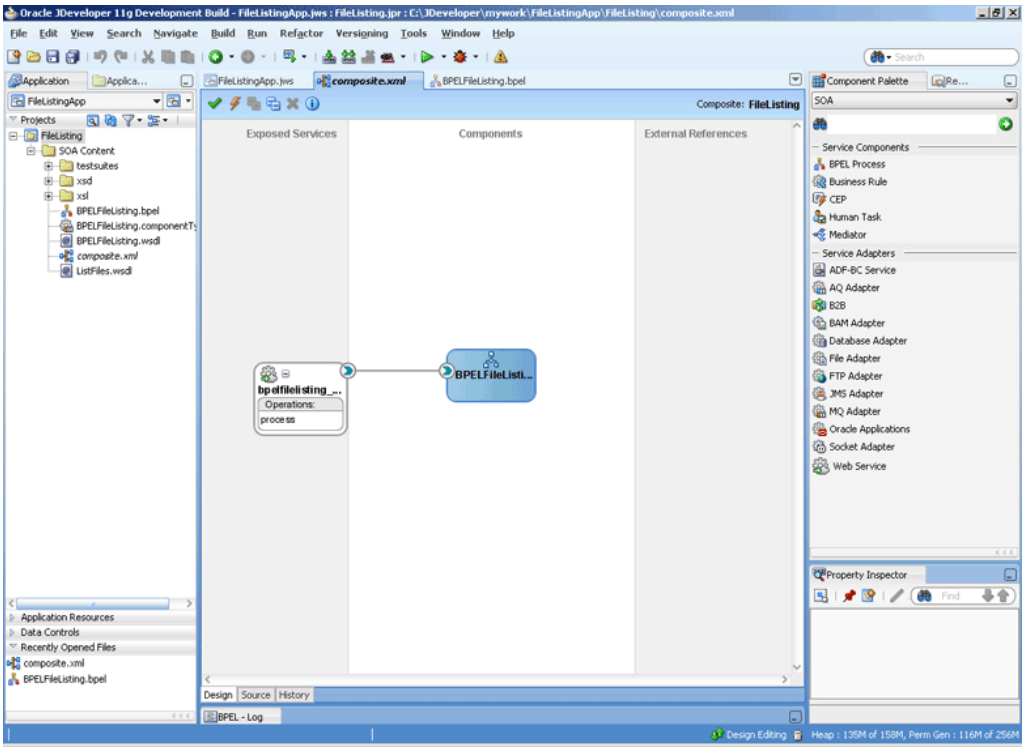
To perform Oracle File Adapter Listing, you require `*.txt` files. You must create and save the `*.txt` files in the target directory.

4.5.7.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `FileListingApp` in the **Application Name** field, and click **Next**. The Create Generic Application - Name your project page is displayed.
3. Enter `FileListing` in the **Project Name** field.
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process - BPEL Process page is displayed.
7. Enter `BPELFileListing` in the **Name** field, select **One Way BPEL Process** from the Template box.
8. Click **OK**. The FileListingApp application and the FileListing project appears in the design area, as shown in Figure 4–131.

Figure 4–131 The JDeveloper - Composite.xml

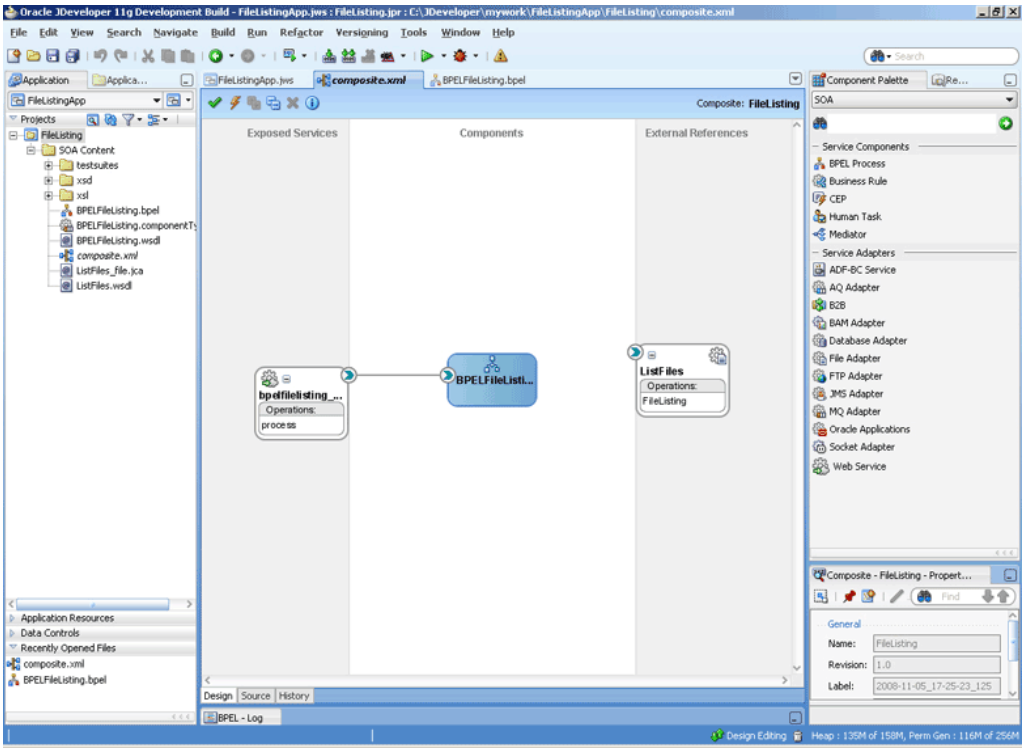


4.5.7.3 Creating the Outbound Oracle File Adapter Service

Perform the following steps to create an outbound Oracle File Adapter service to list the file from a target directory:

1. Drag and drop File Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `ListFiles` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **List Files**, enter `FileListing` in the **Operation Name** field, and then click **Next**. The File Directories page is displayed.
7. Enter the physical path for the input directory, as shown in Figure 4–55.
8. Click **Next**. The File Filtering page is displayed.
9. Enter `*.txt` in the **Include Files with Name Pattern** field.
10. Click **Next**. The Finish page is displayed.
11. Click **Finish**. The outbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–132.

Figure 4–132 The JDeveloper - Composite.xml



12. Click **File**, **Save All**.

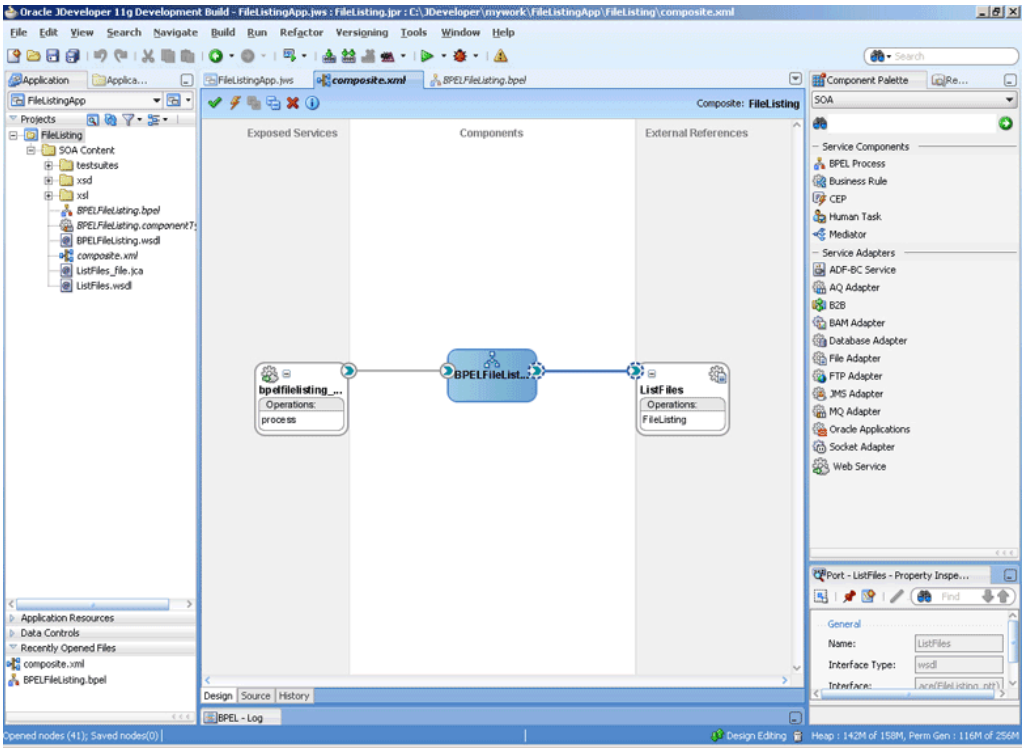
4.5.7.4 Wiring Services and Activities

You have to assemble or wire the two components that you have created: BPEL process, and the Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in ListFiles in the External References area.

The JDeveloper Composite.xml appears, as shown in Figure 4–133.

Figure 4–133 The JDeveloper - Composite.xml

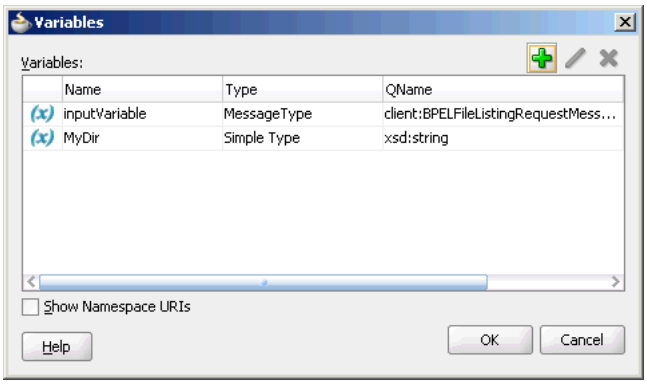


2. Click **File**, **Save All**.

Create a String Variable

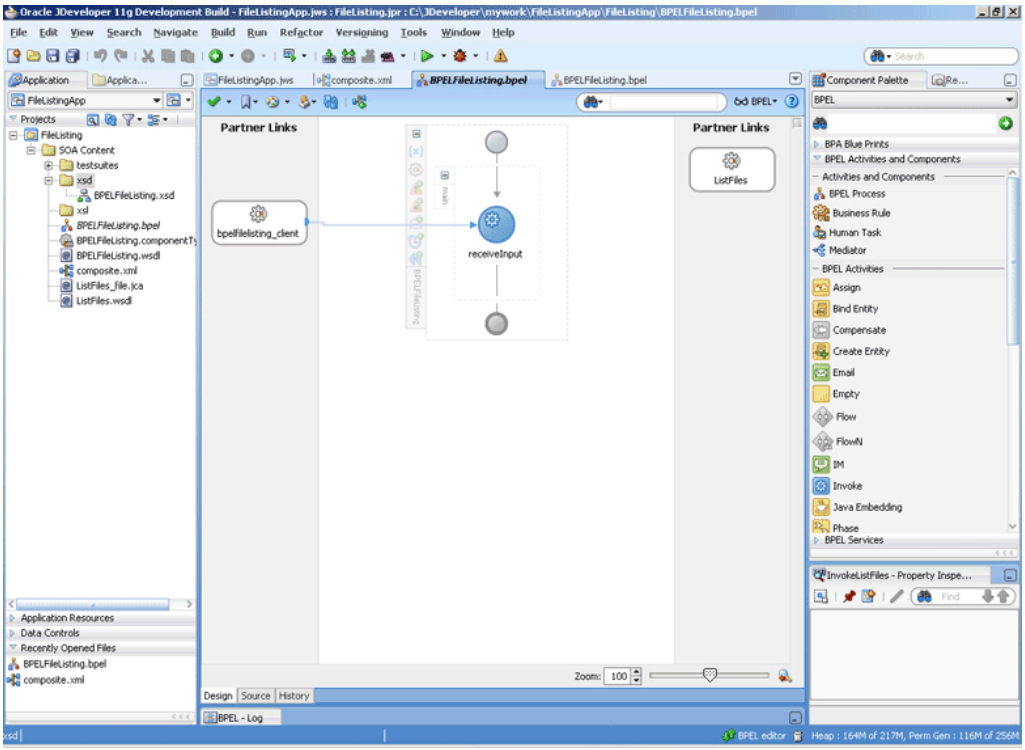
1. Double-click **BPELFileListing**. The BPELFileListing.bpel page is displayed.
2. Click the **Variables...** icon represented by (x). The Variables dialog is displayed.
3. Click the **Create...** icon. The Create Variable dialog is displayed.
4. Create a variable, `MyDir` of type `xsd:string`, as shown in Figure 4–134, for later use.

Figure 4–134 The Variables Dialog



5. Click **OK**. The JDeveloper BPELFileListing.bpel page appears, as shown in Figure 4–135

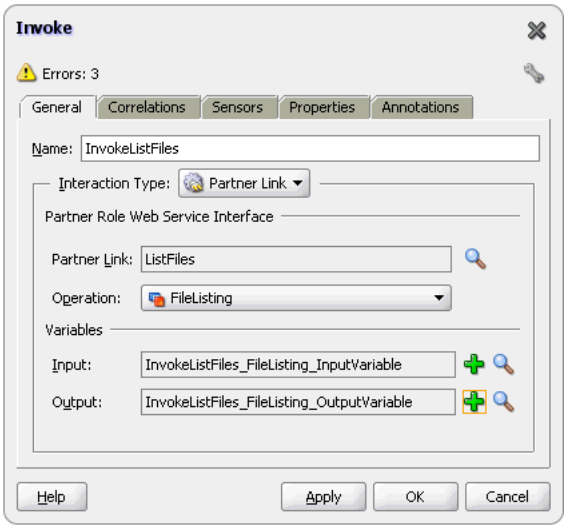
Figure 4–135 The JDeveloper - BPELFileListing.bpel



Add an Invoke Activity

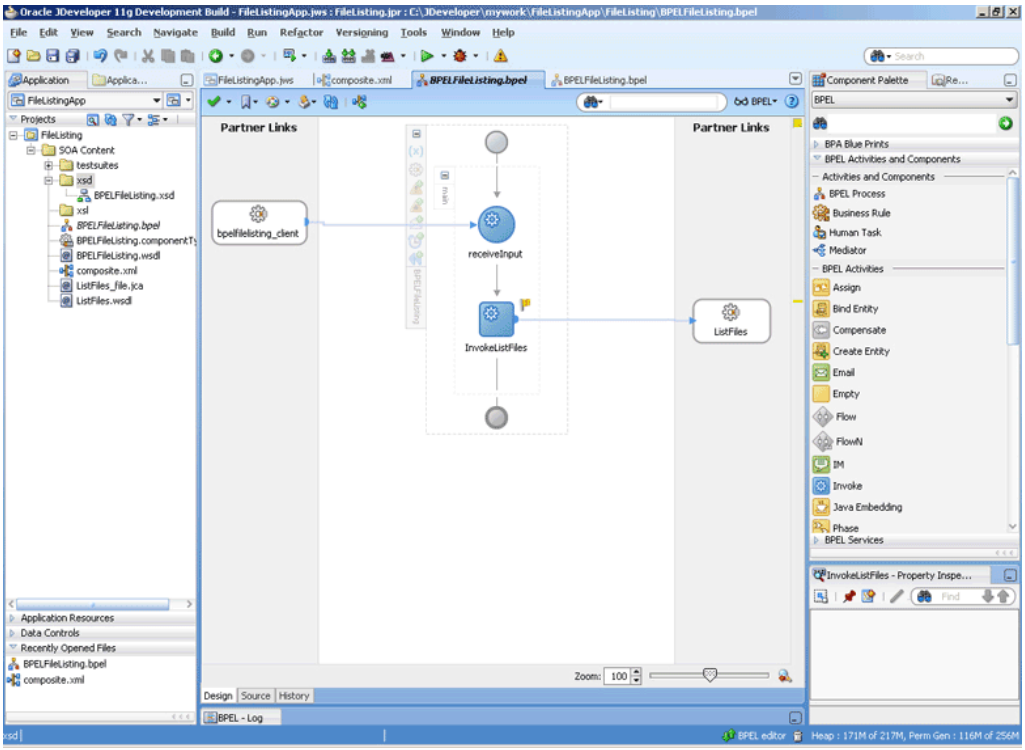
1. Drag and drop an **Invoke** activity below the receive Activity from the Component Palette to the design area.
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `InvokeListFiles` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **ListFiles**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name. The Invoke dialog is displayed with input variable populated.
8. Repeat the same to select the output variable. The Invoke dialog is displayed, as shown in Figure 4–136.

Figure 4–136 The Invoke Dialog



9. Click the **Properties** tab. The properties and the corresponding value column is displayed.
10. Select jca.file.Directory property. Double-click in the corresponding value column. The Adapter Property Value dialog is displayed.
11. Click the **Browse Variables** icon. The Variable XPath Builder dialog is displayed.
12. Expand **Variables**, select **MyDir**, and then click **OK**. The value of the `jca.file.Directory` is set to `Mydir`.
13. Click **OK**. The JDeveloper BPELFileListing.bpel page appears, as shown in Figure 4–137.

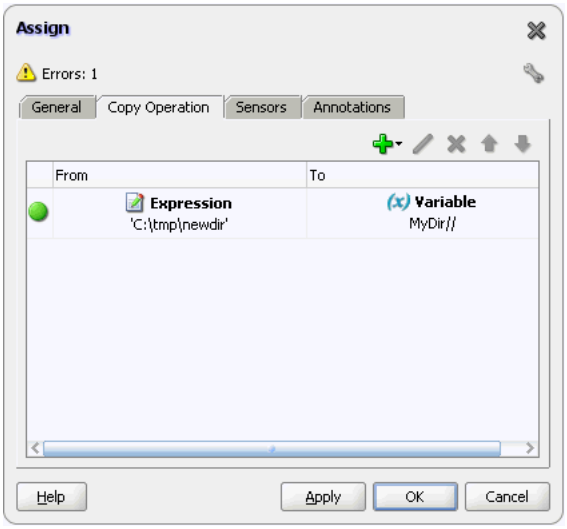
Figure 4–137 The JDeveloper - BPELFileListing.bpel



Add an Assign Activity

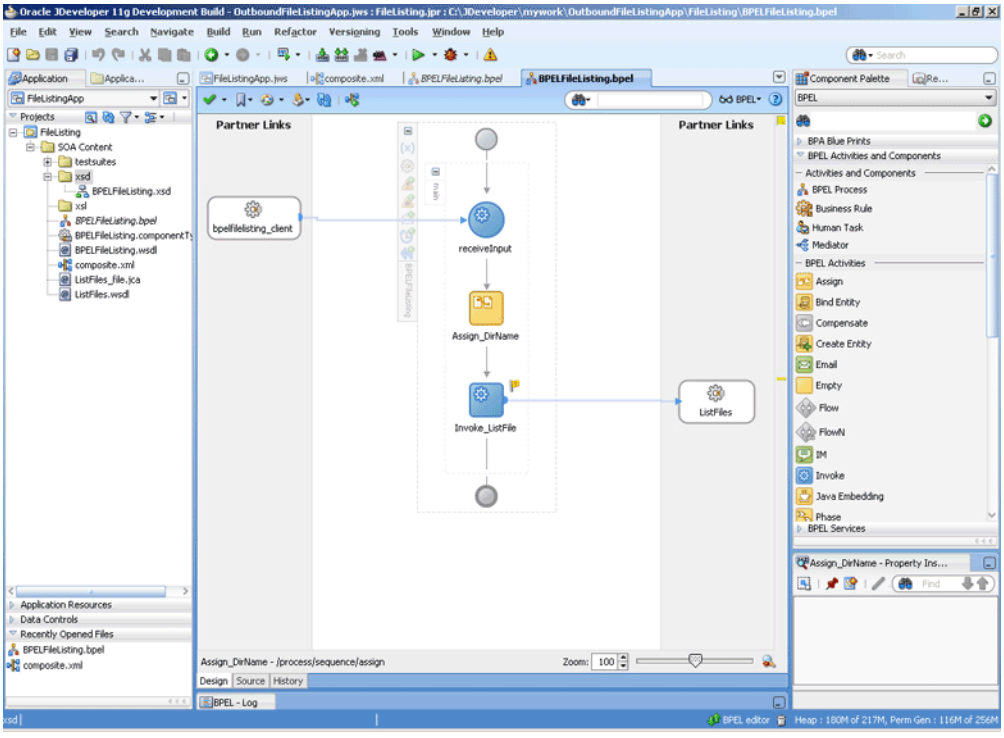
1. Drag and drop an **Assign** activity from the Component Palette in between the Receive activities and the Invoke activity in the design area.
2. Double-click the **Assign** activity. The Assign dialog is displayed.
3. Enter `AssignDirName` in the **Name** field.
4. Click the **Copy Operation** tab. The Assign dialog is displayed.
5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
6. Set the values for the headers, as shown in Figure 4–138.

Figure 4–138 The Assign Dialog



7. Click **OK**, the JDeveloper BPELFileListing.bpel page is displayed, as shown in Figure 4–139.

Figure 4–139 The JDeveloper - BPELFileListing.bpel



8. Click **File**, **Save All**.

4.5.7.5 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application. For more information, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

4.5.7.6 Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the `*.txt` files to the input directory (see Section 4.5.7.1, "Prerequisites" for details) and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the Recent Instances area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The Flow Trace page is displayed.
6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow. Additionally, click an activity to view the details of an activity.

4.5.8 Oracle File Adapter Complex Structure

This use case demonstrates the ability of the Oracle File Adapter to process native data defined in a custom format. In this sample, the custom format represents an invoice defined in `invoice-nxsd.xsd`. The Oracle File Adapter processes the `invoice.txt` file and publishes this to the ComplexStructure BPEL process. This is then transformed to a PurchaseOrder and written out as an xml file.

This use case includes the following sections:

- Section 4.5.8.1, "Prerequisites"
- Section 4.5.8.2, "Designing the SOA Composite"
- Section 4.5.8.3, "Creating the Inbound Oracle File Adapter Service"
- Section 4.5.8.4, "Creating the Outbound Oracle File Adapter Service"
- Section 4.5.8.5, "Wiring Services and Activities"
- Section 4.5.8.6, "Deploying with JDeveloper"
- Section 4.5.8.7, "Monitoring Using Fusion Middleware Control Console"

4.5.8.1 Prerequisites

To perform the complex structure business process, you require the following files from the `artifacts.zip` file contained in the `Adapters-104FileAdapterComplexStructure` sample:

- `artifacts/schemas/invoice-nxsd.xsd`

- `artifacts/schemas/po.xsd`
- `artifacts/xsl/InvToPo.xsl`
- `artifacts/input/invoice.txt`

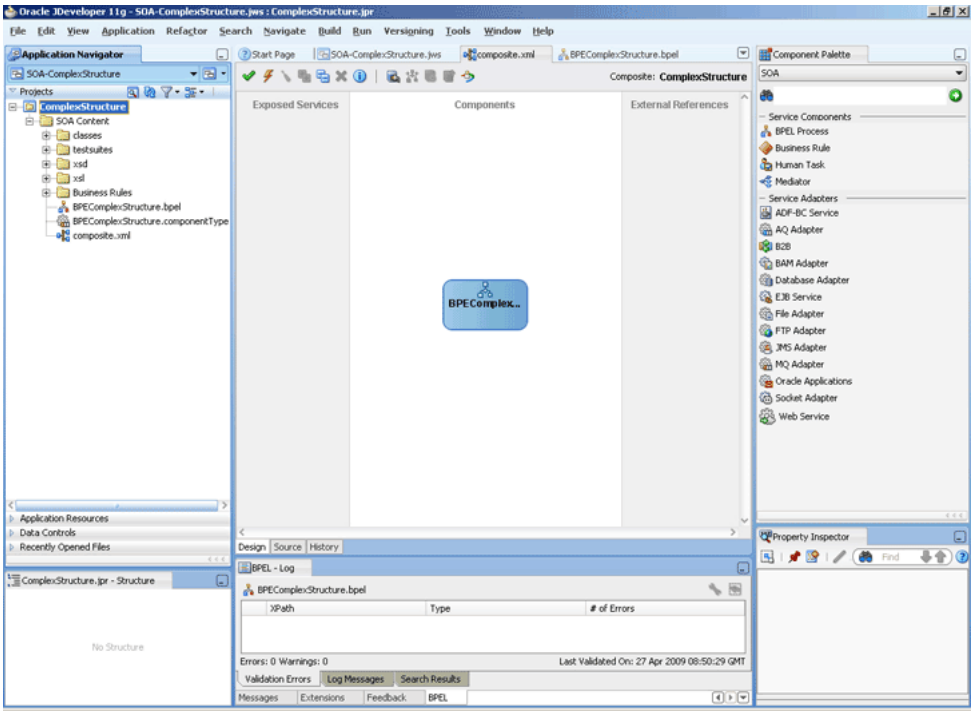
You can obtain the `Adapters-104FileAdapterComplexStructure` sample by accessing the Oracle SOA Sample Code site.

4.5.8.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `SOA-ComplexStructure` in the **Application Name** field, and click **Next**. The Create Generic Application - Name your project page is displayed.
3. Enter `ComplexStructure` in the **Project Name** field.
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process - BPEL Process page is displayed.
7. Enter `BPEComplexStructure` in the **Name** field, select **Define Service Later** from the Template box.
8. Click **OK**. The SOA-ComplexStructure application and the ComplexStructure project appears in the design area, as shown in Figure 4–140.

Figure 4–140 The JDeveloper - Composite.xml



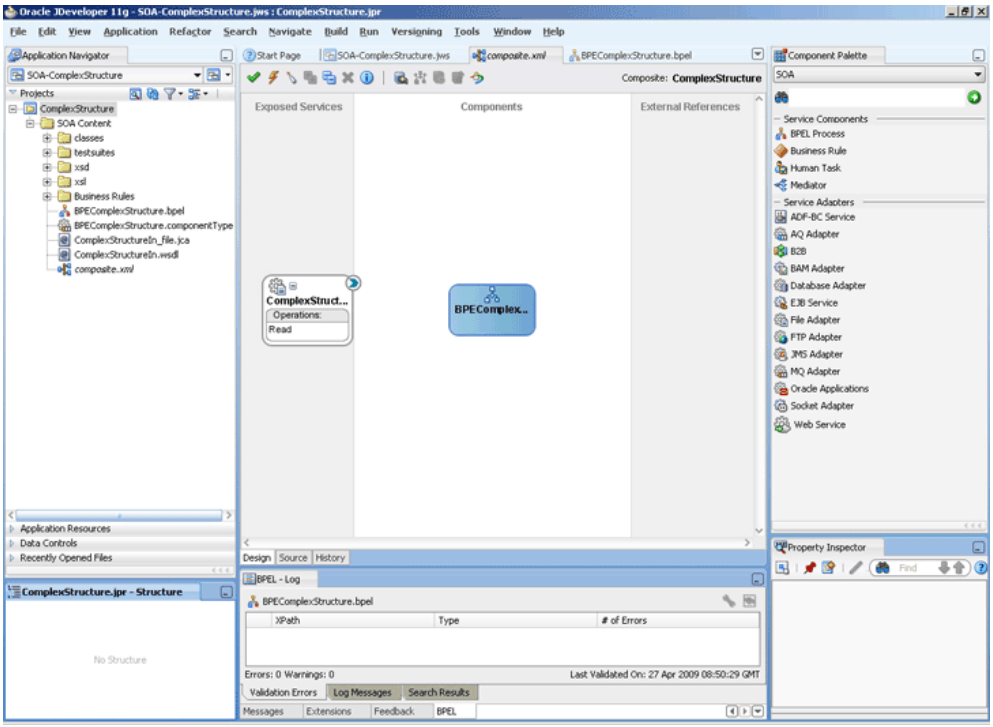
9. Copy the **invoice-nxsd.xsd** and **po.xsd** files to the schema directory in your project (see Section 4.5.8.1, "Prerequisites" for the location of these files).
10. Copy **InvToPo.xsl** to the xsl directory of your project (see Section 4.5.8.1, "Prerequisites" for the location of this file).

4.5.8.3 Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory:

1. Drag and drop File Adapter from the Component Palette to the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `Complex Structure In` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Read File**, and click **Next**. The File Directories page is displayed.
7. Enter the physical path for the input directory and click **Next**. The File Filtering page is displayed.
8. Enter `*.txt` in the **Include Files With Name Pattern** field, click **Next**. The File Polling page is displayed.
9. Click **Next**. The Messages page is displayed.
10. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
11. Click **Project Schema Files**, **invoice-nxsd.xsd**, and **invoice**.
12. Click **OK**. The URL field in the Messages page is populated with the invoice-nxsd.xsd file.
13. Click **Next**. The Finish page is displayed.
14. Click **Finish**. The inbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–141.

Figure 4–141 The JDeveloper - Composite.xml

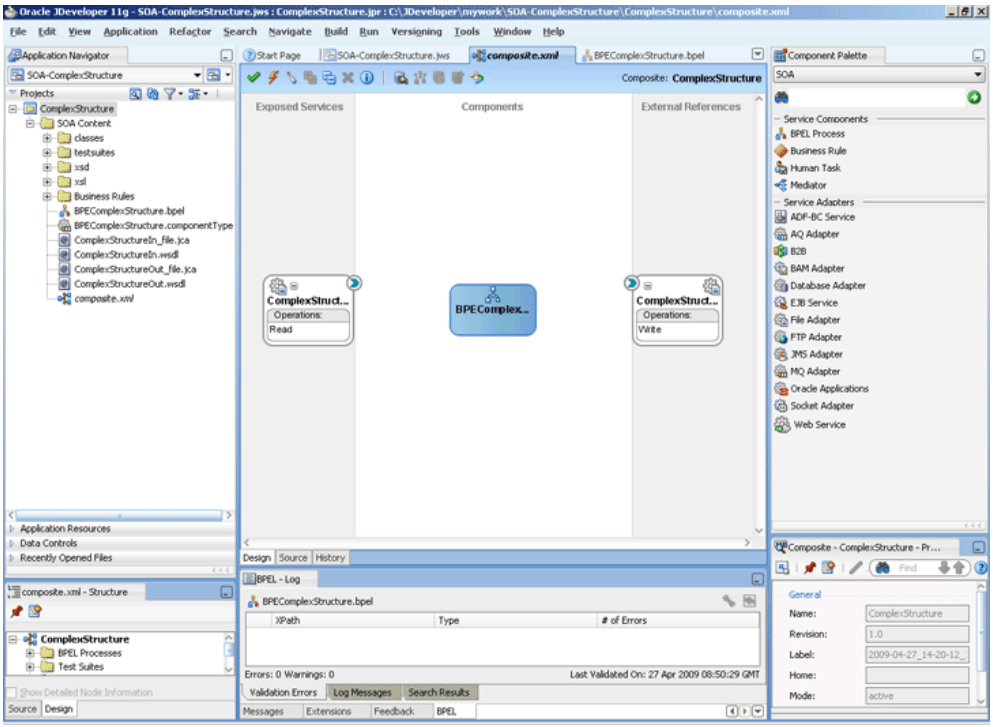


4.5.8.4 Creating the Outbound Oracle File Adapter Service

Perform the following steps to create an outbound Oracle File Adapter service to write the file from a local directory to the FTP server:

1. Drag and drop the Oracle File Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `ComplexStructureOut` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, and click **Next**. The File Configuration page is displayed.
7. Enter the physical path for the output directory and enter `invoice_%SEQ%.txt` in the **File Naming Convention(po_%SEQ%.txt)** field.
8. Click **Next**. The Messages page is displayed.
9. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
10. Click **Project Schema Files**, **po.xsd**, and **po**.
11. Click **OK**. The URL field in the Messages page is populated with the po.xsd file.
12. Click **Next**. The Finish page is displayed.
13. Click **Finish**. The outbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–142.

Figure 4–142 The JDeveloper - Composite.xml



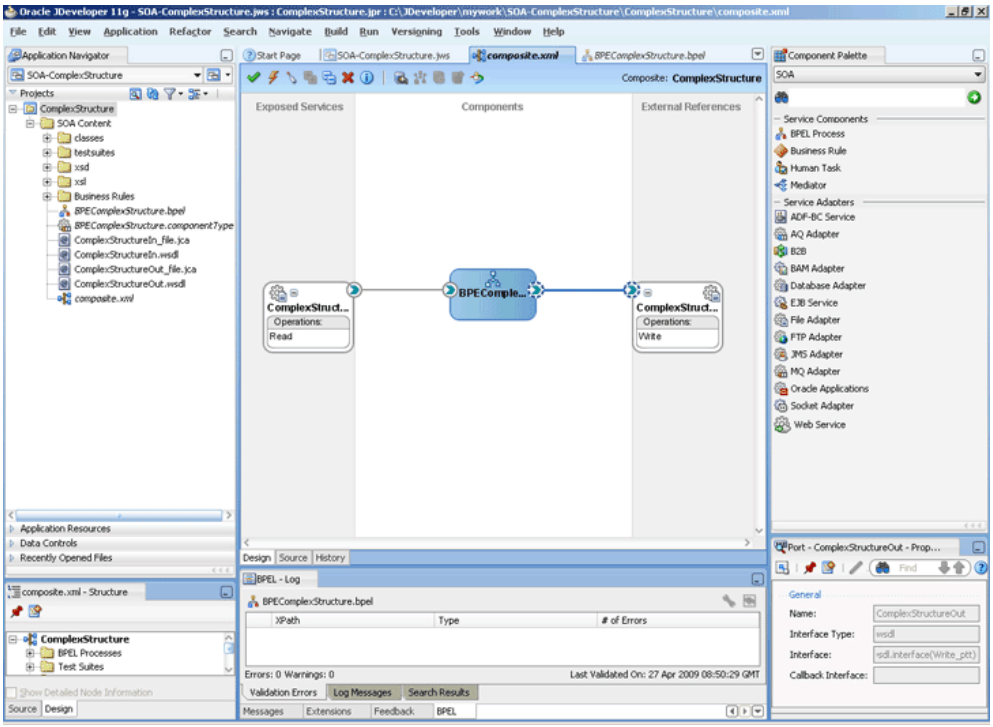
4.5.8.5 Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the ComplexStructureIn service in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the ComplexStructureOut reference in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 4–143.

Figure 4–143 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.

Add a Receive Activity

1. Double-click **BPELComplexStructure**. The BPELComplexStructure.bpel page is displayed.
2. Drag and drop a **Receive** activity from the Component Palette to the design area.
3. Double-click the **Receive** activity. The Receive dialog is displayed.
4. Enter `ReceiveInvoice` in the **Name** field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
6. Select **ComplexStructureIn**, and click **OK**.
7. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog. The Create Variable dialog is displayed.
8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper BPELComplexStructure.bpel page appears.

Add an Invoke Activity

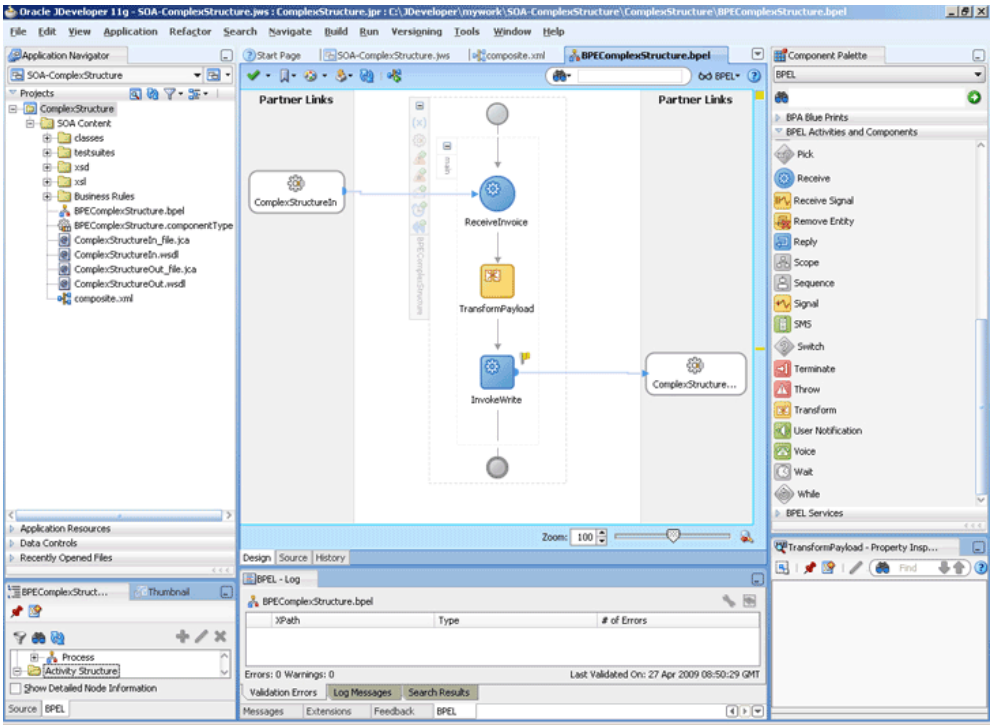
1. Drag and drop an **Invoke** activity from the Component Palette to the design area.
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `InvokeWrite` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.

5. Select **ComplexStructureOut**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.
7. Enter `InvokeWrite_Write_OutputVariable` in the variable name field and click **OK**. The Invoke dialog is displayed.
8. Click **OK**. The JDeveloper BPELComplexStructure.bpel page appears.

Add a Transform Activity

1. Drag and drop a **Transform** activity from the Component Palette in between the Receive and Invoke activities in the design area.
2. Double-click the **Transform** activity. The Transform dialog is displayed.
3. Enter `TransformPayload` in the **Name** field.
4. Click the **Transformation** tab. The Transform dialog is displayed.
5. Click the **Create...** icon. The Source Variable dialog is displayed.
6. Select **ReceiveInvoice_Read_InputVariable** in the Source Variable box, and select **body** in the Source Part box, and then click **OK**. The Transform dialog is displayed with the Source and Part selected.
7. Select **InvokeWrite_Write_OutputVariable** in the Target Variable list, select **body** in the Target Part.
8. Click the **Browse Mapping** icon at the end of the Mapper File field and select **InvToPo.xsl** file from the xsl directory in your project.
9. Click **OK**.
10. Click **File**, **Save All**. The BPELComplexStructure.bpel page is displayed, as shown in Figure 4–144.

Figure 4–144 The JDeveloper - BPELComplexStructure.bpel



4.5.8.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application. For more information, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

4.5.8.7 Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the `invoice.txt` file to the input directory (see Section 4.5.8.1, "Prerequisites" for the location of this file) and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the Recent Instances area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The Flow Trace page is displayed.
6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.

8. Click the **Flow** tab to view the process flow.
9. Click **ReceiveInvoice** to display the activity details

4.5.9 Oracle FTP Adapter Debatching

This is an Oracle FTP Adapter feature that debatches a large XML document into smaller individual XML fragments. This use case demonstrates how the debatching business process sample uses the Oracle FTP Adapter to process a file containing a batch of business records such as one or more invoice and purchase orders. The PurchaseOrders (POs) are then debatched and written to separate output files.

This use case includes the following sections:

- Section 4.5.9.1, "Prerequisites"
- Section 4.5.9.2, "Designing the SOA Composite"
- Section 4.5.9.3, "Creating the Inbound Oracle FTP Adapter Service"
- Section 4.5.9.4, "Creating the Outbound Oracle FTP Adapter Service"
- Section 4.5.9.5, "Wiring Services and Activities"
- Section 4.5.9.6, "Deploying with JDeveloper"
- Section 4.5.9.7, "Monitoring Using Fusion Middleware Control Console"

4.5.9.1 Prerequisites

To perform the complex structure business process, you require the following files from the `artifacts.zip` file contained in the `Adapters-101FTPAdapterDebatching` sample:

- `artifacts/schemas/container.xsd`
- `artifacts/schemas/po.xsd`
- `artifacts/xsl/InvToPo.xsl`
- `artifacts/xsl/PoToPo.xsl`
- `artifacts/input/container.txt`

You can obtain the `Adapters-101FTPAdapterDebatching` sample by accessing the Oracle SOA Sample Code site.

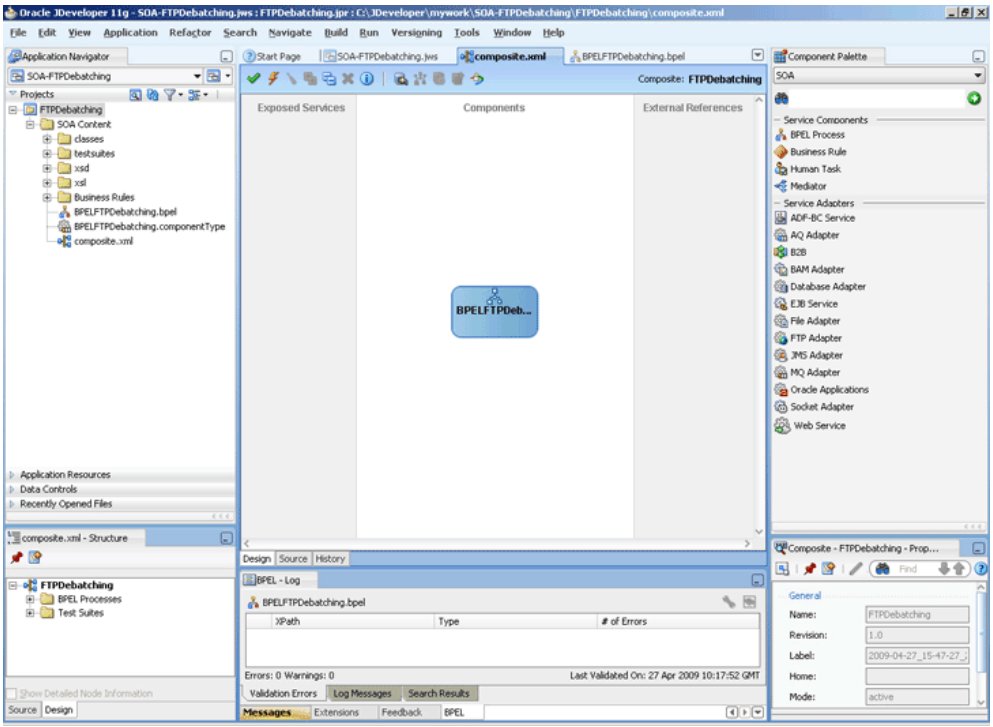
4.5.9.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `SOA-FTPDebatching` in the **Application Name** field, and click **OK**. The Create Generic Application - Name your project page is displayed.
3. Enter `FTPDebatching` in the **Project Name**.
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process - BPEL Process page is displayed.

7. Enter `BPELFTPDebatching` in the **Name** field, select **Define Service Later** from the Template box.
8. Click **OK**. The SOA-FTPDebatching application and the FTPDebatching project appears in the design area, as shown in Figure 4–145.

Figure 4–145 The JDeveloper - Composite.xml



9. Copy the `container.xsd` and `po.xsd` files to the `xsd` directory of your project (see Section 4.5.9.1, "Prerequisites" for the location of these files).
10. Copy the `InvToPo.xsl` and `PoToPo.xsl` files to the xsl directory of your project (see Section 4.5.9.1, "Prerequisites" for the location of these files).

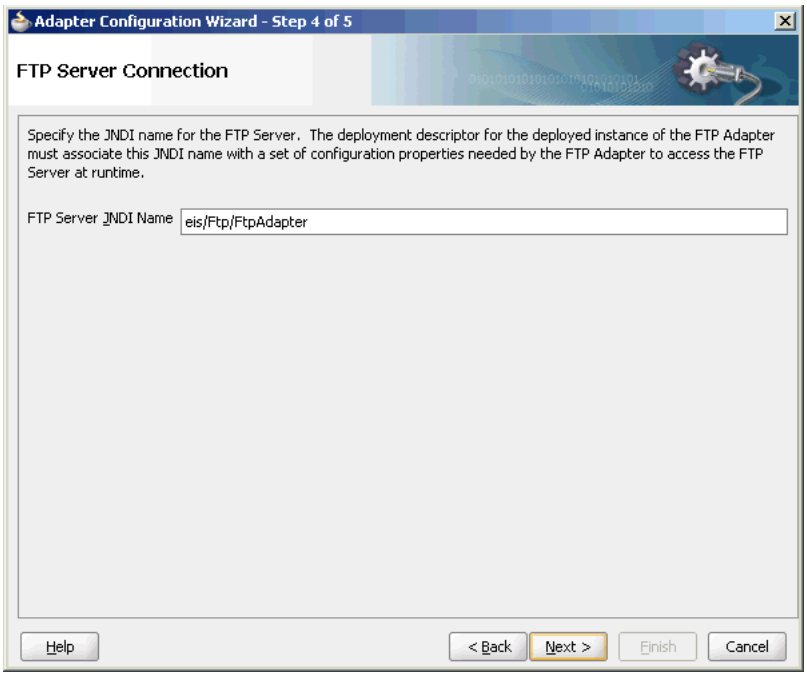
4.5.9.3 Creating the Inbound Oracle FTP Adapter Service

Perform the following steps to create an inbound Oracle FTP Adapter service to read the file from a local directory:

1. Drag and drop the Oracle FTP Adapter from the Component Palette to the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `FTPDebatchingIn` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The FTP Server Connection page is displayed, as shown in Figure 4–146.

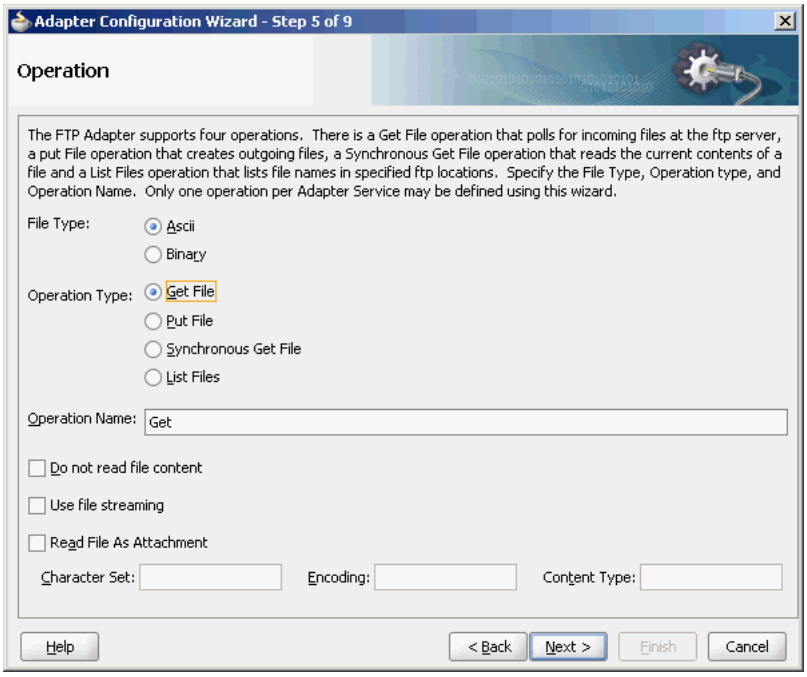
> **Note:** Ensure that you have configured the jndi-name in the deployment descriptor for Oracle FTP Adapter before deploying this application.

Figure 4–146 The Adapter Configuration Wizard FTP Server Connection Page



6. Click **Next**. The Operation page is displayed.
7. Select **Get File**, as shown in Figure 4–147, and click **Next**. The File Directories page is displayed.

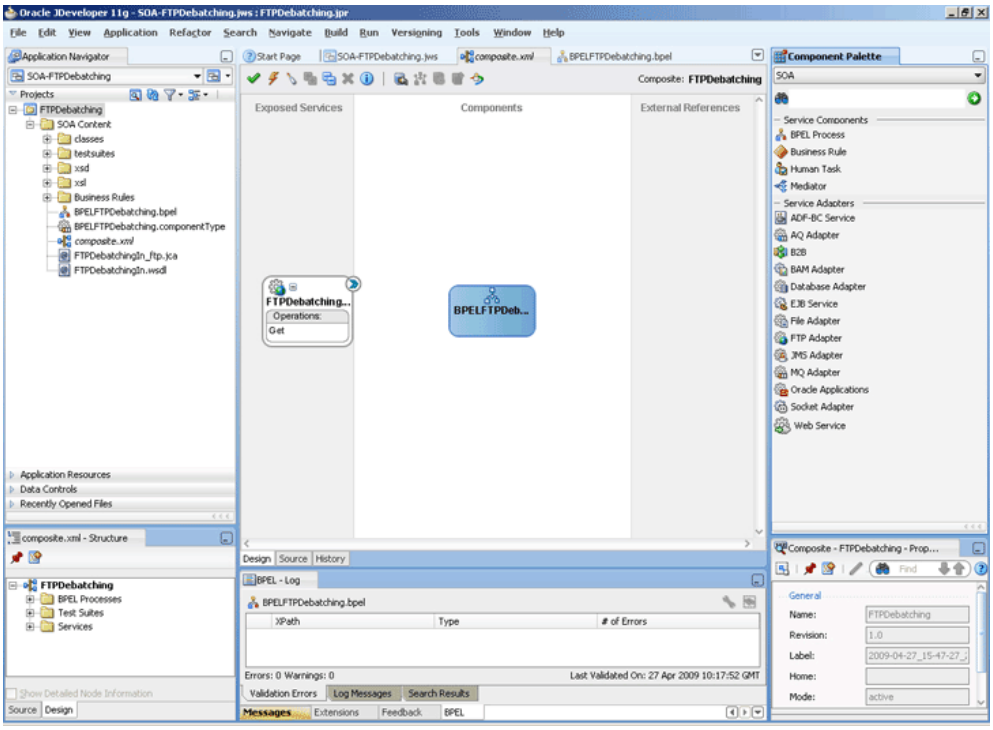
Figure 4–147 The Adapter Configuration Wizard Operation Page



8. Enter the physical path for the input directory, and click **Next**. The File Filtering page is displayed.

9. Enter `*.txt` in the **Include Files With Name Pattern** field, select **Files Contain Multiple Messages** check box, specify `1` as the value for Publish Messages in Batches Of box.
10. Click **Next**. The File Polling page is displayed.
11. Click **Next**. The Messages page is displayed.
12. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
13. Click **Project Schema Files**, **container.xsd**, and **container**.
14. Click **OK**. The URL field in the Messages page is populated with the **container.xsd** file.
15. Click **Next**. The Finish page is displayed.
16. Click **Finish**. The inbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–148.

Figure 4–148 The JDeveloper - Composite.xml



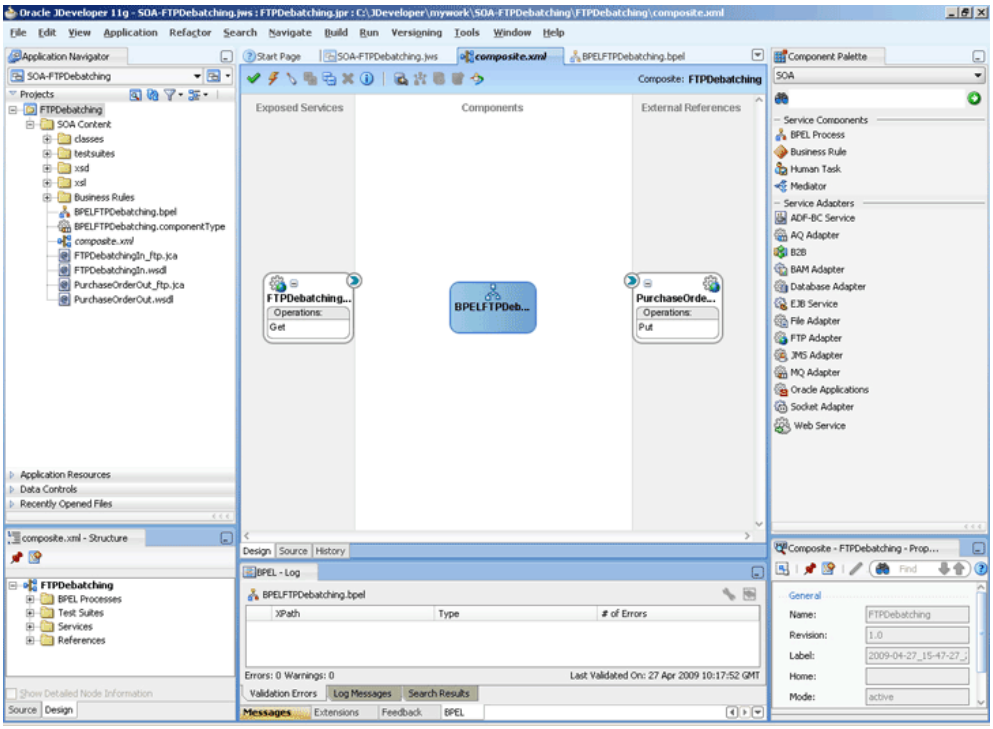
4.5.9.4 Creating the Outbound Oracle FTP Adapter Service

Perform the following steps to create an outbound Oracle FTP Adapter service to write the file from a local directory to the FTP server:

1. Drag and drop the FTP Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `PurchaseOrderOut` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.

5. Select **Define from operation and schema (specified later)**, and click **Next**. The FTP Server Connection page is displayed.
6. Click **Next**. The Operation page is displayed.
7. Select **Put File**, and click **Next**. The File Configuration page is displayed.
8. Enter the physical path for the output directory and enter `po_%SEQ%.txt` in the **File Naming Convention(po_%SEQ%.txt)** field.
9. Select **Number of Messages Equals** option, if not selected. The default value is `1`.
10. Click **Next**. The Messages page is displayed.
11. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
12. Click **Project Schema Files**, **po.xsd**, and **po**.
13. Click **OK**. The URL field in the Messages page is populated with the po.xsd file.
14. Click **Next**. The Finish page is displayed.
15. Click **Finish**. The outbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–149.

Figure 4–149 The JDeveloper - Composite.xml



4.5.9.5 Wiring Services and Activities

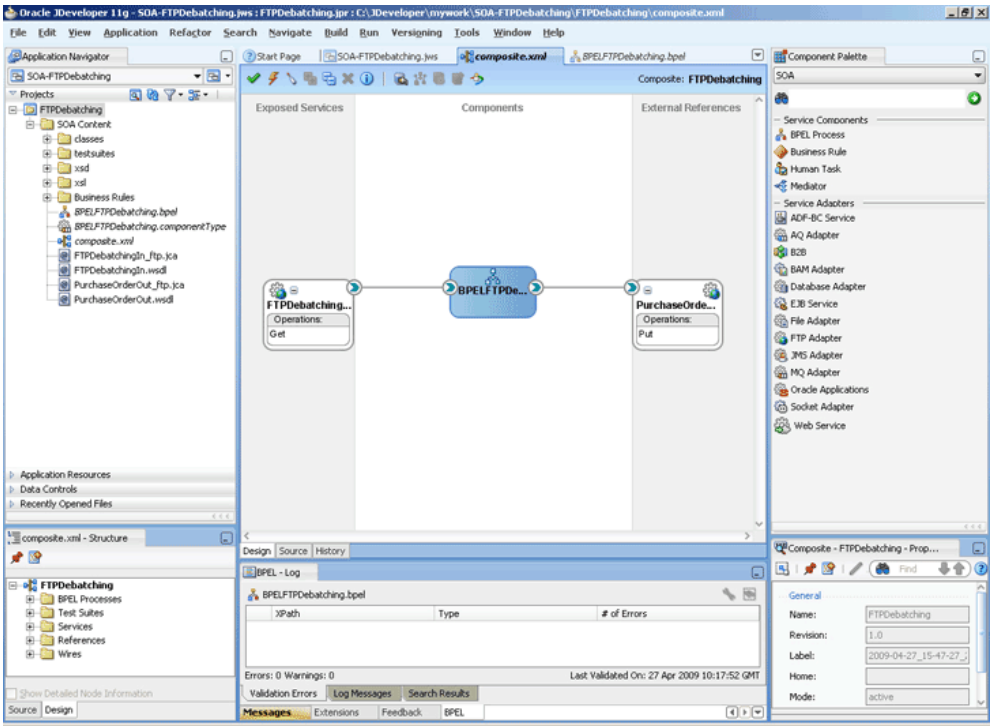
You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the FTPDebatchingIn service in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.

2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the PurchaseOrderOut reference in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 4–150.

Figure 4–150 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.

Add a Receive Activity

1. Double-click **BPELFTPDebatching**. The `BPELFTPDebatching.bpel` page is displayed.
2. Drag and drop a **Receive** activity from the Component Palette to the design area.
3. Double-click the **Receive** activity. The Receive dialog is displayed.
4. Enter `Receive` in the **Name** field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
6. Select **FTPDebatchingIn**, and click **OK**.
7. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog. The Create Variable dialog is displayed.
8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper `BPELFTPDebatching.bpel` page appears with the Receive activity added.

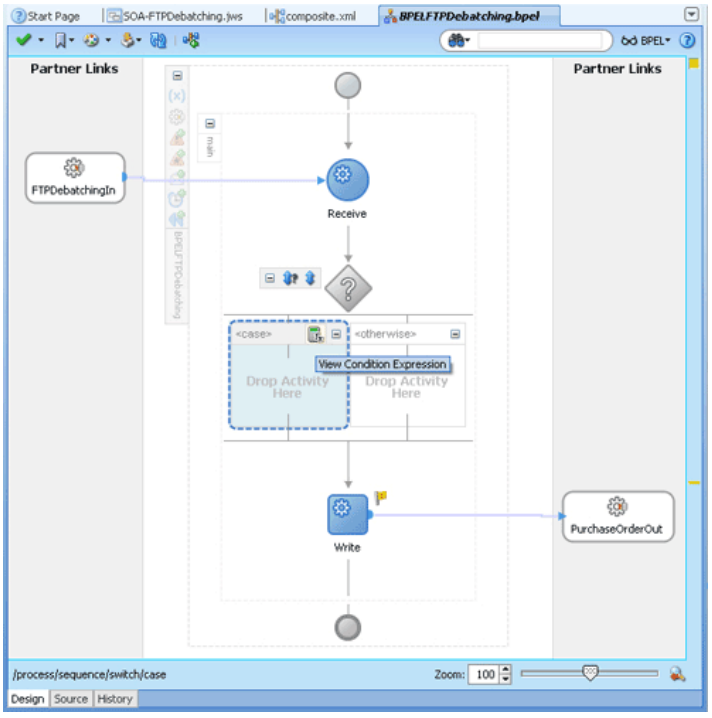
Add an Invoke Activity

1. Drag and drop an **Invoke** activity from the Component Palette to the design area.
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `Write` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **PurchaseOrderOut**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Receive dialog. The Create Variable dialog is displayed.
7. Enter `Write_Put_OutputVariable` in the Variable field and click **OK**. The Invoke dialog is displayed.
8. Click **OK**. The JDeveloper `BPELFTPDebatching.bpel` page appears with the invoke activity added.

Add a Switch Activity

1. Drag and drop a **Switch** activity from the Component Palette in between the Receive and Invoke activities in the design area.
2. Expand the **Switch** activity. This displays a screen to enter the values for <case> and <otherwise>.
3. In the `<case>` section, click the **View Condition Expression** icon, as shown in Figure 4–151. The Condition Expression pop-up window is displayed.

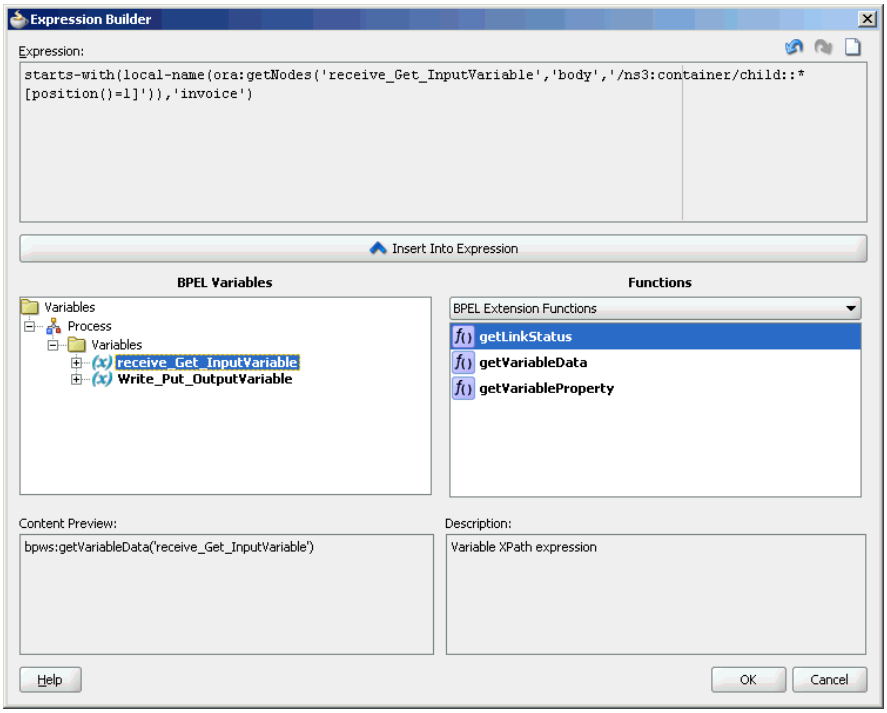
Figure 4–151 BPELFTPDebatching.bpel Page



4. Click the **Xpath Expression Builder** icon in the pop-up window. The Expression Builder dialog is displayed.

5. Enter `starts-with(local-name(ora:getNodes('receive_Get_InputVariable','body','/ns3:container/child::*[position()=1]' )),'invoice')` as the expression, as shown in Figure 4–152, and click **OK**. The screen returns to the Condition Expression pop-up window.

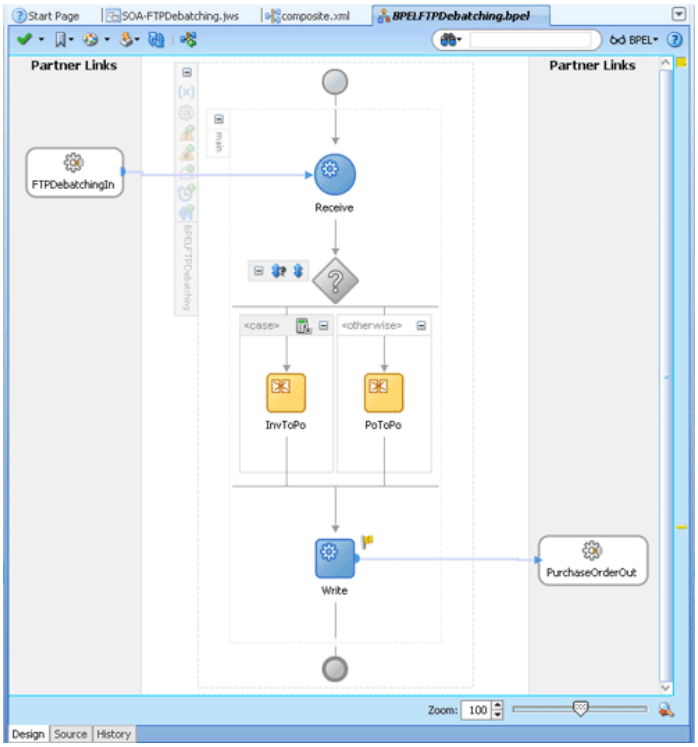
Figure 4–152 The Expression Builder Dialog



6. Add two transformation activities, one each for <case> and <otherwise> sections.
   a. Drag and drop a **Transform** activity in the <case> section.
   b. Double-click the **Transform** activity.
   c. Enter `InvToPo` in the **Name** field.
   d. Click the **Transformation** tab.
   e. Click the **Create...** icon. The Source Variable dialog is displayed.
   f. Accept the defaults and click **OK**.
   g. Select **Write_Put_OutputVariable** in the Target Variable list.
   h. Click the **Browse Mappings** icon at the end of the Mapper File field, and select the **InvToPo.xsl** file.
   i. Click **OK**.
   j. Repeat the same process for the second transformation. Select `PoToPo.xsl` as the Mapper File for this transform activity.

   The BPELFTPDebatching.bpel page is displayed, as shown in Figure 4–153.

Figure 4–153 The BPELFTPDebatching.bpel Page



7. Click **File**, **Save All**.

4.5.9.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application. For more information, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

4.5.9.7 Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the `container.txt` file to the input directory (see Section 4.5.9.1, "Prerequisites" for the location of this file) and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the Recent Instances area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The Flow Trace page is displayed.

6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow. Additionally, click an activity (such as invoke, receive) to view the details of an activity.

4.5.10 Oracle FTP Adapter Dynamic Synchronous Read

This use case demonstrates the ability of the Oracle FTP Adapter to perform a mid-process synchronous read operation using an Invoke activity. This use case illustrates the following adapter functionality:

- Oracle File Adapter (Read Operation)
- Oracle FTP Adapter (Synchronous Read operation)

  Ability to specify the file name to be read during run-time
- Oracle File Adapter (Write Operation)

The process is initiated by the presence of a trigger file appearing in a local directory monitored by the inbound Oracle File Adapter. The trigger file contains the name of the file to be read by the synchronous read operation. This file name is passed through headers to the adapter. This can be done using the Properties tab for the Invoke activity. This synchronous read file operation is performed against a remote directory on a FTP server. The result of the read is then transformed and written out to a local directory through the outbound Oracle File Adapter. This section includes the following topics:

- Section 4.5.10.1, "Prerequisites"
- Section 4.5.10.2, "Designing the SOA Composite"
- Section 4.5.10.3, "Creating the Inbound Oracle File Adapter Service"
- Section 4.5.10.4, "Creating the Outbound Oracle FTP Adapter Service"
- Section 4.5.10.5, "Wiring Services and Activities"
- Section 4.5.10.6, "Deploying with JDeveloper"
- Section 4.5.10.7, "Monitoring Using Fusion Middleware Control Console"

4.5.10.1 Prerequisites

To perform FTP Dynamic Synchoronous Read, you require the following files from the `artifacts.zip` file contained in the `Adapters-102FTPAdapterDynamicSynchronousRead` sample:

- `artifacts/schemas/address-csv.xsd`
- `artifacts/schemas/address-fixedLength.xsd`
- `artifacts/schemas/trigger.xsd`
- `artifacts/xsl/addr1Toaddr2.xsl`
- `artifacts/input/address_csv.txt`
- `artifacts/input/trigger.trg`

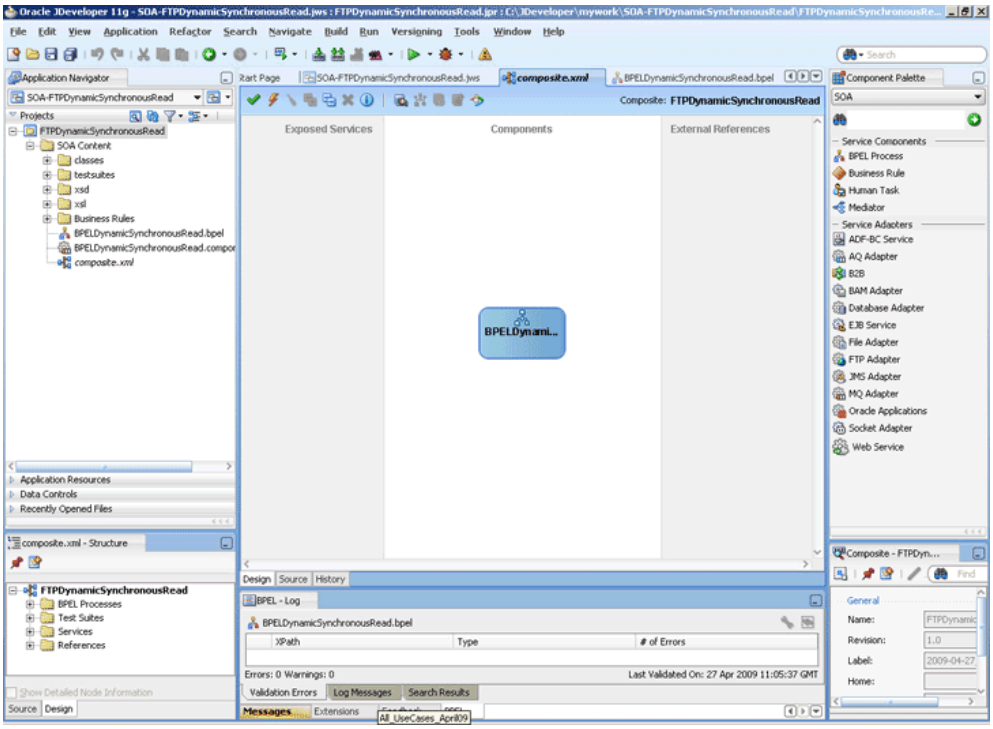
You can obtain the `Adapters-102FTPAdapterDynamicSynchronousRead` sample by accessing the Oracle SOA Sample Code site, and selecting the Adapters tab.

4.5.10.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `SOA-FTPDynamicSynchronousRead` in the **Application Name** field, and click **OK**. The Create Generic Application - Name your project page is displayed.
3. Enter `FTPDynamicSynchronousRead` in the **Project Name**.
4. In the Available list under the Project Technologies tab, double-click **SOA** to move it to the Selected list.
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process - BPEL Process page is displayed.
7. Enter `BPELDynamicSynchronousRead` in the **Name** field, select **Define Service Later** from the Template box.
8. Click **OK**. The SOA-FTPDynamicSynchronousRead application and FTPDynamicSynchronousRead project appears in the design area, as shown in Figure 4–154.

Figure 4–154 The JDeveloper - Composite.xml



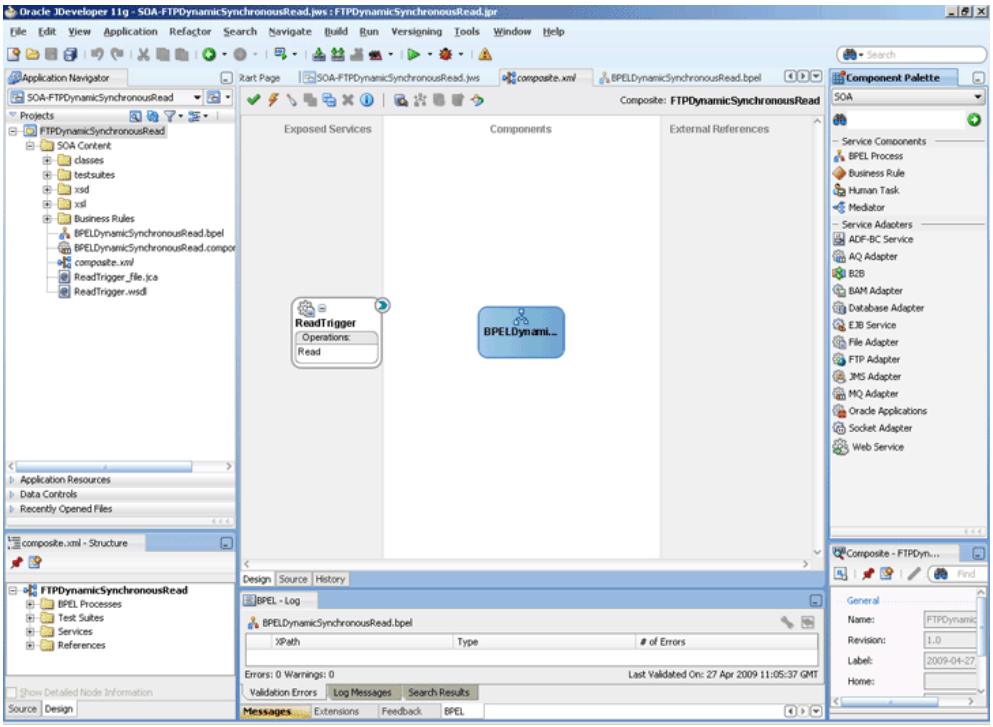
9. Copy the `address-csv.xsd`, `address-fixedLength.xsd`, and `trigger.xsd` files to the xsd directory of your project (see Section 4.5.10.1, "Prerequisites" for the location of these files).
10. Copy the `addr1Toaddr2.xsl` file to the xsl directory of your project (see Section 4.5.10.1, "Prerequisites" for the location of this file).

4.5.10.3 Creating the Inbound Oracle File Adapter Service

Perform the following steps to create an inbound Oracle File Adapter service to read the file from a local directory:

1. Drag and drop File Adapter from the Component Palette to the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `ReadTrigger` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Read File**, and click **Next**. The File Directories page is displayed.
7. Enter the physical path for the input directory and click **Next**. The File Filtering page is displayed.
8. Enter `*.trg` in the **Include Files With Name Pattern** field, click **Next**. The File Polling page is displayed.
9. Click **Next**. The Messages page is displayed.
10. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
11. Click **Project Schema Files**, **trigger.xsd**, and **trigger**.
12. Click **OK**. The URL field in the Messages page is populated with the trigger.xsd file.
13. Click **Next**. The Finish page is displayed.
14. Click **Finish**. The inbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–155.

Figure 4–155 The JDeveloper - Composite.xml



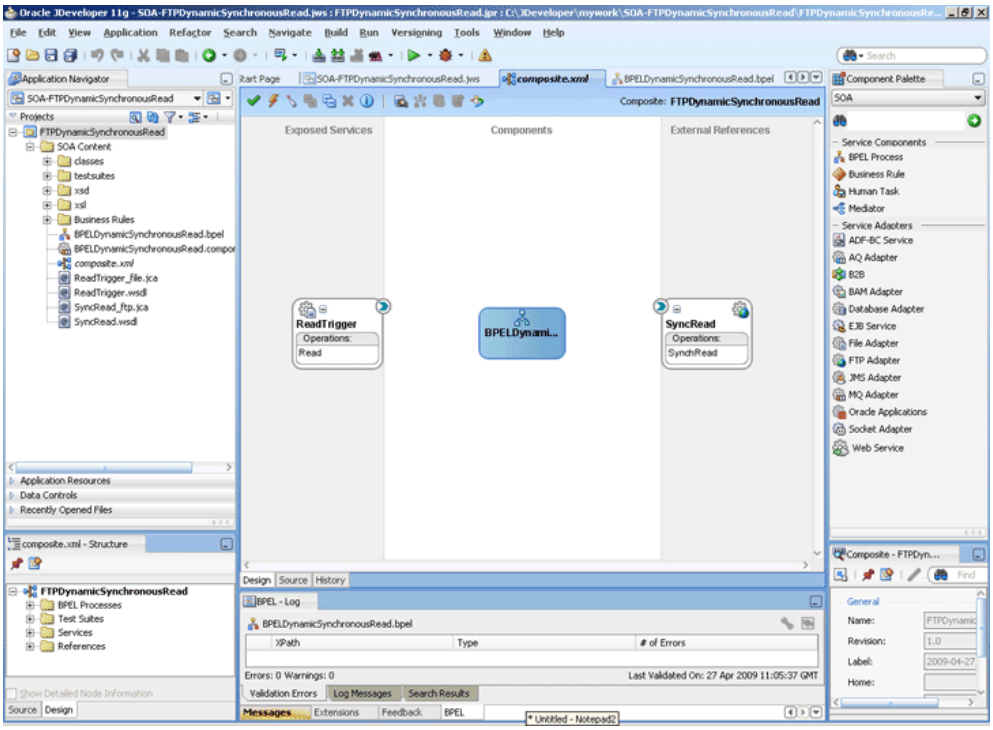
4.5.10.4 Creating the Outbound Oracle FTP Adapter Service

Perform the following steps to create an outbound Oracle FTP Adapter service to write the file from a local directory to the FTP server:

1. Drag and drop FTP Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `SyncRead` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The FTP Server Connection page is displayed.
6. Click **Next**. The Operation page is displayed.
7. Select **Synchronous Get File**, and click **Next**. The File Directories page is displayed.
8. Enter the physical path for the output directory.
9. Click **Next**. The File Name page is displayed.
10. Enter `dummy.txt` in the **File Name** field and click **Next**. The Messages page is displayed.
11. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
12. Click **Project Schema Files**, **address-csv.xsd**, and **Root-Element**.
13. Click **OK**. The URL field in the Messages page is populated with the address-csv.xsd file.

14. Click **Next**. The Finish page is displayed.
15. Click **Finish**. The outbound Oracle FTP Adapter is now configured and composite.xml appears, as shown in Figure 4–156.

Figure 4–156 The JDeveloper - Composite.xml

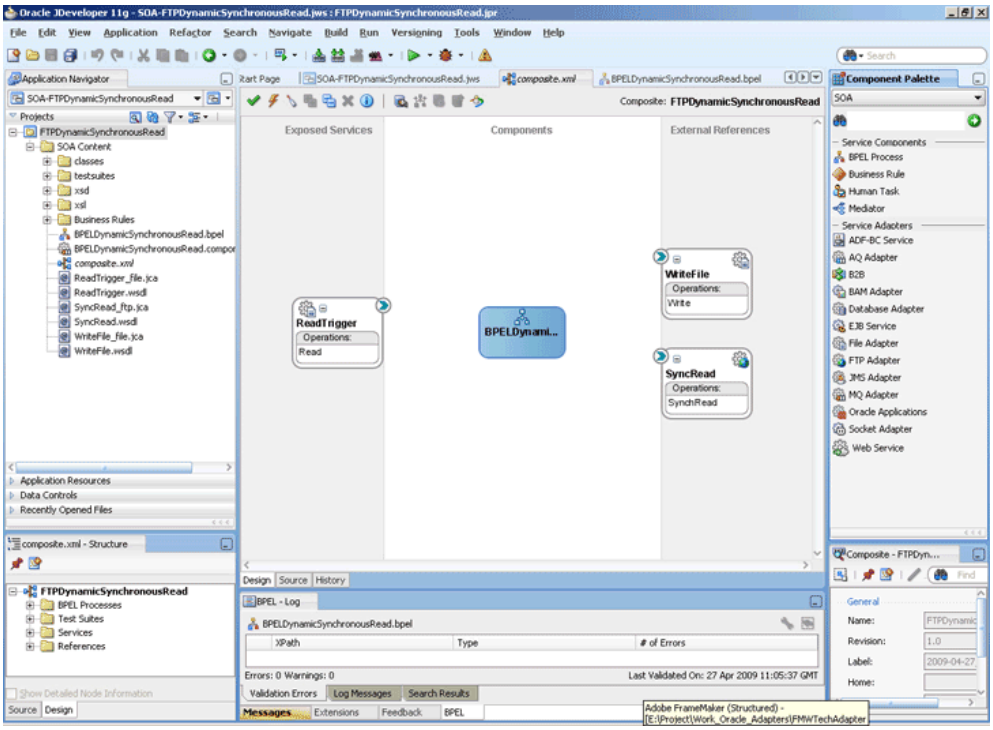


Add An Outbound Oracle File Adapter Service

1. Drag and drop the Oracle File Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `WriteFile` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Write File**, enter `Write` in the **Operation Name** field, and then click **Next**. The File Configuration page is displayed.
7. Enter the physical path for the output directory, enter `address_%SEQ%.txt` in the **File Naming Convention (po_%SEQ%.txt)**.
8. Click **Next**. The Messages page is displayed.
9. Click **Browse For Schema File** that appears at the end of the URL field. The Type Chooser dialog is displayed.
10. Click **Project Schema Files**, **address-fixedLength.xsd**, and **Root-Element**.
11. Click **OK**. The URL field in the Messages page is populated with the address-fixedLength.xsd file.

12. Click **Next**. The Finish page is displayed.
13. Click **Finish**. The outbound Oracle File Adapter is now configured and composite.xml appears, as shown in Figure 4–157.

Figure 4–157 The JDeveloper - Composite.xml



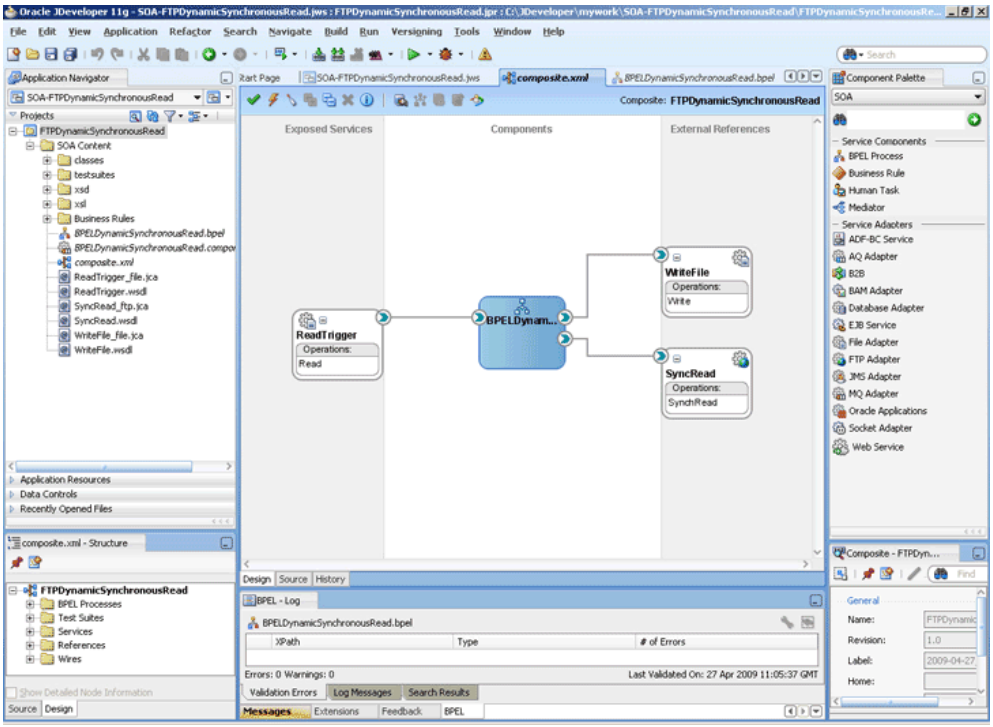
4.5.10.5 Wiring Services and Activities

You have to assemble or wire the four components that you have created: Inbound adapter service, BPEL process, two Outbound adapter references. Perform the following steps to wire the components:

1. Drag the small triangle in the ReadTrigger in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the SyncRead in the External References area and to the drop zone that appears as a green triangle in WriteFile.

   The JDeveloper Composite.xml appears, as shown in Figure 4–158.

Figure 4–158 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.

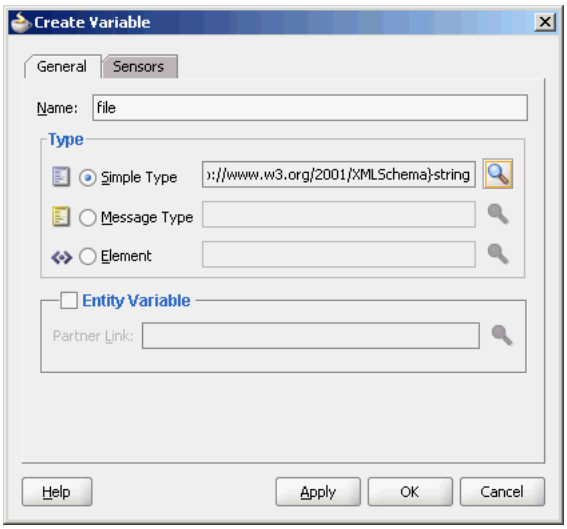
Add a Receive Activity

1. Double-click **BPELDynamicSynchronousRead**. The BPELDynamicSynchronousRead.bpel page is displayed.
2. Drag and drop a **Receive** activity from the Component Palette to the design area.
3. Double-click the **Receive** activity. The Receive dialog is displayed.
4. Enter `ReceiveTrigger` in the **Name** field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
6. Select **ReadTrigger**, and click **OK**.
7. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog. The Create Variable dialog is displayed.
8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper BPELDynamicSynchronousRead.bpel page appears.

Create a Variable and add an Invoke Activity

1. Click the **Variables...** icon represented by (x). The Variables dialog is displayed.
2. Click the **Create...** icon. The Create Variable dialog is displayed.
3. Create a variable called file of type xsd:string, as shown in Figure 4–159.

Figure 4–159 The Create Variable Dialog



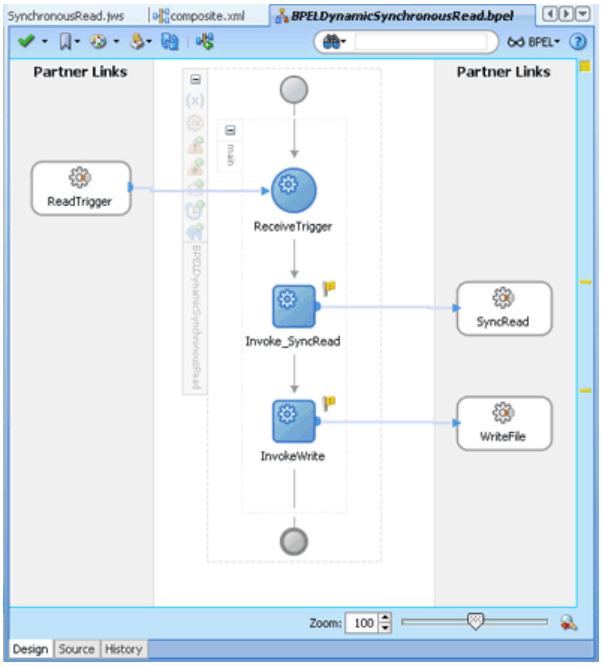
4. Click **OK** to return to BPELDynamicSynchronousRead.bpel page.
5. Drag and drop an **Invoke** activity from the Component Palette to the design area.
6. Double-click the **Invoke** activity. The Invoke dialog is displayed.
7. Enter `Invoke_SyncRead` in the **Name** field.
8. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
9. Select **SyncRead**, and click **OK**.
10. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.
11. Select the default variable name and click **OK**. The Input variable field is populated with the default variable name.
12. Repeat the same for the Output Variable field.
13. Click the **Properties** tab. The properties and the corresponding value column is displayed.
14. Select `jca.ftp.FileName` property. Double-click in the corresponding value column. The Adapter Property value dialog is displayed.
15. Click the **Browse variables** icon. The Variable XPath Builder dialog is displayed.
16. Expand **Variables**, select **file**, and then click **OK**. The value of the `jca.ftp.FileName` is set to `file`.
17. Click **OK**. The JDeveloper BPELDynamicSynchronousRead.bpel page appears.

Add another Invoke Activity

1. Drag and drop an **Invoke** activity from the Component Palette to the design area.
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `InvokeWrite` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.

5. Select **WriteFile**, and click **OK**.
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
8. Click **OK**. The JDeveloper BPELDynamicSynchronousRead.bpel page appears, as shown in *** 'The JDeveloper - BPELDynamicSynchronousRead.bpel Page' on page 201 ***.

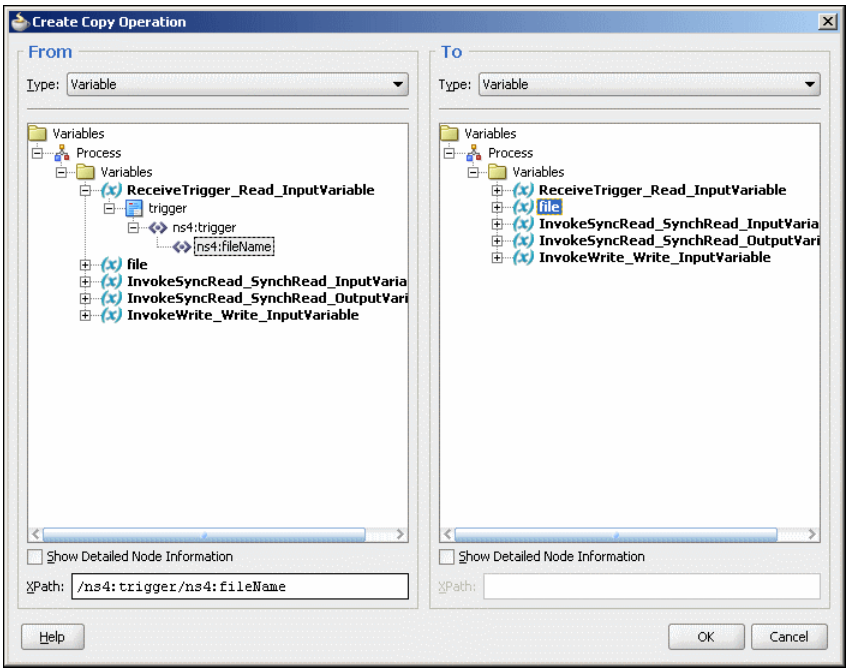
Figure 4–160 The JDeveloper - BPELDynamicSynchronousRead.bpel Page



Add an Assign Activity

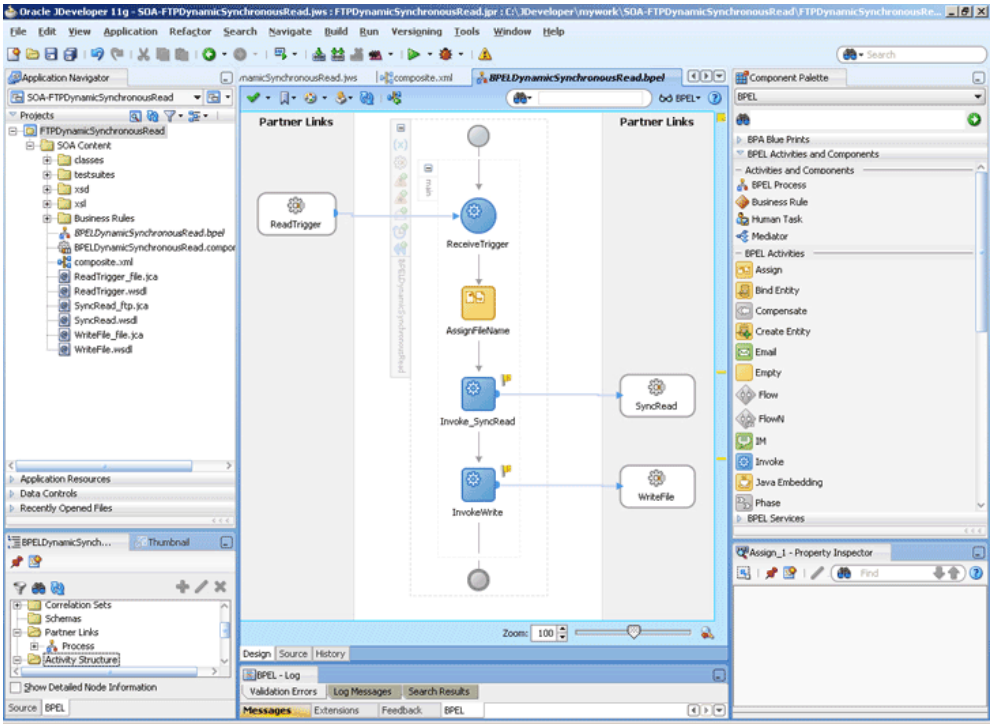
1. Drag and drop an **Assign** activity from the Component Palette in between the ReceiveTrigger and Invoke_SyncRead activities in the design area.
2. Double-click the **Assign** activity. The Assign dialog is displayed.
3. Enter `AssignFileName` in the **Name** field.
4. Click the **Copy Operation** tab. The Assign dialog is displayed.
5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
6. Create the copy operation between the trigger file name and the file variable, as shown in Figure 4–161.

Figure 4–161 The Create Copy Operation Dialog



7. Click **OK** in the Create Copy Operation dialog.
8. Click **OK** to return to the JDeveloper BPELDynamicSynchronousRead.bpel page, as shown in Figure 4–162.

Figure 4–162 The JDeveloper - BPELDynamicSynchronousRead.bpel

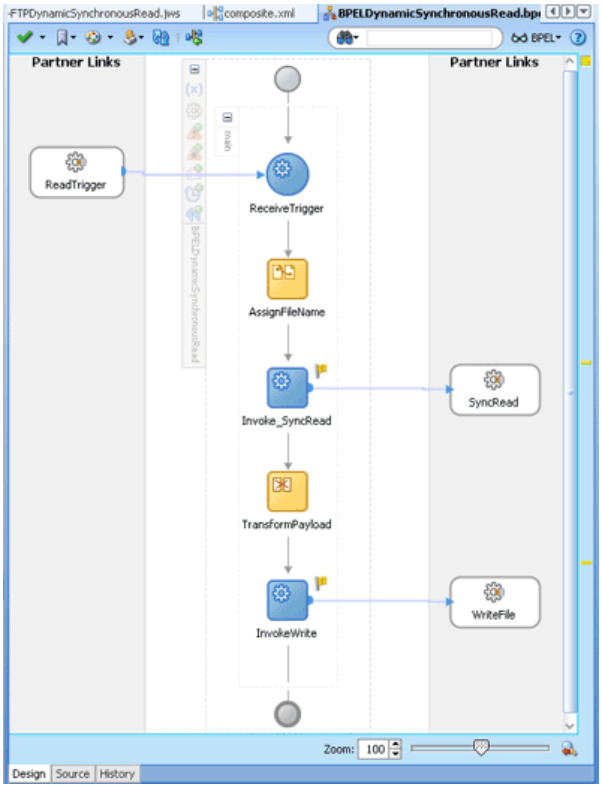


9. Click **File**, **Save All**.

Add a Transform Activity

1. Drag and drop a **Transform** activity from the Component Palette in between the Invoke_SyncRead and InvokeWrite activities in the design area.
2. Double-click the **Transform** activity. The Transform dialog is displayed.
3. Enter `TransformPayload` in the **Name** field.
4. Click the **Transformation** tab. The Transform dialog is displayed.
5. Click the **Create...** icon. The Source Variable dialog is displayed.
6. Select **InvokeSyncRead_SyncRead_OutputVariable** in the Source Variable box, and select **body** in the Source Part box, and then click **OK**. The Transform dialog is displayed with the Source and Part selected.
7. Select **InvokeWrite_Write_InputVariable** in the Target Variable list, select **body** in the Target Part.
8. Click the **Browse Mappings** icon at the end of the Mapper File field and select `addr1Toaddr2.xsl` file from the xsl directory in your project.
9. Click **OK**.
10. Click **File**, **Save All**. The BPELDynamicSynchronousRead.bpel page is displayed, as shown in Figure 4–163.

Figure 4–163 The JDeveloper - BPELDynamicSynchronousRead.bpel



4.5.10.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application. For more information, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

4.5.10.7 Monitoring Using Fusion Middleware Control Console

You can monitor the deployed SOA composite using Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Copy the `address-csv.txt` file to the input directory (see Section 4.5.10.1, "Prerequisites" for the location of this file) and ensure it gets processed. Check the output directory to ensure that an output file has been created.
3. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the Recent Instances area.
4. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
5. Click the Instance ID that you noted in Step 3. The Flow Trace page is displayed.
6. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
7. Expand a payload node to view payload details.
8. Click the **Flow** tab to view the process flow.
9. Click **ReceiveTrigger** to display the activity details.

4.5.11 Copying, Moving, and Deleting Files

The Oracle File and FTP Adapters let you copy or move a file from one location to another, or delete a file from the target directory. Additionally, the Oracle FTP Adapter lets you move or copy files from a local file system to a remote file system and from a remote file system to a local file system. This feature is implemented as a interaction specification for outbound services. So, this feature can be accessed either by using a BPEL invoke activity or a Mediator routing rule.

At a high level, you must create an outbound service and configure this service with the source and target directories and file names.

The following use cases demonstrate the new functionality supported by Oracle File and FTP Adapters that allow you to copy, move, and delete files by using an outbound service:

- Section 4.5.11.1, "Moving a File from a Local Directory on the File System to Another Local Directory"
- Section 4.5.11.2, "Copying a File from a Local Directory on the File System to Another Local Directory"
- Section 4.5.11.3, "Deleting a File from a Local File System Directory"
- Section 4.5.11.4, "Using a Large CSV Source File"
- Section 4.5.11.5, "Moving a File from One Remote Directory to Another Remote Directory on the Same FTP Server"

- Section 4.5.11.6, "Moving a File from a Local Directory on the File System to a Remote Directory on the FTP Server"
- Section 4.5.11.7, "Moving a File from a Remote Directory on the FTP Server to a Local Directory on the File System"
- Section 4.5.11.8, "Moving a File from One FTP Server to another FTP Server"

4.5.11.1 Moving a File from a Local Directory on the File System to Another Local Directory

You can model only a part of this procedure by using the wizard because the corresponding Adapter Configuration Wizard is not available. You must complete the remaining procedure by manually configuring the generated JCA file.

You must perform the following steps to move a file from a local directory on the file system to another local directory:

1. Create an empty BPEL process.
2. Drag and drop File Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
3. Click **Next**. The Service Name page is displayed.
4. Enter a service name in the Service Name field.
5. Click **Next**. The Adapter Interface page is displayed.
6. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
7. Select **Synchronous Read File**, enter `FileMove` in the **Operation Name** field, and then click **Next**. The File Directories page is displayed.

   Note: You have selected Synchronous Read File as the operation because the `WSDL` file that is generated because this operation is similar to the one required for the file I/O operation.

8. Enter a dummy physical path for the directory for incoming files, and then click **Next**. The File name page is displayed.

   Note: The dummy directory is not used. You must manually change the directory in a later step.

9. Enter a dummy file name, and then click **Next**. The Messages page is displayed.

   Note: The dummy file name you enter is not used. You must manually change the file name in a later step.

10. Select **Native format translation is not required (Schema is opaque)**, and then click **Next**. The Finish page is displayed.
11. Click **Finish**. The outbound Oracle File Adapter is now configured.

12. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in `FileMove` in the External References area. The BPEL component is connected to the Oracle File Adapter outbound service.
13. Create an invoke activity for the `FileMove` service that you just created by selecting the default settings.

    The next step is to modify the generated `WSDL` file for `MoveFileService` service and configure it with the new interaction specification for the move operation.
14. Open the `FileMove_file.jca` file and modify the endpoint interaction, as shown in the following example.

    You must configure the JCA file with the source and target directory and file details. You can either hardcode the source and target directory and file details in the JCA file or use header variables to populate them. In this example, header variables are used.

```
<adapter-config name="FileMove" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter" adapterRef=""/>
  <endpoint-interaction portType="FileMove_ptt" operation="FileMove">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.FileIoInteractionSpec">
      <property name="SourcePhysicalDirectory" value="foo1"/>
      <property name="SourceFileName" value="bar1"/>
      <property name="TargetPhysicalDirectory" value="foo2"/>
      <property name="TargetFileName" value="bar2"/>
      <property name="Type" value="MOVE"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

    Note: You have modified the `className` attribute, and added `SourcePhysicalDirectory`, `SourceFileName`,`TargetPhysicalDirectory`, `TargetFileName` and `Type`. Currently, the values for the source and target details are dummy. You must populate them at run-time. You can also hardcode them to specific directories or file names.

    The `Type` attributes decides the type of operation. Apart from `MOVE`, the other acceptable values for the `Type` attribute are `COPY` and `DELETE`.
15. Map the actual directory and file names to the source and target file parameters by performing the following procedure:
    a. Create 4 string variables with appropriate names. You must populate these variables with the source and target directory details. The BPEL source view shows you this:

```
<variables>
    <variable name="InvokeMoveOperation_FileMove_InputVariable"
messageType="ns1:Empty_msg"/>
    <variable name="InvokeMoveOperation_FileMove_OutputVariable"
messageType="ns1:FileMove_msg"/>
    <variable name="sourceDirectory" type="xsd:string"/>
    <variable name="sourceFileName" type="xsd:string"/>
    <variable name="targetDirectory" type="xsd:string"/>
```

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

b. Create an assign activity to assign values to `sourceDirectory`, `sourceFileName`, `targetDirectory`, and `targetFileName` variables. The assign operation appears in the BPEL source view as in the following example:

```
<assign name="AssignFileDetails">
      <copy>
        <from expression="'/home/alex'"/>
        <to variable="sourceDirectory"/>
      </copy>
      <copy>
        <from expression="'input.txt'"/>
        <to variable="sourceFileName"/>
      </copy>
      <copy>
        <from expression="'/home/alex'"/>
        <to variable="targetDirectory"/>
      </copy>
      <copy>
        <from expression="'output.txt'"/>
        <to variable="targetFileName"/>
      </copy>
    </assign>
```

In the preceding example, `input.txt` is moved from `/home/alex` to `output.txt` in `/home/alex`.

Note: The source and target details are hardcoded in the preceding example. You can also provide these details as run-time parameters.

---

c. Pass these parameters as headers to the invoke operation. The values in these variables override the parameters in the JCA file.

```
<invoke name="InvokeMoveOperation"
            inputVariable="InvokeMoveOperation_FileMove_InputVariable"
            outputVariable="InvokeMoveOperation_FileMove_OutputVariable"
            partnerLink="FileMove" portType="ns1:FileMove_ptt"
            operation="FileMove">
      <bpelx:inputProperty name="jca.file.SourceDirectory"
variable="sourceDirectory"/>
      <bpelx:inputProperty name="jca.file.SourceFileName"
variable="sourceFileName"/>
      <bpelx:inputProperty name="jca.file.TargetDirectory"
variable="targetDirectory"/>
      <bpelx:inputProperty name="jca.file.TargetFileName"
variable="targetFileName"/>
    </invoke>
```

16. Finally, add an initial receive or pick activity.

You have completed moving a file from a local directory on the file system to another local directory.

4.5.11.2 Copying a File from a Local Directory on the File System to Another Local Directory

Perform the following procedure to copy a file from a local directory on the file system to another local directory:

1. Follow steps 1 through 12 of Section 4.5.11.1, "Moving a File from a Local Directory on the File System to Another Local Directory".
2. Change the value of the `TYPE` attribute to `COPY` instead of `MOVE` in the endpoint interaction, in Step 14 of Section 4.5.11.1, "Moving a File from a Local Directory on the File System to Another Local Directory" as shown in the following example:

```
<adapter-config ...>
  <connection-factory .../>
  <endpoint-interaction ...>
    <interaction-spec
className="oracle.tip.adapter.file.outbound.FileIoInteractionSpec">
      <property .../>
      <property name="Type" value="COPY"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

4.5.11.3 Deleting a File from a Local File System Directory

To delete a file, you require `TargetPhysicalDirectory` and `TargetFileName` parameters.

> **Note:** You do not require `SourcePhysicalDirectory` and `SourceFileName` to delete a file from a local file system directory.

To delete a file, `delete_me.txt`, from `/home/alex` directory, you must perform the following:

1. Follow steps 1 through 12 in Section 4.5.11.1, "Moving a File from a Local Directory on the File System to Another Local Directory"
2. Change the value of the `TYPE` attribute to `DELETE` in the endpoint interaction in Step 14 of Section 4.5.11.1, "Moving a File from a Local Directory on the File System to Another Local Directory", as shown in the following example:

```
<adapter-config name="FileDelete" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter" adapterRef=""/>
  <endpoint-interaction portType="FileDelete_ptt" operation="FileDelete">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.FileIoInteractionSpec">
      <property name="TargetPhysicalDirectory" value="/home/alex"/>
      <property name="TargetFileName" value="delete_me.txt"/>
      <property name="Type" value="DELETE"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

4.5.11.4 Using a Large CSV Source File

Consider the following scenario, where you have a large CSV file of size 1 gigabyte coming on the source directory, and you must perform the following:

1. Translate the CSV into XML.
2. Transform the resulting XML using XSL.
3. Translate the result from the transform operation into a fixed length file.

This use case is similar to the `FlatStructure` sample in the BPEL samples directory. The difference is that the three steps occur in a single File I/O interaction.

> **Note:** All the three steps occur in a single File I/O interaction. This works only if all the records in the data file are of the same type.

To use a large CSV file and perform the operations listed in the preceding scenario, you must perform the following steps:

1. Copy the `address-csv.xsd` and `address-fixedLength.xsd` files from the FlatStructure sample into the `xsd` directory of your project.
2. Copy `addr1Toaddr2.xsl` from the FlatStructure sample into the `xsl` directory of your project.
3. Configure the File I/O interaction, as shown in the following example. At a high level, you must specify the source schema, the target schema, and the XSL in the interaction specification along with the source and target directory or file details, as shown in the following example:

```
<adapter-config name="FileMove" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter" adapterRef=""/>
  <endpoint-interaction portType="FileMove_ptt" operation="FileMove">
    <interaction-spec
className="oracle.tip.adapter.file.outbound.FileIoInteractionSpec">
      <property name="SourcePhysicalDirectory" value="foo1"/>
      <property name="SourceFileName" value="bar1"/>
      <property name="SourceSchema" value="xsd/address-csv.xsd"/>
      <property name="SourceSchemaRoot value="Root-Element"/>
      <property name="SourceType" value="native"/>
      <property name="TargetPhysicalDirectory" value="foo2"/>
      <property name="TargetFileName" value="bar2"/>
      <property name="TargetSchema" value="xsd/address-fixedLength.xsd"/>
      <property name="TargetSchemaRoot value="Root-Element"/>
      <property name="TargetType" value="native"/>
      <property name="Xsl value="xsl/addr1Toaddr2.xsl"/>
      <property name="Type" value="MOVE"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

You have provided the following additional parameters:

- `SourceSchema`: Relative path to the source schema.
- `SourceSchemaRoot`: The root element in the source schema.
- `SourceType`: The type of data. The other possible type is XML.
- `TargetSchema`: Relative path to the target schema.

- `TargetSchemaRoot`: The root element in the target schema.
- `TargetType`: The type of data. The other possible type is XML.
- `Xsl`: Relative path to the Xsl file.

4.5.11.5 Moving a File from One Remote Directory to Another Remote Directory on the Same FTP Server

The I/O use cases for the Oracle FTP Adapter are very similar to those for Oracle File Adapter. However, there are a few nuances that need attention.

In this use case you move a file within the same directory, which is similar to a rename operation on the same server. Most FTP servers support the `RNFR/RNTO` FTP commands that let you rename a file on the FTP server.

However, even if the `RNFR/RNTO` commands are not supported, moving a file within the same directory is still possible because of a binding property, `UseNativeRenameOperation`. By default, this property is set to `TRUE`, and in this case the Oracle FTP Adapter uses the native `RNFR/RNTO` commands. However, if this property is set to `FALSE`, then the Oracle FTP Adapter uses the `Get` and `Put` commands followed by a `Delete` command to emulate a move operation.

You can model only a part of this procedure by using the wizard because the corresponding Adapter Configuration Wizard is not available. You must complete the remaining procedure by manually configuring the generated JCA file.

You must perform the following steps to move a file from a remote directory to another remote directory on the same FTP server:

1. Create an empty BPEL process.
2. Drag and drop FTP Adapter from the Component Palette to the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
3. Click **Next**. The Service Name page is displayed.
4. Enter a service name in the Service Name field.
5. Click **Next**. The Adapter Interface page is displayed.
6. Click **Next**. The FTP Server Connection page is displayed.
7. Enter the JNDI name for the FTP server, and click **Next**. The Operation page is displayed.
8. Select **Synchronous Get File**, enter `FTPMove` in the **Operation Name** field, and then click **Next**. The File Directories page is displayed.

   Note: You have selected Synchronous Get File as the operation because the `WSDL` file that is generated because this operation is similar to the one required for the file I/O operation.

9. Enter a dummy physical path for the directory for incoming files, and then click **Next**. The File name page is displayed.

   Note: The dummy directory is not used. You must manually change the directory in a later step.

10. Enter a dummy file name, and then click **Next**. The File Name page is displayed.

    Note: The dummy file name you enter is not used. You must manually change the file name in a later step.

11. Click **Next**. The Messages page is displayed.
12. Select **Native format translation is not required (Schema is opaque)**, and then click **Next**. The Finish page is displayed.
13. Click **Finish**. The outbound Oracle File Adapter is now configured.
14. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in `FTPMove` in the External References area. The BPEL component is connected to the Oracle FTP Adapter outbound service.
15. Click **File**, **Save All**.
16. Create an invoke activity for the `FTPMove` service that you just created.

    The next step is to modify the generated `WSDL` file for `FTPMove` service and configure it with the new interaction specification for the move operation.

17. Open the `FTPMove_ftp.jca` file and modify the `interaction-spec`, as shown in the following example.

    You must configure the JCA file with the source and target directory and file details. You can either hardcode the source and target directory and file details in the JCA file or use header variables to populate them. In this example, header variables are used.

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

  <connection-factory location="eis/Ftp/FtpAdapter" adapterRef=""/>
  <endpoint-interaction portType="FTPMove_ptt" operation="FTPMove">
    <interaction-spec
className="oracle.tip.adapter.ftp.outbound.FTPIoInteractionSpec">
      <property name="SourcePhysicalDirectory" value="foo1"/>
      <property name="SourceFileName" value="bar1"/>
      <property name="TargetPhysicalDirectory" value="foo2"/>
      <property name="TargetFileName" value="bar2"/>
      <property name="Type" value="MOVE"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

    Note: You have modified the `className` attribute, and added `SourcePhysicalDirectory`, `SourceFileName`, `TargetPhysicalDirectory`, `TargetFileName`, and `Type`. Currently, the values for the source and target details are dummy. You must populate them at run-time. You can also hardcode them to specific directories or file names.

    The `Type` attributes decides the type of operation. Apart from `MOVE`, the other acceptable values for the `Type` attribute are `COPY` and `DELETE`.

18. Map the actual directory and file names to the source and target file parameters by performing the following procedure:

    a. Create 4 string variables with appropriate names. You must populate these variables with the source and target directory details. The BPEL source view shows you this:

```
  <variables>
    <variable name="InvokeMoveOperation_FileMove_InputVariable"
messageType="ns1:Empty_msg"/>
    <variable name="InvokeMoveOperation_FileMove_OutputVariable"
messageType="ns1:FileMove_msg"/>
    <variable name="sourceDirectory" type="xsd:string"/>
    <variable name="sourceFileName" type="xsd:string"/>
    <variable name="targetDirectory" type="xsd:string"/>
    <variable name="targetFileName" type="xsd:string"/>
  </variables>
```

    b. Create an assign activity to assign values to `sourceDirectory`, `sourceFileName`, `targetDirectory`, and `targetFileName` variables. The assign operation appears in the BPEL source view as in the following example:

```
    <assign name="AssignFTPFileDetails">
      <copy>
        <from expression="'/home/ftp'"/>
        <to variable="sourceDirectory"/>
      </copy>
      <copy>
        <from expression="'input.txt'"/>
        <to variable="sourceFileName"/>
      </copy>
      <copy>
        <from expression="'/home/ftp/out'"/>
        <to variable="targetDirectory"/>
      </copy>
      <copy>
        <from expression="'output.txt'"/>
        <to variable="targetFileName"/>
      </copy>
    </assign>
```

    In the preceding example, `input.txt` is moved or renamed from `/home/ftp` to `output.txt` in `/home/ftp/out`.

    Note: The source and target details are hardcoded in the preceding example. You can also provide these details as run-time parameters.

    c. Pass these parameters as headers to the invoke operation. The values in these variables override the parameters in the JCA file.

```
<invoke name="InvokeRenameService"
inputVariable="InvokeRenameService_RenameFile_InputVariable"
partnerLink="RenameFTPFile" portType="ns2:RenameFile_ptt"
operation="RenameFile">
<bpelx:inputProperty name="jca.file.SourceDirectory"
variable="returnDirectory"/>
<bpelx:inputProperty name="jca.file.SourceFileName"
variable="returnFile"/>
```

```
<bpelx:inputProperty name="jca.file.TargetDirectory"
variable="returnDirectory"/>
<bpelx:inputProperty name="jca.file.TargetFileName"
variable="targetFile"/>
</invoke>
```

19. Finally, add an initial receive or pick activity.

    You have completed moving or renaming a file from a remote directory to another remote directory on the same FTP server.

Note: If the FTP server does not support the `RNFR/RNTO` FTP commands, then you must set `UseNativeRenameOperation` to `FALSE` and define the property in `composite.xml`, as shown in the following example:

```
<reference name="FTPMove" ui:wsdlLocation="FTPMove.wsdl">
  <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/ftp/SOAFtpIO/SOAF
tpIO/FTPMove/#wsdl.interface(FTPMove_ptt)"/>
    <binding.jca config="FTPMove_ftp.jca">
      <property name="UseNativeRenameOperation" type="xs:string"
many="false" override="may">false</property>
    </binding.jca>
</reference>
```

4.5.11.6 Moving a File from a Local Directory on the File System to a Remote Directory on the FTP Server

The steps for this use case are the same as the steps for the use case in Section 4.5.11.5, "Moving a File from One Remote Directory to Another Remote Directory on the Same FTP Server" except that you must configure the source directory as local and the target directory as remote.

Use the `SourceIsRemote` and `TargetIsRemote` properties to specify whether the source and target file are on the local or remote file system, as shown in the following example:

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

  <connection-factory location="eis/Ftp/FtpAdapter" adapterRef=""/>
  <endpoint-interaction portType="FTPMove_ptt" operation="FTPMove">
    <interaction-spec
className="oracle.tip.adapter.ftp.outbound.FTPIoInteractionSpec">
      <property name="SourcePhysicalDirectory" value="foo1"/>
      <property name="SourceFileName" value="bar1"/>
      <property name="SourceIsRemote" value="false"/>
      <property name="TargetPhysicalDirectory" value="foo2"/>
      <property name="TargetFileName" value="bar2"/>
      <property name="Type" value="MOVE"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

Note: In this example, you have configured `SourceIsRemote` as `false`. In this case, the FTP input and output operation assumes that the source file comes from a local file system. Also, notice that you did not specify the parameter for target because `TargetIsRemote` is set to `true` by default.

4.5.11.7 Moving a File from a Remote Directory on the FTP Server to a Local Directory on the File System

The steps for this use case are the same as the steps for the use case in Section 4.5.11.6, "Moving a File from a Local Directory on the File System to a Remote Directory on the FTP Server" except that you must configure the source directory as remote and the target directory as local, as shown in the following example:

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

  <connection-factory location="eis/Ftp/FtpAdapter" adapterRef=""/>
  <endpoint-interaction portType="FTPMove_ptt" operation="FTPMove">
    <interaction-spec
className="oracle.tip.adapter.ftp.outbound.FTPIoInteractionSpec">
      <property name="SourcePhysicalDirectory" value="foo1"/>
      <property name="SourceFileName" value="bar1"/>
      <property name="TargetPhysicalDirectory" value="foo2"/>
      <property name="TargetFileName" value="bar2"/>
      <property name="TargetIsRemote" value="false"/>
      <property name="Type" value="MOVE"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

Note: In this example, you have configured `TargetIsRemote` as `false`. In this case, the FTP I/O assumes that the source file comes from a remote file system whereas the target is on a local file system. Also, notice that you did not specify the parameter for source because `SourceIsRemote` is set to true by default.

4.5.11.8 Moving a File from One FTP Server to another FTP Server

To move a file from one FTP server to another FTP server you must sequentially perform the use cases documented in the following sections:

1. Section 4.5.11.7, "Moving a File from a Remote Directory on the FTP Server to a Local Directory on the File System" to upload the file from the local directory to another FTP server
2. Section 4.5.11.6, "Moving a File from a Local Directory on the File System to a Remote Directory on the FTP Server" to download the file from the FTP server to a local directory

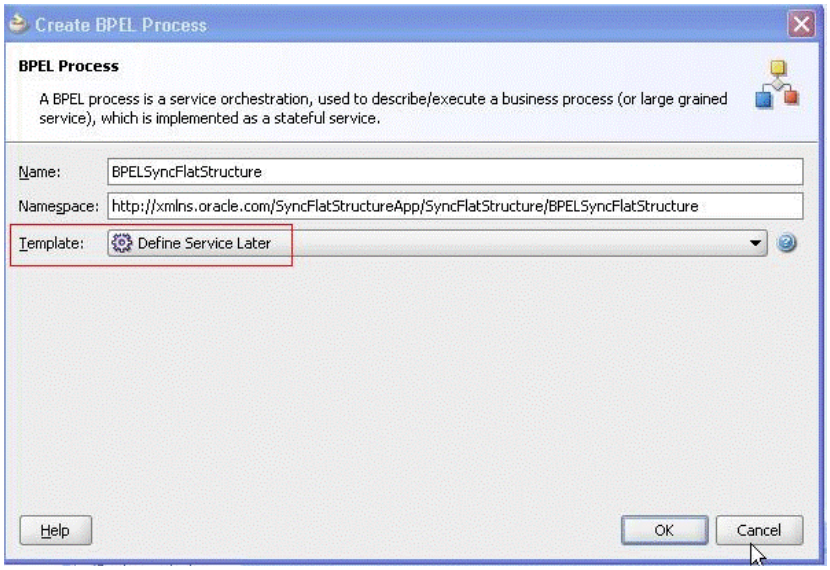
4.5.12 Creating a Synchronous BPEL Composite using File Adapter

By default, the JDeveloper Adapter Wizard generates asynchronous WSDLs when you use technology adapters. Typically, you follow these steps when creating an adapter scenario in 11g:

1. Create a SOA Application with either Composite with BPEL or an Empty Composite selected. If you choose Empty Composite, you must drop the BPEL Process from the Service Components pane onto the SOA Composite Editor.

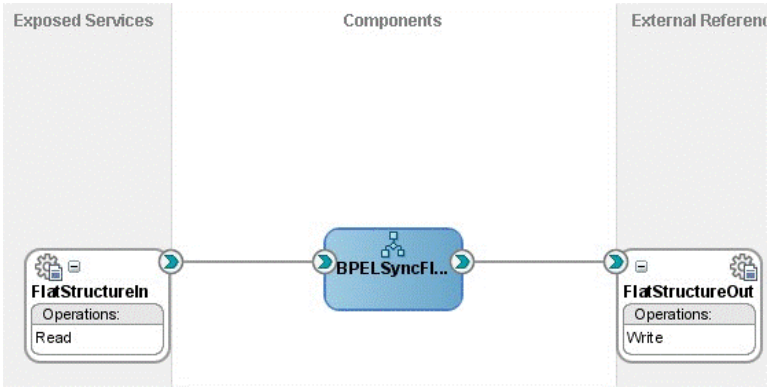
   You arrive at the screen below where you fill in the process details. You are required to choose Define Service Later as the template.

Figure 4–164 Create Process Screen



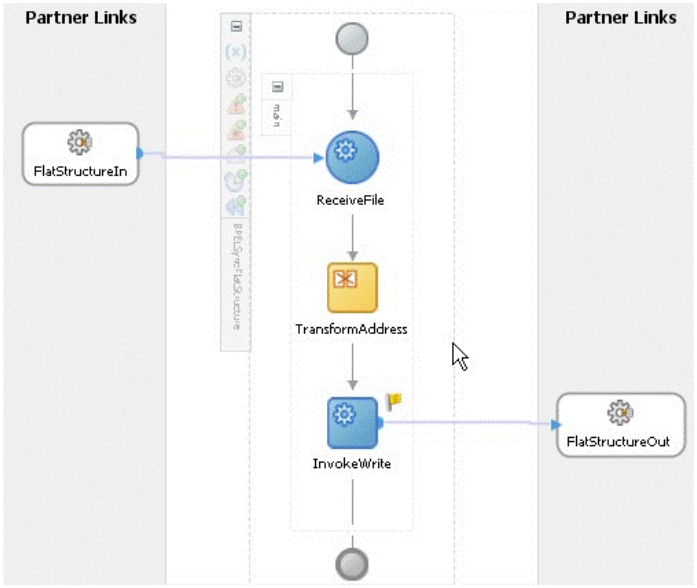
2. Next, you create the inbound service and outbound references and wire them with the BPEL component.

Figure 4–165 Cresting the Inbound Serv ice and Outbound References



3. And, finally you create the BPEL process with the initiating <receive> activity to retrieve the payload and an **<invoke>** activity to write the payload

Figure 4–166



This is how most BPEL processes that use Adapters are modeled. The generated WSDL implies one-way directionally one way and that makes the BPEL process asynchronous:

Figure 4–167 Generated One-Way WSDL

```
<?binding.jca FlatStructureIn_file.jca?>
<wsdl:definitions name="FlatStructureIn"
                  targetNamespace="http://xmlns.oracle.com/...." ...>
    <plt:partnerLinkType name="Read_plt">
        <plt:role name="Read_role">
            <plt:portType name="tns:Read_ptt"/>
        </plt:role>
    </plt:partnerLinkType>
    <wsdl:types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://xmlns.oracle.com/pcbpel/demoSchema/csv"
              schemaLocation="xsd/address-csv.xsd"/>
        </schema>
    </wsdl:types>
    <wsdl:message name="Read_msg">
        <wsdl:part name="body" element="imp1:Root-Element"/>
    </wsdl:message>
    <wsdl:portType name="Read_ptt">
        <wsdl:operation name="Read">
            <wsdl:input message="tns:Read_msg"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

One way WSDL

In other words, the inbound File Adapter polls for files in the directory and for each file that it finds there, it translates the content into XML and publishes to BPEL.

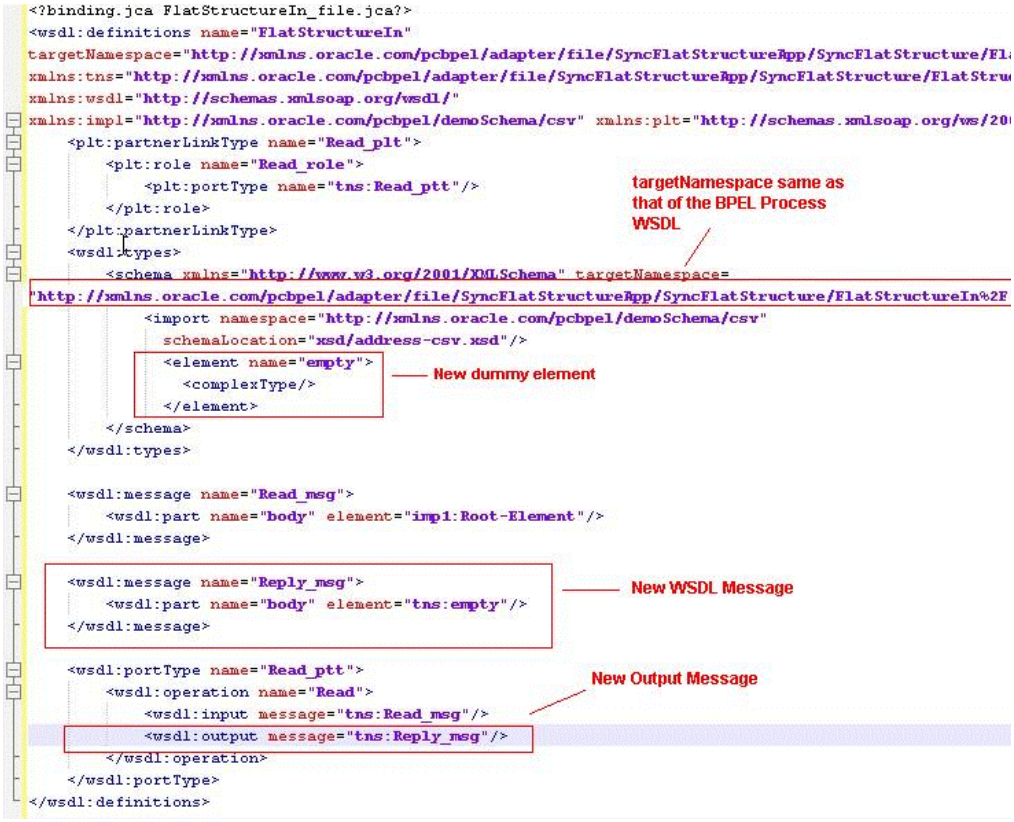
However, because the BPEL process is asynchronous, the File Adapter returns immediately after the publish and performs the required post processing-for example. deletion/archival of data.

The disadvantage with such asynchronous BPEL processes is that it becomes difficult to throttle the inbound adapter. In otherwords, the inbound adapter would keep sending messages to BPEL *without waiting for the downstream business processes to complete*. This can lead to issues such as higher memory usage and CPU usage.

To mitigate the occurrence of these problems, you can manually change the WSDL and BPEL artifacts into synchronous processes. Once you have changed the synchronous to synchronous BPEL processes, the inbound File Adapter automatically throttles itself because the File Adapter is forced to wait for the downstream process to complete with a <reply> before processing the next file or message.

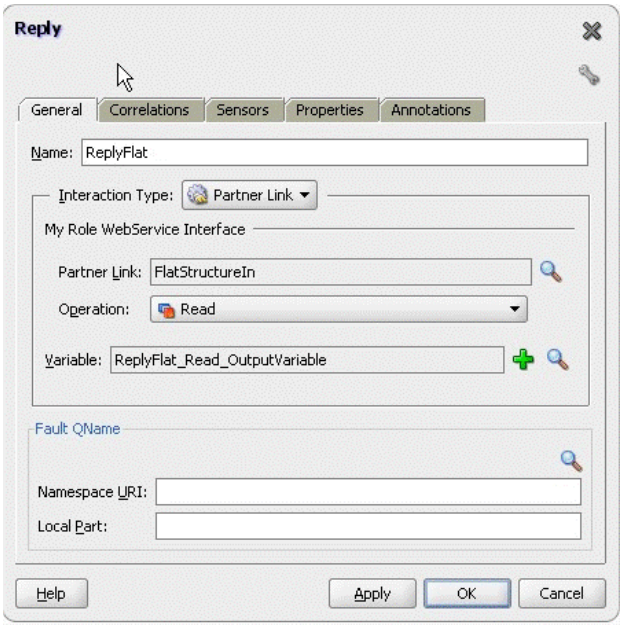
Refer to the altered WSDL below. Here, you convert the one-way WSDL to a two-way WSDL-- thereby making the WSDL synchronous.

Figure 4–168 Asynchronous WSDL Altered to be Two-Way WSDL



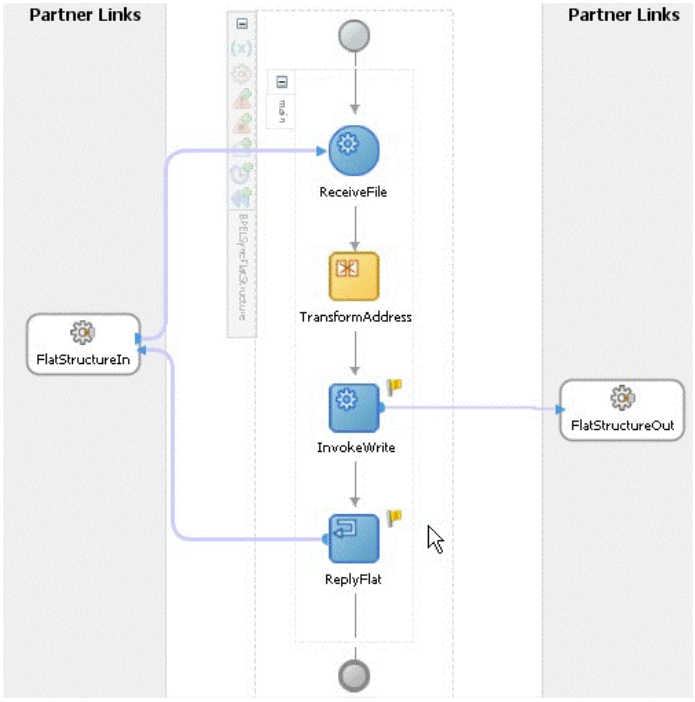
The next step is to add a <reply> activity to the inbound adapter partnerlink at the end of your BPEL process, for example:

Figure 4–169 Specifying a Reply to the Inbound Adapter



Finally, the process looks like this:

Figure 4–170 The Synchronous File Adapter Process with Receive and Reply BPEL Activities



This type of exercise is not required for the Mediator because the Mediator routing rules are sequential by default. In other words, the Mediator uses the caller thread (inbound file adapter thread) for processing routing rules. This is the case even if the WSDL for mediator is one-way..

4.5.12.1 Changing the Connection Factory JNDI Dynamically in Ftp Adapter

Where there is a requirement to send the same file to five different FTP servers, you could create, for example, five FtpAdapter references, one for each connection-factory location. However, this is not the most optimal approach; instead, you can use the concept of "Dynamic Partner Links".

If you're running the adapter in managed mode, it requires you to configure the connection factory JNDI in the WebLogic Server console for the FtpAdapter.

In the sample below, the connection-factory JNDI location "eis/Ftp/FtpAdapter" has been mapped with the Ftp server running on localhost.

```
<?xml version="1.0" encoding="UTF-8"?>
<wls:weblogic-connector ...>
   <wls:outbound-resource-adapter>
      <wls:default-connection-properties>
         <wls:pool-params../>
         <[..]>
      </wls:default-connection-properties>
      <wls:connection-definition-group.../>
         <[..]>
         <wls:connection-instance>
            <wls:description>Ftp Adapter</wls:description>
            <wls:jndi-name>eis/Ftp/FtpAdapter</wls:jndi-name>
            <wls:connection-properties>
               <wls:properties>
                  <wls:property>
                     <wls:name>host</wls:name>
                     <wls:value>localhost</wls:value>
                  </wls:property>
                  <wls:property>
                     <wls:name>port</wls:name>
                     <wls:value>21</wls:value>
                  </wls:property>
                  <wls:property>
                     <wls:name>user</wls:name>
                     <wls:value>blah</wls:value>
                  </wls:property>
                  <wls:property>
                     <wls:name>password</wls:name>
                     <wls:value>blah</wls:value>
                  </wls:property>
                  <[..]>
                  <[..]>
```

```
<adapter-config name="FTPOut" adapter="Ftp Adapter" wsdlLocation="FTPOut.wsdl"
        xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/Ftp/FtpAdapter" adapterRef=""/>
  <endpoint-interaction portType="Put_ptt" operation="Put">
    <interaction-spec className="oracle.tip.adapter.ftp.outbound.FTPInteractionSpec">
      <property name="FileType" value="ascii"/>
      <property name="PhysicalDirectory" value="/tmp/ftp/out"/>
      <property name="FileNamingConvention" value="file_%SEQ%.out"/>
      <property name="Append" value="false"/>
      <property name="NumberMessages" value="1"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

After you've configured the connection factory on your application server, you must refer to the connection-factory JNDI in the jca artifact of your SCA process. In the example below, the FTPOut reference in the following .jca file uses the FTP server corresponding to `eis/Ftp/FtpAdapter`

You can change this connection-factory location dynamically using JCA header properties in both BPEL and Mediator service engines. To do so, the business scenario

involving BPEL or Mediator is required to use a reserved JCA header property `jca.jndi` as shown in the following.

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<process name="BPELDynamicPL"...>
  -->
  <partnerLinks.../>
  <variables>
    <variable name="Receive_File_Get_InputVariable" messageType="ns1:Get_msg"/>
    <variable name="Invoke_FTPServer1_Put_InputVariable" messageType="ns2:Put_msg"/>
    <variable name="jndiLocation" type="xsd:string"/>
  </variables>
  <sequence name="main">
    <receive .../>
    <assign name="Assign_Payload".../>
    <assign name="Assign_FtpServer1_JNDI">
      <copy>
        <from expression="'eis/Ftp/FtpAdapter1'"/>
        <to variable="jndiLocation"/>
      </copy>
    </assign>
    <invoke name="Invoke_FTPServer1"
            inputVariable="Invoke_FTPServer1_Put_InputVariable"
            partnerLink="FTPOut" portType="ns2:Put_ptt" operation="Put">
      <bpelx:inputProperty name="jca.jndi" variable="jndiLocation"/>
    </invoke>

    <assign name="Assign_FtpServer2_JNDI">
      <copy>
        <from expression="'eis/Ftp/FtpAdapter2'"/>
        <to variable="jndiLocation"/>
      </copy>
    </assign>
    <invoke name="Invoke_FTPServer2"
            inputVariable="Invoke_FTPServer1_Put_InputVariable"
            partnerLink="FTPOut" portType="ns2:Put_ptt" operation="Put">
      <bpelx:inputProperty name="jca.jndi" variable="jndiLocation"/>
    </invoke>

  </sequence>
</process>
```

Similarly, for the Mediator, the mplan is:

You must remember the following when using dynamic partner links:

- You must preconfigure the connection factories on the SOA server. In the BPEL example above, both `eis/Ftp/FtpAdater1` and `eis/Ftp/FtpAdater2` must be configured in the WebLogic Server deployment descriptor for the FtpAdapter before your deployment of the scenario.
- Dynamic Partner Links are applicable to *outbound* invocations only

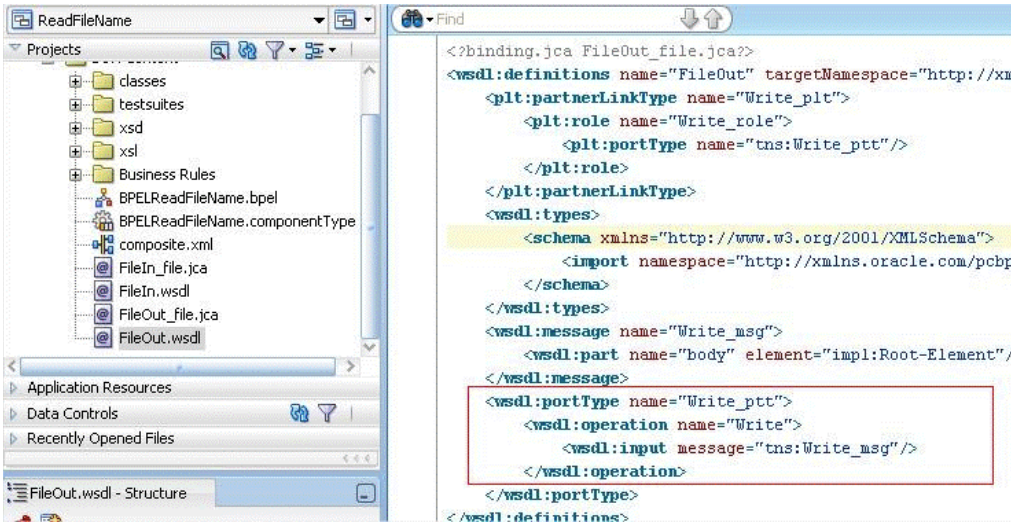
4.5.12.2 Retrieving the Details of the File from an Outbound Write Operation

You can capture the details of the file that was written out as a part of a BPEL process that is invoking a File/FTP Adapter

For example, using `FileNamingConvention` as `PurchaseOrder_%SEQ%.txt`, you might be required to perform post processing based on the file that was written out (the name of the file is not known until the adapter invocation completes for example, `PurchaseOrder_1.txt, PurchaseOrder_2.txt` ...)

To provide for the capture of metadata, you must edit the WSDL so the File/Ftp Adapter can return the metadata of the file being written. In general, the File/Ftp Write/Put WSDL operations are one way as shown below.

Figure 4–171 Editing the WSDL File So the File/FTP Adapter Returns File Metadata



In general, the File/FTP Write/Put WSDL operations, as inserted in the .jca file, are one way of enabling the return of the metadata, as shown below.

The File/TP Adapters are designed to return the metadata back if the WSDL is tweaked into a two-way WSDL. In addition, the `</wsdl.output>` must import the `fileread.xsd` schema. You must copy the `fileread.xsd` schema from the following location:

http://blogs.oracle.com/adapters/resource/2010-04-12/fileread.xsd

Figure 4–172 Fileread

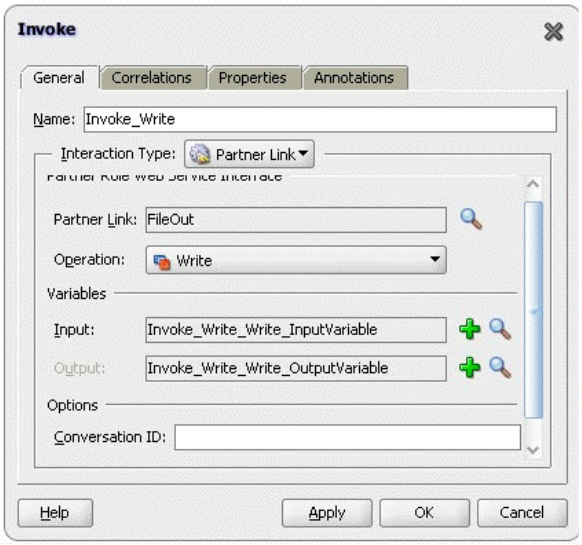
```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">
  <xsd:element name="OutboundFileHeaderType">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="fileName" type="xsd:string"/>
        <xsd:element name="directory" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The next step is to edit the WSDL to ensure it is changed into a two-way WSDL.

Figure 4–173 Editing the WSDL to Change It Into a Two-Way WSDL

Specify an Invoke WSDL statement through the BPEL Invoke statement panel.The file metadata would be returned as a part of the BPEL output variable:

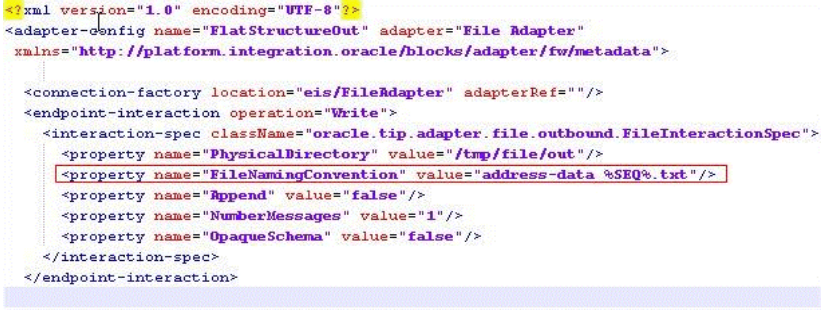
Figure 4–174 Specifying an Invoke WSDL Statement through the BPEL Invoke Statement Panel



4.5.13 Changing the Sequencing Strategy for FILE/Ftp Adapter

The File/Ftp Adapter enables you to configure outbound writes to use a sequence number. For example, if you choose `address-data_%SEQ%.txt` as the FileNamingConvention, all files would be generated as `address-data_1.txt`, `address-data_2.txt`,..

Figure 4–175 Configuring Outbound Writes



The sequence number comes from the control directory for the particular adapter project(or scenario). For each project that use the File or Ftp Adapter, a unique directory is created for book-keeping purposes. Because this control directory must be unique, the adapter uses a digest to ensure that no two control directories are the same.

For example, for the FlatStructure sample in the example above, the control information for my project would go under `FMW_HOME/user_projects/domains/soainfra/fileftp/controlFiles/[DIGEST]/outbound` where the value of `DIGEST` would differ from one project to another.

Within this directory, there is a file `control_ob.properties` file where the sequence number is maintained. The sequence number is maintained in binary form and you might need a hexadecimal editor to view its content. There is another zero byte file, `SEQ_nnn`. This extra file is maintained as a backup.

One of the challenges faced by the adapter run time is to guard all writes to the control files so no two threads inadverently attempt to update the control files at the same time. It does this guarding with the help of a "Mutex". The mutex is of different types:

- In-memory
- DB-based
- Coherence-based
- User-defined

There might be scenarios, particularly when the Adapter is under heavy tranactional load, where the mutex is a bottleneck. The Adapter, however, enables you to change the configuration so the adapter sequence value is derived from a database sequence or a stored procedure. In such a situation, the mutex is by-passed, and the process results in improved throughput.

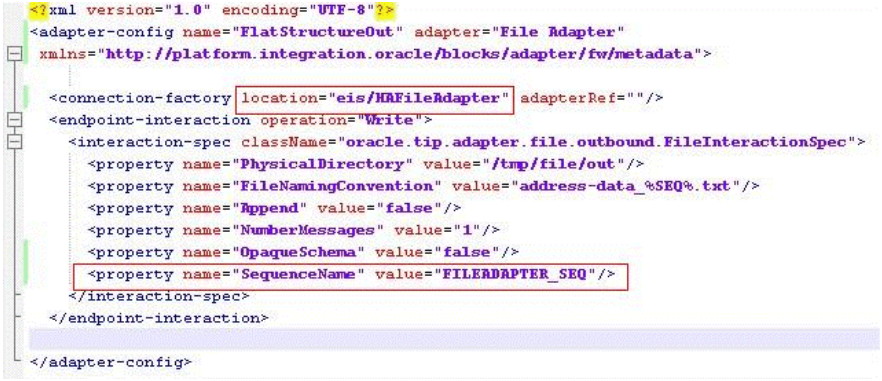
The simplest way to achieve improved throughput is by switching your JNDI connection factory location for the outbound JCA file to use the `eis/HAFileAdapter`:

Figure 4–176 Switching the JNDI Connection Factory to Use the HAFileAdapter

With this change, the Adapter run time creates a sequence on the Oracle database. For example, if you do a `select * from user_sequences` in your soa-infra schema, you see a new sequence being created with name as `SEQ_<GUID>__` (where the `GUID` differs by project).

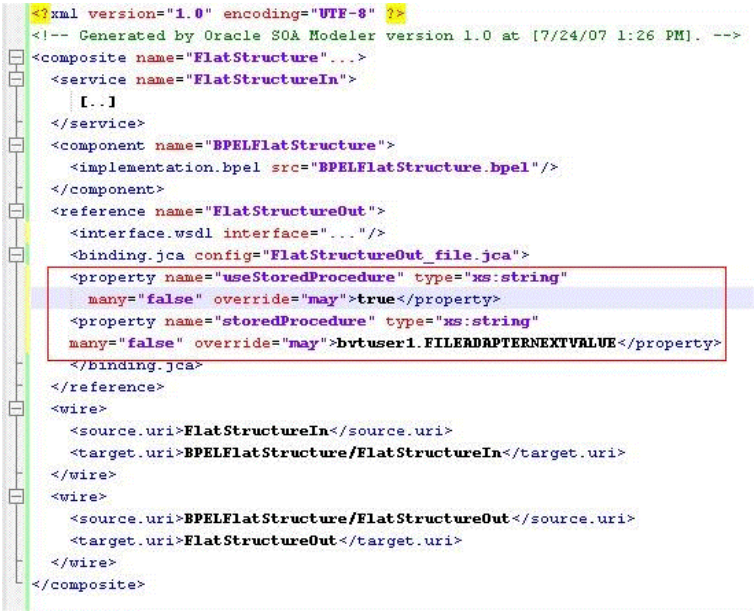
However, to use your *own* sequence, you must add a new property to your JCA file called `SequenceName`. You must create this sequence on your soainfra schema beforehand.

Figure 4–177 Adding the SequenceName Property



Using DB2 or MSSQL Server as the dehydration support is a bit different. DB2 supports sequences natively but MSSQL Server does not. The Adapter run time uses a natively generated sequence for DB2, but, for MSSQL server, the Adapter relies on a stored procedure that ships with the product.To achieve the same result for a SOA Suite running DB2 as the dehydration store, change the connection factory JNDI name in the JCA file to `eis/HAFileAdapterDB2`. For MSSQL, use `eis/HAFileAdapterMSSQL`. To use a stored procedure other than the one that ships with the product, you must rely on binding properties to override the adapter behavior; specifically, you must instruct the adapter to use a stored procedure:

Figure 4–178 Using a Stored Procedure



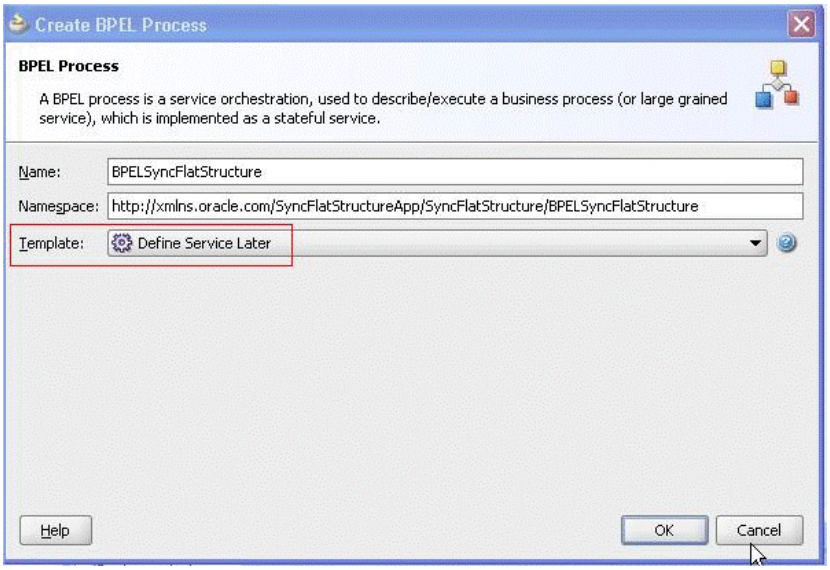
```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SOA Modeler version 1.0 at [7/24/07 1:26 PM]. -->
<composite name="FlatStructure"...>
  <service name="FlatStructureIn">
    [..]
  </service>
  <component name="BPELFlatStructure">
    <implementation.bpel src="BPELFlatStructure.bpel"/>
  </component>
  <reference name="FlatStructureOut">
    <interface.wsdl interface="..."/>
    <binding.jca config="FlatStructureOut_file.jca">
    <property name="useStoredProcedure" type="xs:string"
      many="false" override="may">true</property>
    <property name="storedProcedure" type="xs:string"
    many="false" override="may">bvtuser1.FILEADAPTERNEXTVALUE</property>
    </binding.jca>
  </reference>
  <wire>
    <source.uri>FlatStructureIn</source.uri>
    <target.uri>BPELFlatStructure/FlatStructureIn</target.uri>
  </wire>
  <wire>
    <source.uri>BPELFlatStructure/FlatStructureOut</source.uri>
    <target.uri>FlatStructureOut</target.uri>
  </wire>
</composite>
```

When the File/Ftp Adapter is used in Append mode, the adapter run time degrades the mutex to use pessimistic locks to prevent writers from different nodes appending to the same file at the same time.

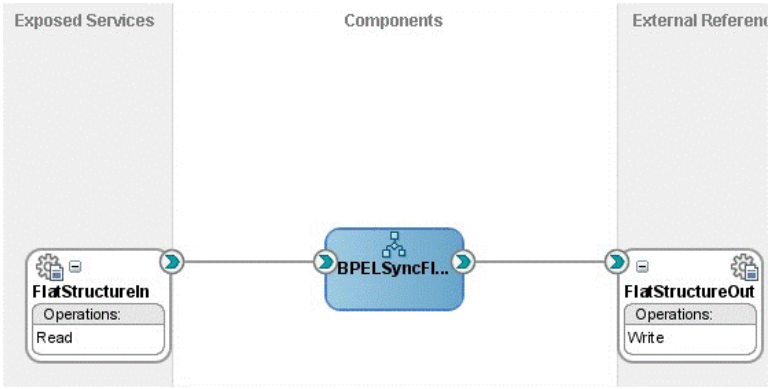
4.5.14 Creating a Synchronous BPEL Composite using the File Adapter

By default, the JDeveloper Adapter wizard generates asynchronous WSDLs when you use technology adapters. Typically, you follow the following steps when creating an adapter scenario in Release 11g:

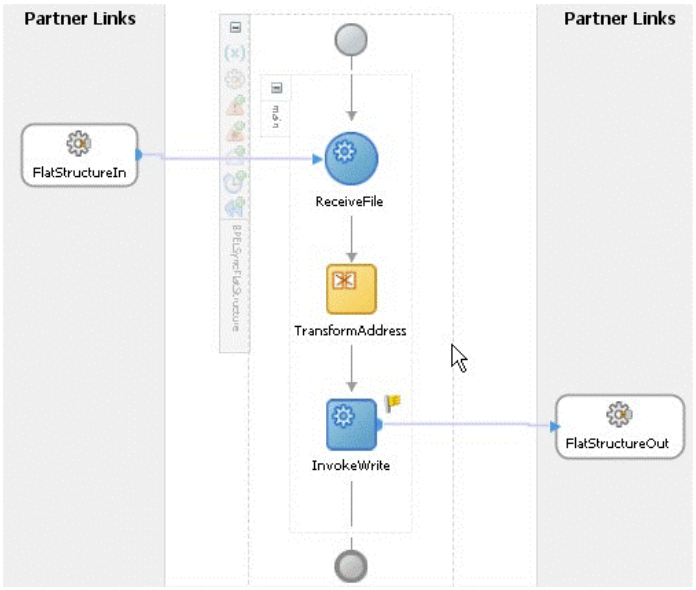
1. Create a SOA Application with either **Composite with BPEL** or an **Empty Composite**. If you choose **Empty Composite**, you must drop the BPEL Process from the **Service Components** panel onto the SOA Composite Editor. Either way, you arrive at the process details screen You must choose **Define Service Later** as the template.



2. Next, you create the inbound service and outbound reference and wire them with the BPEL component:



3. Finally you create the BPEL process with the initiating **<receive>** activity to retrieve the payload and an **<invoke>** activity to write the payload.



This is how most BPEL processes that use Adapters are modeled. The generated WSDL is one-way, which makes the BPEL process asynchronous.

```
<?binding.jca FlatStructureIn_file.jca?>
<wsdl:definitions name="FlatStructureIn"
                  targetNamespace="http://xmlns.oracle.com/...." ...>
    <plt:partnerLinkType name="Read_plt">
        <plt:role name="Read_role">
            <plt:portType name="tns:Read_ptt"/>
        </plt:role>
    </plt:partnerLinkType>
    <wsdl:types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://xmlns.oracle.com/pcbpel/demoSchema/csv"
              schemaLocation="xsd/address-csv.xsd"/>
        </schema>
    </wsdl:types>
    <wsdl:message name="Read_msg">
        <wsdl:part name="body" element="imp1:Root-Element"/>
    </wsdl:message>
    <wsdl:portType name="Read_ptt">
        <wsdl:operation name="Read">
            <wsdl:input message="tns:Read_msg"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

One way WSDL

The inbound File Adapter polls for files in the directory and for each file it finds there, the Adapter translates the content into XML and publishes to BPEL. But, because the BPEL process is asynchronous, the Adapter returns immediately after the publish operation and performs the required post processing, for example, deletion/archival.

The disadvantage with such asynchronous BPEL processes is that it becomes difficult to throttle the inbound adapter. In other words, the inbound adapter keeps sending messages to BPEL without waiting for the downstream business processes to complete. This can lead to several issues including higher memory usage, and CPU usage.

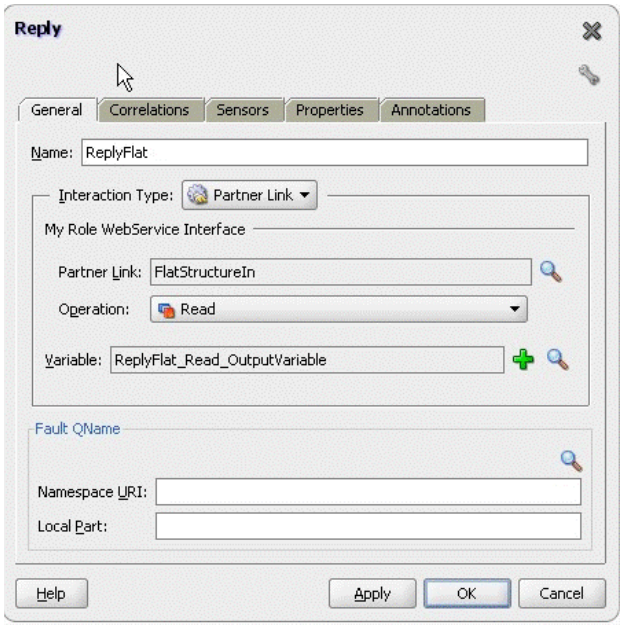
To alleviate these problems, manually tweak the WSDL and BPEL artifacts into synchronous processes. Oncet here are synchronous BPEL processes, the inbound adapter automatically throttles itself as the adapter is forced to wait for the

downstream process to complete with a <reply> before processing the next file or message.

In the following WSDL, the one-way WSDL has been converted to a two-way WSDL, thereby making the WSDL synchronous:

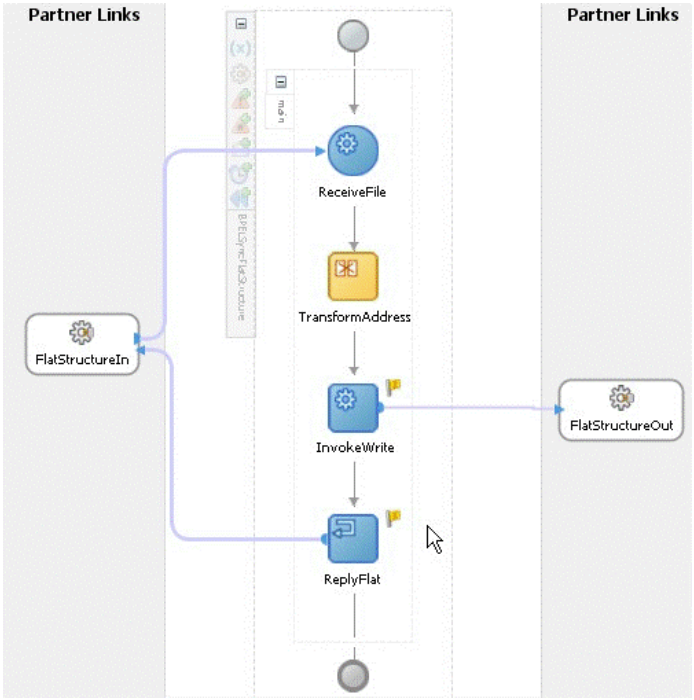
Figure 4–179 One-Way WSDL Converted Into a Two-Way WSDL

Add a **<reply>** activity to the inbound adapter partnerlink at the end of your BPEL process, for example:



Finally, the process resembles the following process, shown in the swim-lane format:

Figure 4–180 The Synchronous BPEL Process



4.5.15 Controlling the Order in which Files Are Processed

The File/FTP Adapter enables you to control the order in which files get processed. For example, you might want the files to be processed in sequience of their modified times/ file sizes , or other determiners.

The File/FTP adapter enables you to achieve controlling the order in which files gets processed through a FileSorter attribute that you can define in the JCA file for your inbound File/Ftp Adapter service.

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

  <connection-factory location="eis/FileAdapter" UIincludeWildcard="*.*" adapterRef=""/>
  <endpoint-activation portType="Read_ptt" operation="Read">
    <activation-spec className="oracle.tip.adapter.file.inbound.FileActivationSpec">
      <property name="UseHeaders" value="false"/>
      <property name="PhysicalDirectory" value="/tmp/flat/in"/>
      <property name="Recursive" value="true"/>
      <property name="SingleThreadModel" value="true"/>
      <property name="ListSorter" value="file.sorter.FileSizeSorter"/>
      <property name="DeleteFile" value="true"/>
      <property name="IncludeFiles" value=".*\..*"/>
      <property name="PollingFrequency" value="1"/>
      <property name="MinimumAge" value="0"/>
    </activation-spec>
  </endpoint-activation>

</adapter-config>
```

Required

The File/FTP Adapter provides two predefined sorters that use the last modified times--or example:

```
<adapter-config name="FlatStructureIn" adapter="File Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/FileAdapter" UIincludeWildcard="*.*" adapterRef=""/>
  <endpoint-activation portType="Read_ptt" operation="Read">
    <activation-spec className="oracle.tip.adapter.file.inbound.FileActivationSpec">
      [..]
      <property name="ListSorter" value="oracle.tip.adapter.file.inbound.listing.TimestampSorterAscendi
      [or]
      <property name="ListSorter" value="oracle.tip.adapter.file.inbound.listing.TimestampSorterDescen
      [..]
    </activation-spec>
  </endpoint-activation>

</adapter-config>
```

However, there are times when you would like to define the order yourself. You can implement a Java Comparator and register that with the File Adapter as described below:

1. Write a comparator. For example, the `FileSizeSorter` comparator below sorts the files in descending order of their sizes:

```
package file.sorter;


import java.util.Comparator;
import oracle.tip.adapter.file.FileInfo;

public class FileSizeSorter implements Comparator {
        public FileSizeSorter() {
            System.out.println("FileSizeSorter constructed");
        }
        public int compare(Object a, Object b) {
                FileInfo first = (FileInfo)a;
                FileInfo second = (FileInfo)b;
                if(first.getSize() < second.getSize())
                        return 1;
                else if(first.getSize() > second.getSize())
                        return -1;
                return 0;

        }
}
```

2. To compile this class, you must place `fileAdapter`.jar in the classpath. From `[FMW_HOME]/AS11gR1SOA/soa/connectors`, expand `FileAdapter.rar`, for example, `jar xvf FileAdapter.rar` to extract `fileAdapter.jar`. You must place `fileAdapter.jar` in your classpath to compile the `FileSizeSorter.java` class, for example:

   ```
   setenv CLASSPATH fileAdapter.jar
   jacac -d .FileSizeSorter.java
   ```

3. After compilation, bundle the class in its own jar file, for example,

   ```
   jar cvf fileadapter-sorter.jar file
   ```

4. Bundle the `fileadapter-sorter.jar` in the `FileAdapter.rar`, for example,

   ```
   jar uvf FileAdapter.rar fileadapter-sorter.jar
   ```

   This step is required since the `fileadapter-sorter.jar` becomes visible to the ClassLoader that loads `FileAdapter.rar`. Though, there are other ways for example, copying the `fileadapter-sorter.jar` and `fileadapter.jar` under `[DOMAIN_HOME]/lib`, but the one above is the simplest and easiest to do.

5. Setting `SingleThreadModel` as `true` in the JCA file (see image at the beginning of this section.)
6. If you're using BPEL, ensure you model a synchronous process.

5 Oracle JCA Adapter for Sockets

This chapter describes how to use Oracle JCA Adapter for Sockets (Oracle Socket Adapter), which works with Oracle BPEL Process Manager (Oracle BPEL PM) and Oracle Mediator (Mediator) as an external service.

This chapter includes the following sections:

- Section 5.1, "Introduction to Oracle Socket Adapter"
- Section 5.2, "Oracle Socket Adapter Features"
- Section 5.3, "Oracle Socket Adapter Concepts"
- Section 5.4, "Configuring Oracle Socket Adapter"
- Section 5.5, "Oracle Socket Adapter Use Cases"

5.1 Introduction to Oracle Socket Adapter

Oracle Socket Adapter is a JCA 1.5 compliant adapter for modeling standard or nonstandard protocols for communication over TCP/IP sockets. You can use an Oracle Socket Adapter to create a client or a server socket, and establish a connection. The data that is transported can be text or binary.

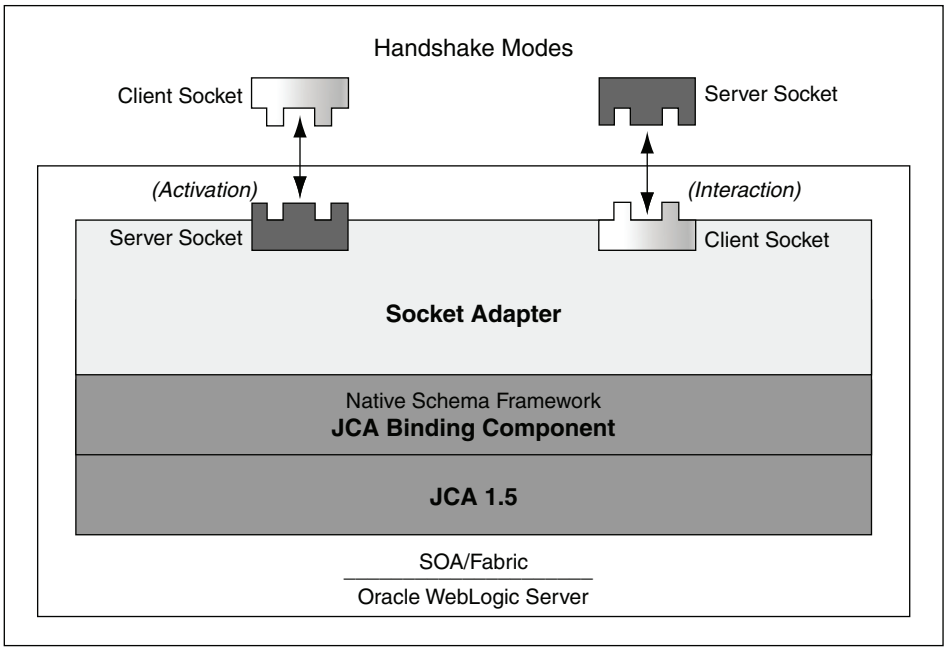
This section includes the following topics:

- Section 5.1.1, "Oracle Socket Adapter Architecture"
- Section 5.1.2, "Oracle Socket Adapter Integration with Mediator"
- Section 5.1.3, "Oracle Socket Adapter Integration with Oracle BPEL PM"
- Section 5.1.4, "Oracle Socket Adapter Integration with SOA Composite"

5.1.1 Oracle Socket Adapter Architecture

Oracle Socket Adapter is based on the JCA 1.5 architecture. JCA provides a standard architecture for integrating heterogeneous enterprise information systems (EIS). The JCA Binding Component of the Oracle Socket Adapter exposes the underlying JCA interactions as services (`WSDL` with JCA binding) for Oracle BPEL PM integration. Figure 5–1 illustrates the architecture of Oracle Socket Adapter. For details about the Oracle JCA Adapter architecture, see Section 1.2.1.1, "Architecture."

Figure 5–1 Oracle Socket Adapter Architecture



5.1.2 Oracle Socket Adapter Integration with Mediator

Oracle Socket Adapter is automatically integrated with Mediator. When you create an Oracle Socket Adapter service in JDeveloper Designer, the Adapter Configuration Wizard is started. This wizard enables you to configure the Oracle Socket Adapter. When configuration is complete, a WSDL file of the same name is created in the Application Navigator section of Oracle JDeveloper (JDeveloper). This WSDL file contains the configuration information you specify in the Adapter Configuration Wizard.

The Operation Type page of the Adapter Configuration Wizard prompts you to select an operation to perform. Based on your selection, different Adapter Configuration Wizard pages appear and prompt you for configuration information.

For more information about Oracle JCA Adapter integration with Mediator, see Section 3.2, "Adapter Integration with Oracle Fusion Middleware."

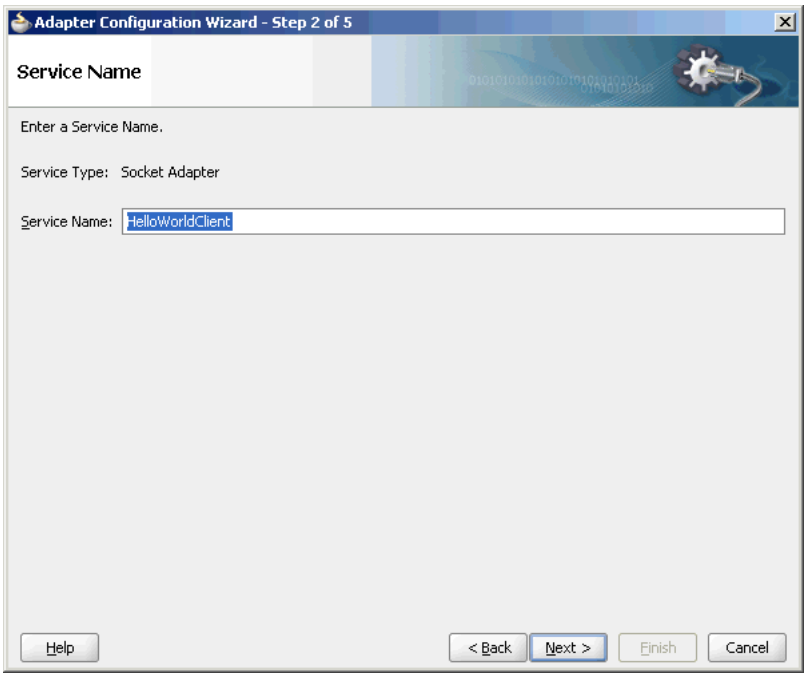
5.1.3 Oracle Socket Adapter Integration with Oracle BPEL PM

Oracle Socket Adapter is automatically integrated with Oracle BPEL PM. When you drag and drop Socket Adapter from the Component Palette of JDeveloper BPEL Designer, the Adapter Configuration Wizard starts with a Welcome page, as shown in Figure 5–2.

Figure 5–2 The Adapter Configuration Wizard - Welcome Page

This wizard enables you to configure an Oracle Socket Adapter. The Adapter Configuration Wizard then prompts you to enter a service name, as shown in Figure 5–3.

Figure 5–3 The Adapter Configuration Wizard Service Name Page



When configuration is complete, a WSDL file of the same name is created in the Application Navigator section of JDeveloper. This WSDL file contains the configuration information you specify in the Adapter Configuration Wizard.

The Operation Type page of the Adapter Configuration Wizard prompts you to select an operation to perform. Based on your selection, different Adapter Configuration Wizard pages appear and prompt you for configuration information.

For more information about Oracle JCA Adapter integration with Oracle BPEL PM, see Section 3.2, "Adapter Integration with Oracle Fusion Middleware."

5.1.4 Oracle Socket Adapter Integration with SOA Composite

A composite is an assembly of services, service components (Oracle BPEL PM and Mediator), wires, and references designed and deployed in a single application. The composite processes the information described in the messages. The details of the composite are stored in the `composite.xml` file. For more information on integration of the Oracle Socket Adapter with SOA composite, see Section 3.2.4, "Oracle SOA Composite Integration with Adapters."

5.2 Oracle Socket Adapter Features

Oracle Socket Adapter enables you to configure a BPEL process or a Mediator service to read and write data over TCP/IP sockets. It includes the following features:

- Allows modeling of standard or nonstandard protocols for communication over TCP/IP sockets
- Supports both inbound and outbound communication
- Allows you to model complex protocol handshakes declaratively, by using XSL
- Allows you the option of plugging in custom Java code to model a protocol handshake
- Provides support for reading and writing native data over sockets as the adapter is integrated with the translator infrastructure (NXSD)
- Supports multiple character encoding

5.3 Oracle Socket Adapter Concepts

This section describes the following Oracle Socket Adapter concepts:

- Section 5.3.1, "Communication Modes"
- Section 5.3.2, "Mechanisms for Defining Protocols"
- Section 5.3.3, "Character Encoding and Byte Order"
- Section 5.3.4, "Performance Tuning"

5.3.1 Communication Modes

Oracle Socket Adapter supports inbound and outbound communication over sockets that can be unidirectional or bidirectional. The communication modes of Oracle Socket Adapter are discussed in the following sections:

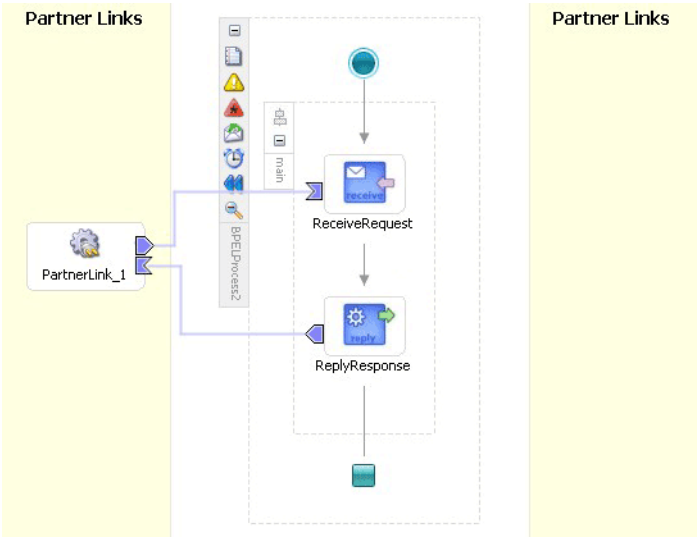
- Section 5.3.1.1, "Inbound Synchronous Request/Response"
- Section 5.3.1.2, "Outbound Synchronous Request/Response"
- Section 5.3.1.3, "Inbound Receive"
- Section 5.3.1.4, "Outbound Invoke"

5.3.1.1 Inbound Synchronous Request/Response

As part of inbound activation, the Oracle Socket Adapter opens a server socket and waits for incoming connections. The adapter uses the connection to the server socket and reads the request message, which is published to BPEL or Mediator. The Oracle Socket Adapter then uses the same connection to send the response back synchronously.

Figure 5–4 illustrates an inbound synchronous request/response scenario.

Figure 5–4 BPEL Scenario of Inbound Synchronous Request/Response

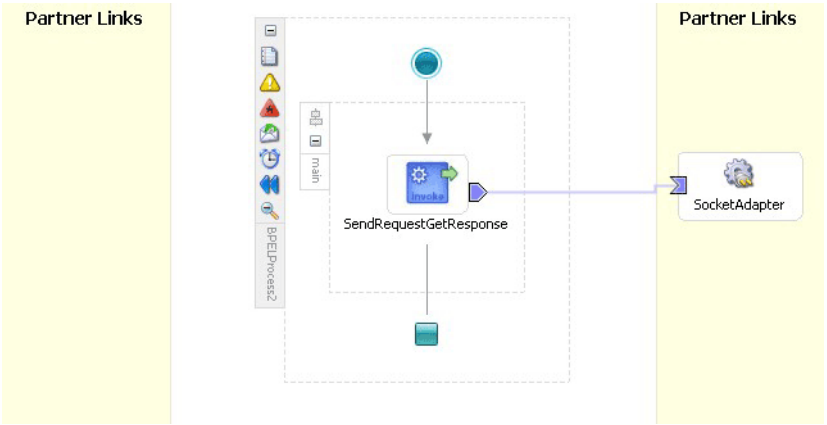


5.3.1.2 Outbound Synchronous Request/Response

In the case of outbound synchronous request/response, a request comes from BPEL or Mediator. The Oracle Socket Adapter connects to the server socket to send the request message to the server socket on the output stream. The Oracle Socket Adapter then blocks the response from the server socket on the input stream and publishes the response back to BPEL or Mediator.

Figure 5–5 illustrates an outbound synchronous request/response scenario.

Figure 5–5 BPEL Scenario of Outbound Synchronous Request/Response



5.3.1.3 Inbound Receive

As part of inbound activation, the Oracle Socket Adapter opens a server socket and waits for incoming connections. The adapter uses the connection to the server socket and reads the request message, which is published to BPEL or Mediator. In this scenario, no reply is sent.

5.3.1.4 Outbound Invoke

In the case of an outbound one way invoke scenario, the request comes from BPEL or Mediator. Oracle Socket Adapter connects to the server socket and sends the request message to the server socket on the output stream without expecting a reply.

5.3.2 Mechanisms for Defining Protocols

Communication protocols or handshakes consist of different discrete steps such as authentication procedures, acknowledgments, and sending or receiving data depending on conditions. Oracle Socket Adapter supports the following mechanisms to define the protocol handshakes.

- Protocol with Handshake Mechanism Using Style Sheet
- Protocol with Handshake Mechanism Using Custom Java Code
- Protocol Without Handshake Mechanism

5.3.2.1 Protocol with Handshake Mechanism Using Style Sheet

Oracle Socket Adapter can be configured to use a protocol designed with a handshake mechanism, defined using style sheets that use XPath Extension functions exposed by the adapter. This can be granular read and write operation on the socket I/O stream or till the end of the stream. These functions also enable you to use native format constructs for reading and writing data. This handshake mechanism uses XSLT constructs to define operations such as assignments, validations, and control flow.

You can use the XPath Extension functions with the translator infrastructure in the following ways:

- By using StyleReader, which is exposed by the NXSD framework, to read and write from the socket stream using the following methods:
  - `socketRead(nxsdStyle:String, nxsdStyleAttributes:String):String`

    You can use this method to read from the socket input stream.
  - `socketWrite(value:String, nxsdStyle:String, nxsdStyleAttributes:String):String`

    You can use this method to write to the socket output stream.

  The XSLT shown in Figure 5–6 demonstrates the usage of extension functions that use StyleReader.

Figure 5–6 XSLT with Extension Functions That Use StyleReader

```
<?xml version="1.0" encoding="windows-1252" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:ora="http://www.oracle.com/XSL/Transform/java/"
                xmlns:socket="http://www.oracle.com/XSL/Transform
                /java/oracle.tip.adapter.socket.ProtocolTranslator"
                xmlns="http://xmlns.oracle.com/HelloWorld/">
 <!-- Root template -->
 <xsl:template match="/">
  <SynchronousRequestResponseProcessRequest>
   <input>
    <xsl:value-of select="socket:socketRead('terminated', 'terminatedBy=;')"/>
   </input>
  </SynchronousRequestResponseProcessRequest>
  <!-- Copy input value into a temporary variable -->
  <xsl:variable name="temp">
   <xsl:value-of select="input"/>
  </xsl:variable>
  <!-- Concat 'Hello ' to the input string and build the response -->
  <SynchronousRequestResponseProcessResponse>
   <result>
    <xsl:value-of select="concat('Hello ', $temp)"/>
   </result>
  </SynchronousRequestResponseProcessResponse>
 </xsl:template>
</xsl:stylesheet>
```

- By annotating the schema, which defines the input and output variables, using NXSD constructs to read and write from the socket stream using the following methods:
  - `socketReadWithXlation():DocumentFragment`

    You can use this method to read from the socket input stream by using the schema and schema element configured for input.
  - `socketWriteWithXlation(xml:NodeList)`

    You can use this method to write to the socket output stream by using the schema configured for output.

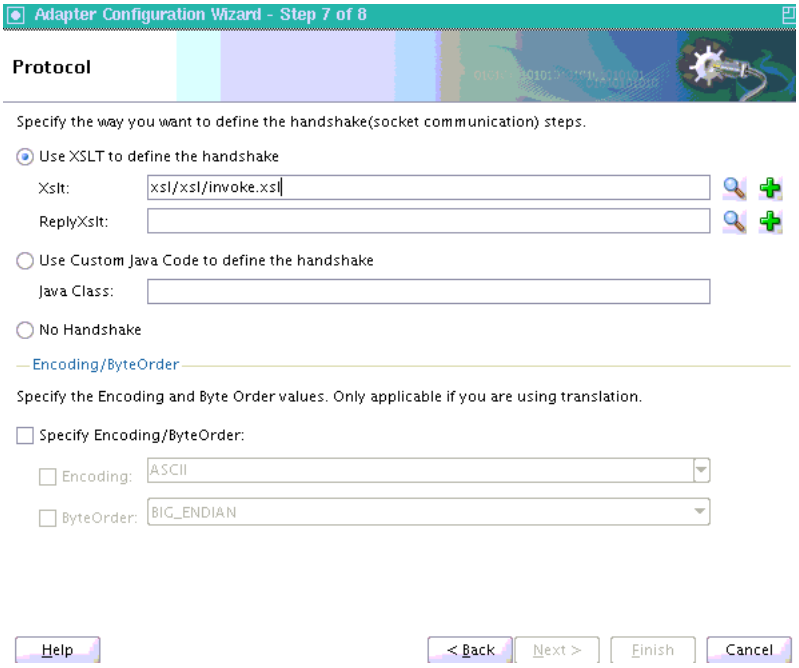
  The XSD file shown in Figure 5–7 demonstrates the usage of extension functions by annotating the schema, which defines the input and output variables, using NXSD constructs.

Figure 5–7 XSD with Extension Functions That Do Not Use StyleReader

```
<schema attributeFormDefault="unqualified" elementFormDefault="qualified"
        targetNamespace="http://xmlns.oracle.com/HelloWorldl"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd" nxsd:stream="chars"
        nxsd:version="NXSD">
 <element name="HelloWorldlProcessRequest">
  <complexType>
   <sequence>
    <element name="input" type="string" nxsd:style="fixedLength"
             nxsd:length="15"/>
   </sequence>
  </complexType>
 </element>
 <element name="HelloWorldlProcessResponse">
  <complexType>
   <sequence>
    <element name="result" type="string" nxsd:style="fixedLength"
             nxsd:length="15"/>
   </sequence>
  </complexType>
 </element>
</schema>
```

To define a handshake using style sheet, you must select **Use XSLT to define the handshake** and browse to select the XSL file in the Protocol page, as shown in Figure 5–8.

Figure 5–8 Defining a Protocol with Handshake Mechanism By Using a Style Sheet



5.3.2.2 Protocol with Handshake Mechanism Using Custom Java Code

Oracle Socket Adapter can be configured to use a protocol with a customized handshake mechanism, defined by plugging in custom Java code. The custom Java code must implement `oracle.tip.pc.services.translation.util.ICustomParser`, the

ICustomParser interface, provided by Oracle Socket Adapter, which enables custom implementation of handshakes.

> **Note:** The ICustomParser interface files are in the `bpm-infra.jar` file. This jar file is available in the following directory:
>
> *$SOA_ORACLE_HOME/soa/modules/oracle.soa.fabric_11.1.1*

The following methods must be implemented based on the appropriate communication paradigm:

- `public Element executeOutbound(InputStream in, OutputStream out, Element payLoad) throws Exception;`

  The outbound handshake must implement this method.

  Example:

```
public Element executeOutbound(InputStream in, OutputStream out, Element
payLoad) throws Exception {
    BufferedReader in1 = new BufferedReader(new InputStreamReader(in));
    PrintWriter out1 = new PrintWriter(new OutputStreamWriter(out));

    out1.println(payLoad.getFirstChild().getNodeValue());

    String retVal = in1.readLine();

    StringBuffer strBuf = new StringBuffer();
    strBuf.append("<?xml version='1.0' encoding='" + enc + "' ?>"
            + "<out xmlns='http://xmlns.oracle.com/EchoServer/'>");
        strBuf.append(retVal + "</out>");

    DOMParser parser = new DOMParser();
    parser.setValidationMode(DOMParser.NONVALIDATING);
        Element elem = (Element) parser.getDocument().getElementsByTagName(
            "out").item(0);

    return elem;
}
```

- `public Element executeInboundRequest(InputStream in) throws Exception;`

  The inbound request must implement this method.

  Example:

```
public Element executeInboundRequest(InputStream in) throws Exception {
    BufferedReader in1 = new BufferedReader(new InputStreamReader(in));

    String input = in1.readLine();

    StringBuffer strBuf = new StringBuffer();
    strBuf.append("<?xml version='1.0' encoding='" + enc + "' ?>"
            + "<EchoClientProcessRequest
xmlns='http://xmlns.oracle.com/EchoClient'>");

        strBuf.append("<input>" + input +
"</input></EchoClientProcessRequest>");
```

```
    DOMParser parser = new DOMParser();
    parser.setValidationMode(DOMParser.NONVALIDATING);
    parser.parse(new InputSource(new StringReader(strBuf.toString())));
    Element elem = (Element) parser.getDocument().getElementsByTagName(
            "EchoClientProcessRequest").item(0);

    return elem;
}
```

- `public void executeInboundReply(Element payLoad, OutputStream out) throws Exception;`

  The inbound reply must implement this method.

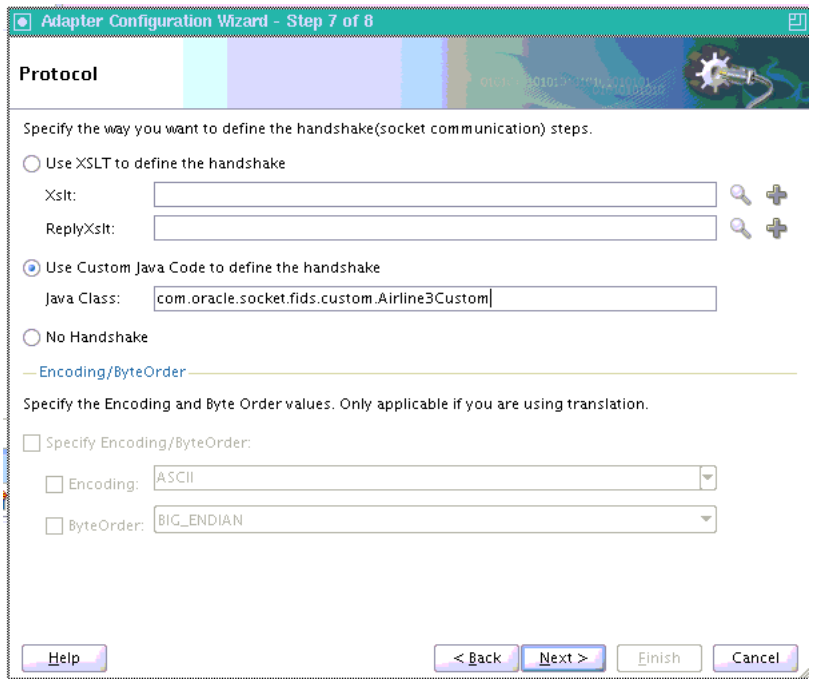
  Example:

```
public void executeInboundReply(Element payLoad, OutputStream out) throws
Exception {
    PrintWriter out1 = new PrintWriter(new OutputStreamWriter(out));

    NodeList list = payLoad.getChildNodes();
    String retVal = null;
    for(int i = 0; i < list.getLength(); i++) {
        Node node = list.item(i);
        NodeList list1 = node.getChildNodes();
        for(int j = 0; j < list1.getLength(); j++) {
            Node node1 = list1.item(j);
            if(node1.getNodeType() == Node.TEXT_NODE) {
                    retVal = node1.getNodeValue();
            }
        }
    }
    out1.println(retVal);
    out1.flush();
}
```

  Note: `in` is the handle to the socket input stream and `out` is the handle to the socket output stream.

To use a custom Java code to define a handshake, you must select **Use Custom Java Code to define the handshake** and specify the Java class implementing the handshake in the **Java Class** field, as shown in Figure 5–9.

Figure 5–9 Defining a Protocol with Handshake Mechanism By Using Custom Java Code

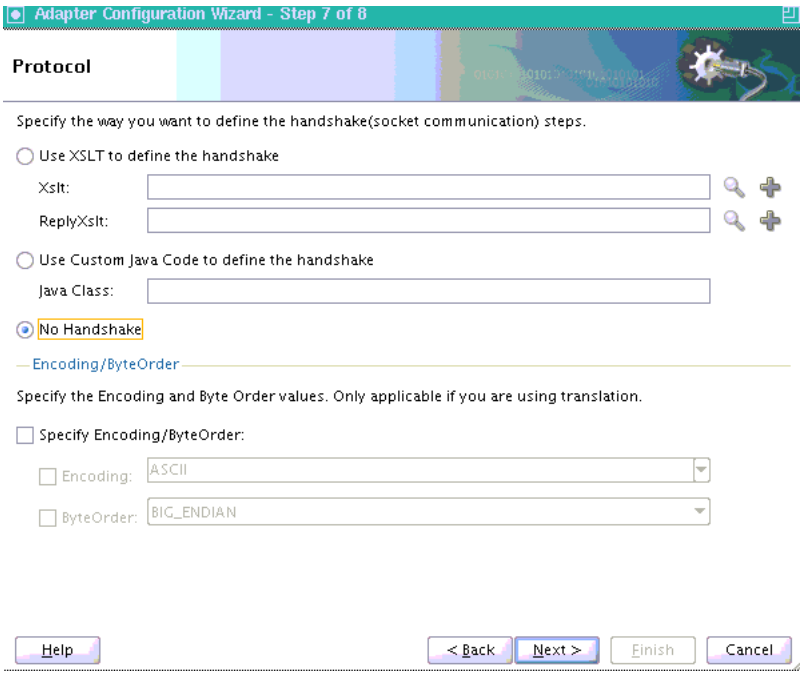


5.3.2.3 Protocol Without Handshake Mechanism

Oracle Socket Adapter can be configured to use protocols that do not require handshakes involving translation to and from the socket I/O stream.

To use a protocol that does not require a handshake, you must select **No Handshake** in the Protocol page, as shown in Figure 5–10.

Figure 5–10 Defining a Protocol without a Handshake Mechanism



5.3.3 Character Encoding and Byte Order

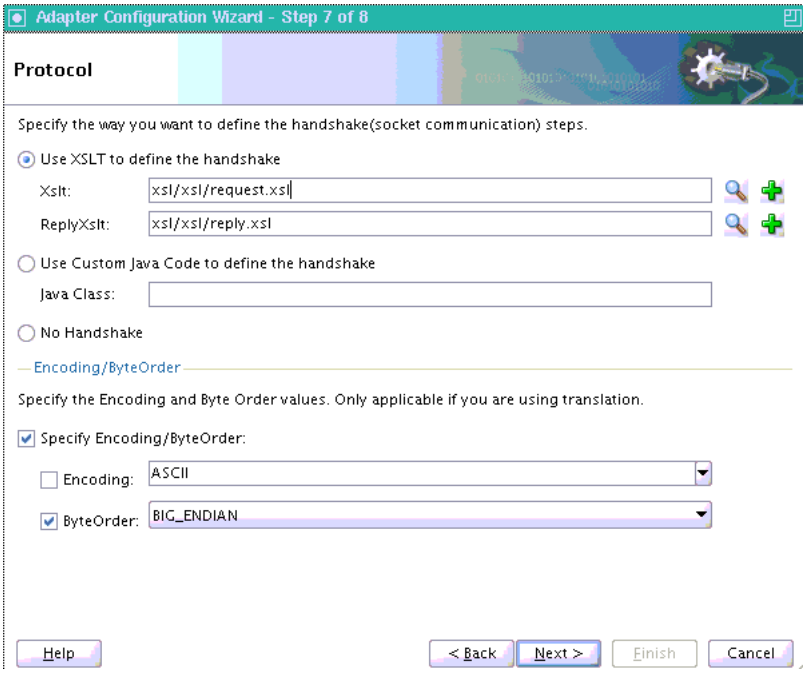
The Encoding property represents the character encoding in which native data is stored, and the ByteOrder property is the byte order of the native data, which is either `BIG_ENDIAN` or `LITTLE_ENDIAN`.

Character encoding and byte order can be specified in the schema file (NXSD), using the Native Format Builder wizard. You can also specify the encoding and the byte order to be used, by using the Adapter Configuration Wizard. When encoding and byte order are not specified, the default values are `US-ASCII` and `BIG_ENDIAN`.

To specify the encoding and byte order values, which are applicable only if you are using translation, you must perform the following steps in the Protocol page of the Adapter Configuration Wizard:

1. In the Encoding/ByteOrder section of the Protocol page, select the **Specify Encoding/Byte Order** option, as shown in Figure 5–11.

Figure 5–11 The Adapter Configuration Wizard - Protocol Page



2. Perform one of these tasks to set the encoding:
   a. To use the encoding specified in the schema file, leave Encoding unchecked.
   b. To specify the encoding using the Adapter Configuration Wizard, select **Encoding**, and then select an encoding type from the Encoding list.

   Note: If you select Encoding, then the encoding type specified using the Adapter Configuration Wizard takes precedence over the encoding type specified in the NXSD file.

3. Perform one of these tasks to set the byte order:
   a. To use the byte order specified in the schema file, select **Use Byte Order Value from the schema**.

b. To specify the byte order using the Adapter Configuration Wizard, select **ByteOrder**, and then select a byte order from the ByteOrder list.

4. Click **Finish**. Once you click Finish, the Configuration Wizard displays a page that indicates that you have finished configuring the Socket Adapter.

5.3.4 Performance Tuning

The Oracle Socket Adapter supports performance tuning features, including:

- Section 5.3.4.1, "Configuring Oracle Socket Adapter Connection Pooling"

For more information about performance tuning, see "Oracle Socket Adapter Tuning" in the *Oracle Fusion Middleware Performance and Tuning Guide*.

5.3.4.1 Configuring Oracle Socket Adapter Connection Pooling

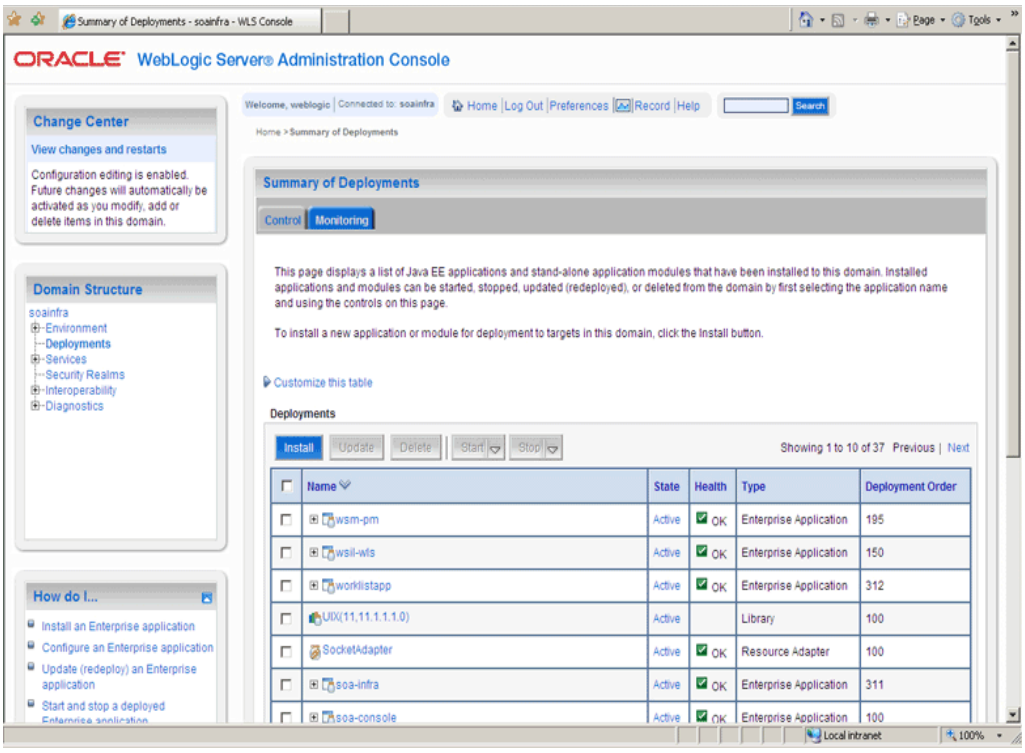
One way to optimize Oracle Socket Adapter performance is by using a Connection Pool. You can use a connection pool while the socket server you are connecting to does not close the socket with each interaction. A connection pool lets you use a socket connection repeatedly, avoiding the overhead of creating a new socket for each interaction. You must configure the connection pool for the Oracle Socket Adapter using the Oracle WebLogic Server console.

Note: The Connection Pool feature is applicable to outbound interactions only.

How to configure Oracle Socket Adapter connection pooling:

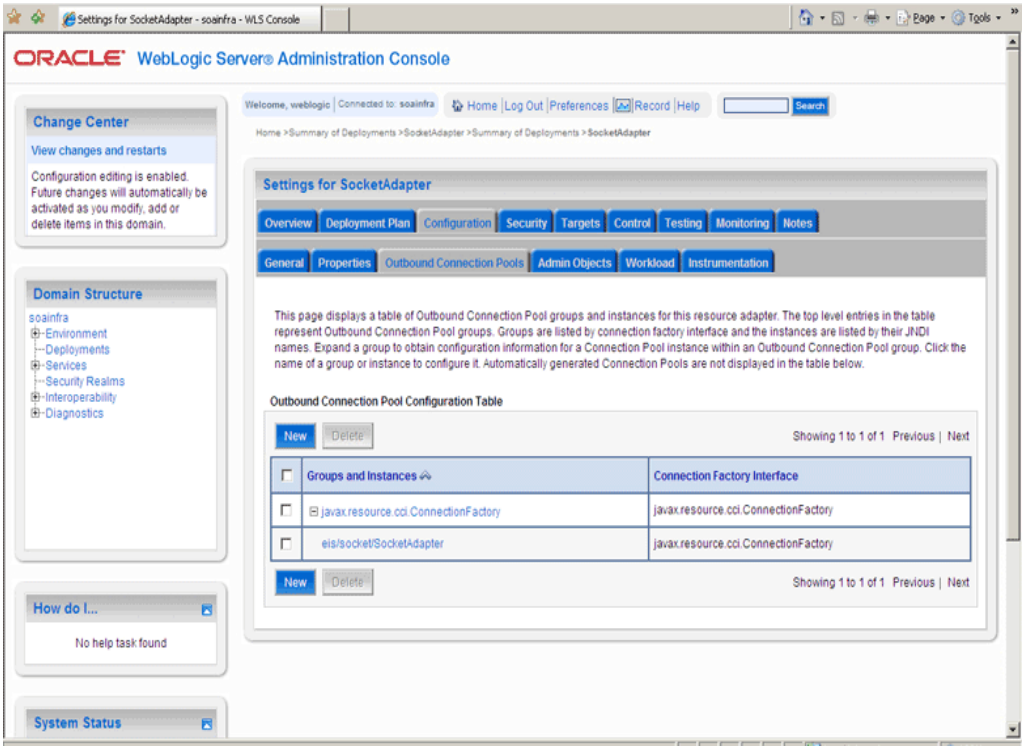
1. Log into your Oracle WebLogic Server console. To access the console navigate to `http://`*`servername`*`:`*`portnumber`*`/console`.
2. Click **Deployments** in the left pane for Domain Structure. The Summary of Deployments page is displayed.

Figure 5–12 Oracle WebLogic Server Console - Summary of Deployments Page



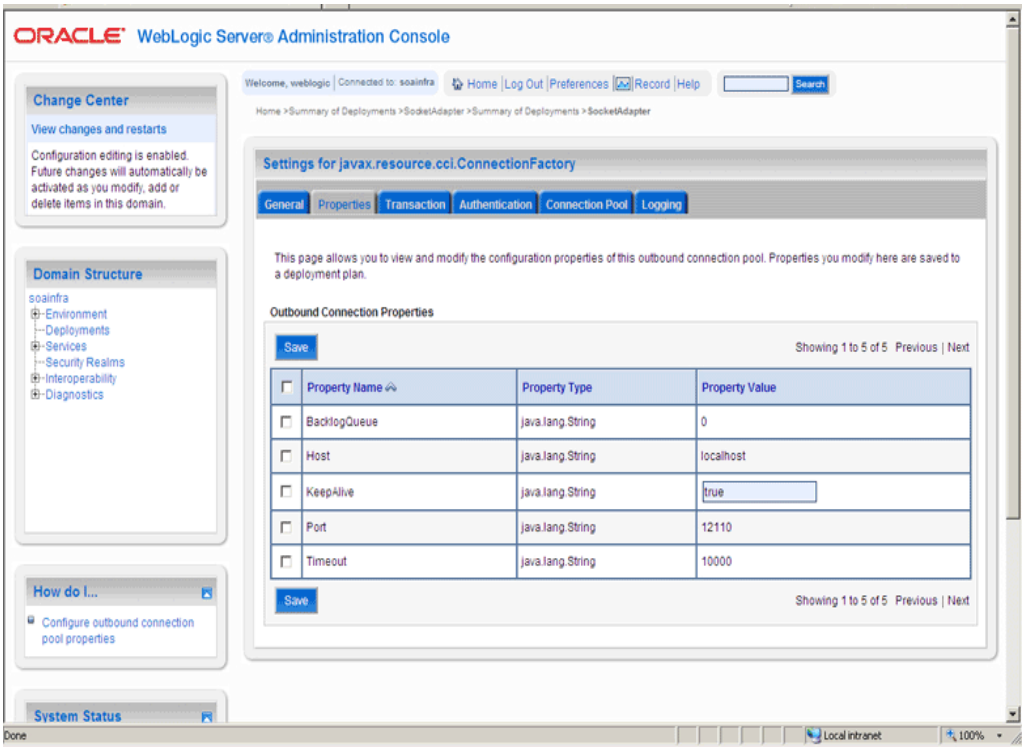
3. Click **SocketAdapter**. The Settings for SocketAdapter page is displayed.
4. Click the **Configuration** tab.
5. Click the **Outbound Connection Pools** tab, and expand **javax.resource.cci.ConnectionFactory** to see the configured connection factories, as shown in Figure 5–13:

Figure 5–13 Oracle WebLogic Server Console - Settings for SocketAdapter Page



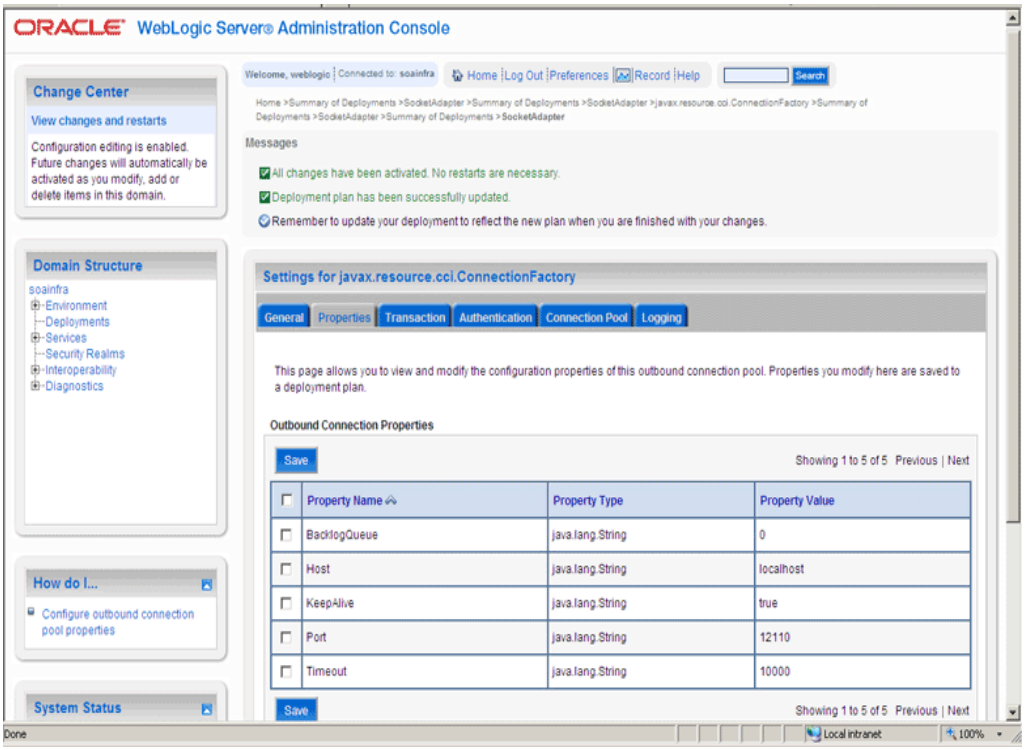
6. Click **eis/socket/SocketAdapter**. The Settings for javax.resource.cci.ConnectionFactory page is displayed.
7. Set the `KeepAlive` connection factory property to `true`, as shown in Figure 5–14. The connection pool feature for the Oracle Socket Adapter is enabled.

Figure 5–14 Oracle WebLogic Server Console - Settings for javax.resource.cci.Connectionfactory Page



8. Click **Save**. The Settings for javax.resource.cci.ConnectionFactory page is displayed with the message, `Deployment plan has been successfully updated`, as shown in Figure 5–15.

Figure 5–15 Oracle WebLogic Server Console - Settings for javax.resource.cci.Connectionfactory Page



Note: You can modify connection pool parameters by using the **Connection Pool** tab of Oracle WebLogic Server Administration Console.

5.4 Configuring Oracle Socket Adapter

The following tasks are required for configuring Oracle Socket Adapter:

- Modifying the weblogic-ra.xml File
- Modeling a Handshake
- Designing an XSL File Using the XSL Mapper Tool
- Specifying a TCP Port in a Configuration Plan For an Oracle Socket Adapter

5.4.1 Modifying the weblogic-ra.xml File

To configure Oracle Socket Adapter, you must specify the value of the properties listed in Table 5–1 in the `weblogic-ra.xml` file. You can update these properties from the Oracle WebLogic Server Administration Console. For more information, see Section 2.18, "Adding an Adapter Connection Factory."

Table 5–1 Oracle Socket Adapter Configuration Properties

Property	Description
Host	In case of outbound interaction, the system name on which the socket server is running, to which you want to connect. In case of inbound interaction, it is always `localhost`.
Port	In case of outbound interaction, it is the port number on which a socket server is running, to which an adapter connects. In case of inbound interaction, it is the port number on which the socket adapter listens for incoming connections.
Timeout	With this value set to a nonzero timeout interval, a `read()` call on the InputStream associated with this socket blocks for only this amount of time. If the timeout interval expires, then a `java.net.SocketTimeoutException` is raised though the socket is still valid. The option must be enabled before entering the blocking operation to have effect. The timeout interval must be greater than `0`. A timeout interval of `0` is interpreted as an infinite timeout. The value is in milliseconds.
KeepAlive	Applicable only in case of outbound interactions. Should be set to `true` to use connection pool feature.
BacklogQueue	Applicable in case of inbound interactions. This value indicates the maximum queue length for incoming connection indications (a request to connect). If a connection indication arrives when the queue is full, then the connection is refused.

The following is a sample `weblogic-ra.xml` file:

```
<wls:connection-instance>
                <wls:description>Socket Adapter</wls:description>
                <wls:jndi-name>eis/socket/SocketAdapter</wls:jndi-name>
                <wls:connection-properties>
                    <wls:pool-params>
                        <wls:initial-capacity>0</wls:initial-capacity>
                        <wls:max-capacity>200</wls:max-capacity>
                        <wls:capacity-increment>5</wls:capacity-increment>
                        <wls:shrinking-enabled>true</wls:shrinking-enabled>

<wls:shrink-frequency-seconds>60</wls:shrink-frequency-seconds>

<wls:connection-creation-retry-frequency-seconds>2</wls:connection-creation-retry
-frequency-seconds>

<wls:connection-reserve-timeout-seconds>5</wls:connection-reserve-timeout-seconds>

<wls:match-connections-supported>true</wls:match-connections-supported>
                        <wls:use-first-available>true</wls:use-first-available>
                    </wls:pool-params>

<wls:transaction-support>NoTransaction</wls:transaction-support>

<wls:reauthentication-support>true</wls:reauthentication-support>
                    <wls:properties>
                        <wls:property>
                            <wls:name>Host</wls:name>
                            <wls:value>localhost</wls:value>
                        </wls:property>
                        <wls:property>
                            <wls:name>Port</wls:name>
                            <wls:value>12110</wls:value>
                        </wls:property>
                        <wls:property>
                            <wls:name>Timeout</wls:name>
```

```
                    <wls:value>10000</wls:value>
                </wls:property>
                <wls:property>
                    <wls:name>BacklogQueue</wls:name>
                    <wls:value>0</wls:value>
                </wls:property>
                <wls:property>
                    <wls:name>KeepAlive</wls:name>
                    <wls:value>True</wls:value>
                </wls:property>
            </wls:properties>
            <wls:res-auth>Application</wls:res-auth>
        </wls:connection-properties>
    </wls:connection-instance>
```

Note: To set up connection pooling, you must set the `KeepAlive` property to `true`.

5.4.2 Modeling a Handshake

A handshake may be required to negotiate a connection with a client or a server socket.

5.4.2.1 Modeling an Outbound Handshake

The outbound XSLT uses an input corresponding to the invoked message. The outbound XSLT writes to the socket output stream by using extension functions. The output is dummy for unidirectional or a response for bidirectional communication.

The following example demonstrates the modeling of a Synchronous Request/Response communication paradigm:

```
<xsl:stylesheet
...
xmlns:socket="http://www.oracle.com/XSL/Transform/java/oracle.tip.adapter.socket.P
rotocolTranslator" />
xmlns:request="http://www.TragetNameSpace.com/Request" >

  <xsl:template match="/">

    <!-- Write the entire content of "books" element using translator -->
    <xsl:variable name="username" select="socket:socketWriteWithXlation(.)" />

    <!-- Read the stream using translator -->
    <xsl:copy-of select="socket:socketReadWithXlation()" />

  </xsl:template>
</xsl:stylesheet>
```

5.4.2.2 Modeling an Inbound Handshake

The inbound XSLT uses a dummy input, reads the socket input stream through extension functions, and constructs the XML record to be published.

The following example demonstrates a handshake in which the client sends across a user identification terminated by a comma (,) and a password terminated by a semicolon (;) for validation, and then sends the message payload:

```
<xsl:stylesheet
...
xmlns:socket="http://www.oracle.com/XSL/Transform/java/oracle.tip.adapter.socket.P
rotocolTranslator" />
  <xsl:template match="/">
    <!-- Read the user name -->
    <xsl:variable name="username"
select="socket:socketRead('terminated','terminatedBy=,')" />
    <!-- Read password if user name is correct -->
    <xsl:if test="normalize-space($username)='user'">
      <xsl:variable name="password"
select="socket:socketRead('terminated','terminatedBy=;')" />
      <!-- If password is correct proceed to read the payload using translator
-->
      <xsl:if test="normalize-space($password)='password'">
        <!--  Send an OK -->
        <xsl:variable name="ack1" select="socket:socketWrite('001','','')" />

        <output>
<!--  Wait for the payload -->
          <xsl:copy-of select="socket:socketReadWithXlation()" />
        </output>

      </xsl:if>
      <!--  Send an error -->
      <xsl:else><xsl:variable name="ack2"
select="socket:socketWrite('000','','')" /></xsl:else>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

5.4.3 Designing an XSL File Using the XSL Mapper Tool

You can design an XSL file by using the XSL mapper tool for Oracle Socket Adapter. The following sections describe the procedure for designing XSL for different communication scenarios:

- Section 5.4.3.1, "Designing XSL for Inbound Synchronous Request/Reply"
- Section 5.4.3.2, "Designing XSL for Outbound Synchronous Request/Reply"

5.4.3.1 Designing XSL for Inbound Synchronous Request/Reply

This section describes the procedure for designing XSL for an inbound synchronous request/reply scenario by using the XSL mapper tool:

Note: To perform this use case, you require the following files from the `artifacts.zip` file contained in the `Adapters-101SocketAdapterHelloWorld` sample:

- `artifacts/schemas/HelloWorld.xsd`

You can access the `Adapters-101SocketAdapterHelloWorld` sample on the Oracle SOA Sample Code site.

Copy this file to the `HelloWorldComposite\xsd` folder under the `HelloWorldComposite` project.

Design an SOA Composite

To design an SOA composite, perform the steps described in Section 5.5.1.2, "Designing the SOA Composite."

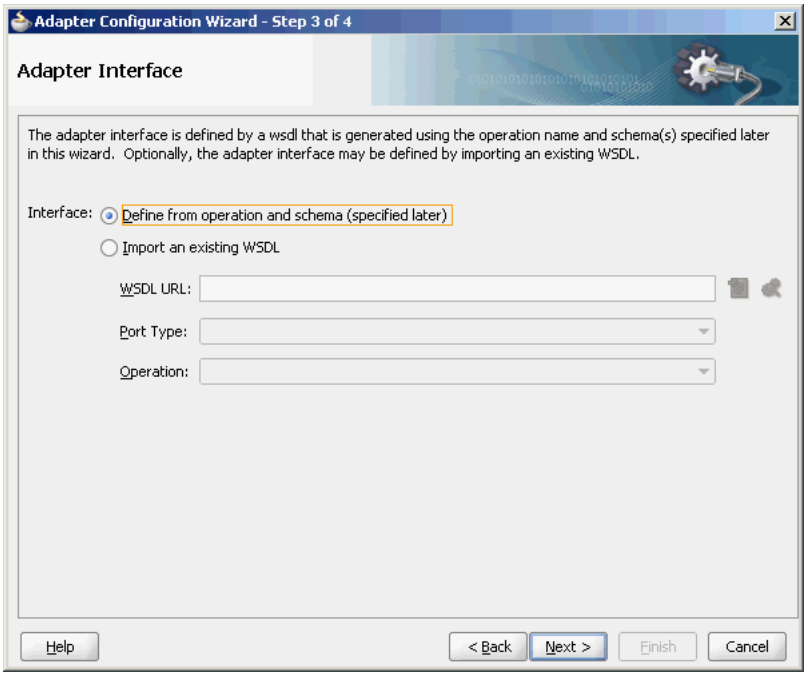
> **Note:** The steps provided in Section 5.5.1.2, "Designing the SOA Composite" are applicable to a composite with Oracle BPEL PM. Alternatively, you can create a composite with Mediator.

Create an Inbound Oracle Socket Adapter Service

To create an inbound Oracle Socket Adapter service, perform the following steps:

1. Drag and drop **Socket Adapter** from the Components Palette to the Exposed Services swim lane. The Welcome page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter the service name, `HelloWorld` in the **Service Name** field and then click **Next**. The Adapter Interface page is displayed.
4. Select **Define from operation and schema (specified later)**, as shown in the Figure 5–16, and click **Next**. The Operation page is displayed.

Figure 5–16 The Adapter Configuration Wizard - Adapter Interface Page

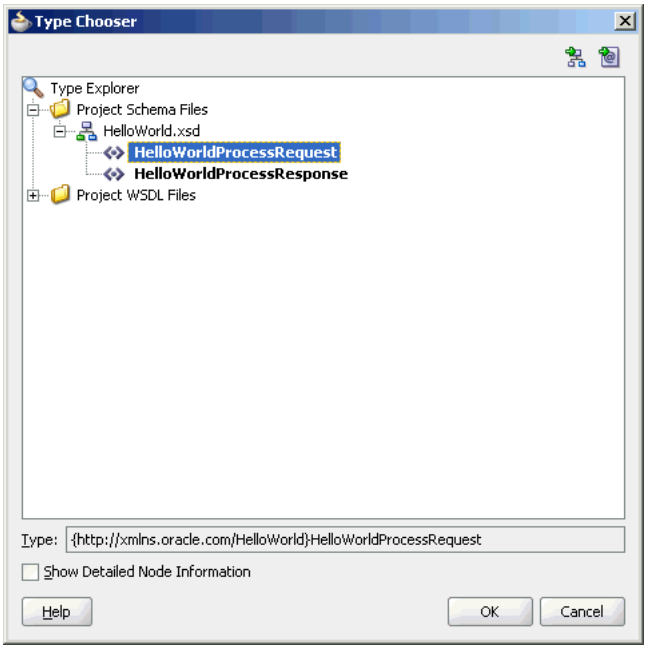


5. Select **Inbound Synchronous Request/Reply** as the Operation Type and then click **Next**. The Socket Connection page is displayed.
6. Enter `eis/socket/InboundSocketAdapter` in the **Socket Connection JNDI Name** field, as shown in Figure 5–17, and click **Next**. The Messages page is displayed.

Figure 5–17 The Adapter Configuration Wizard Socket Connection Page

7. Click **Browse For Schema File** that appears at the end of the URL field in the Request Message Schema box. The Type Chooser dialog is displayed.
8. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessRequest**, as shown in Figure 5–18.

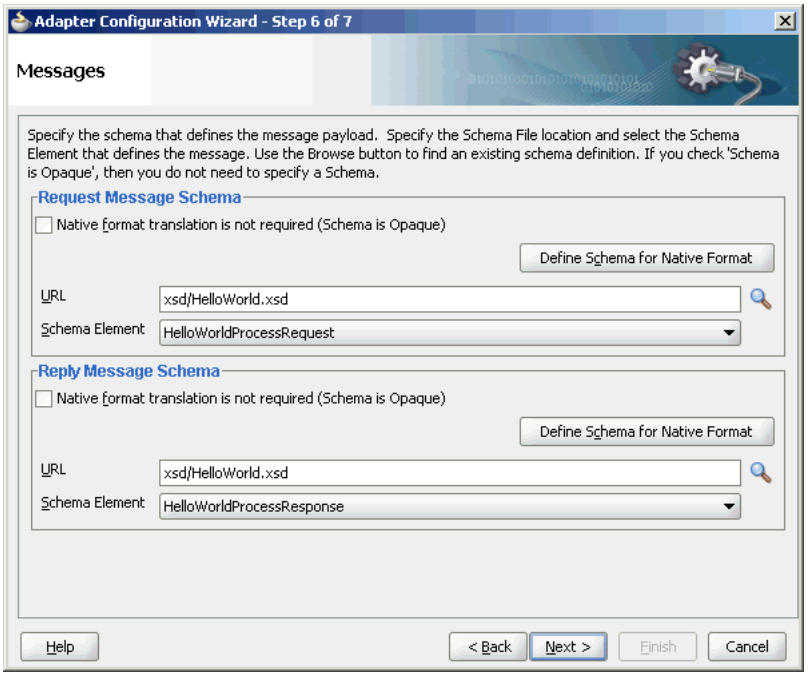
Figure 5–18 The Type Chooser Dialog



9. Click **OK**. The URL field in the Messages page is populated with the **HelloWorld.xsd** file.

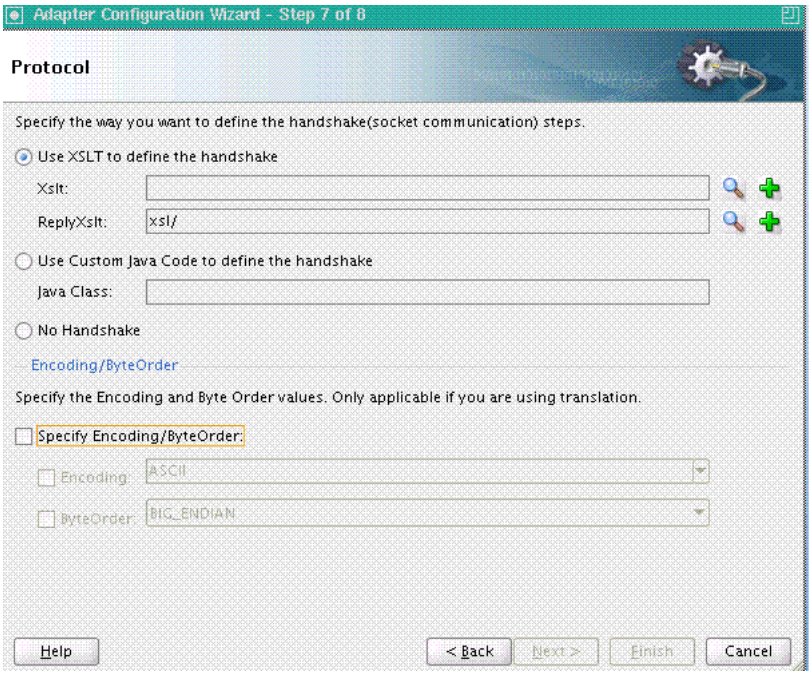
10. Click **Browse For Schema File** that appears at the end of the URL field in the Reply Message Schema box. The Type Chooser dialog is displayed.
11. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessResponse**.
12. Click **OK**. The URL fields in the Messages page are populated with the **HelloWorld.xsd** files, as shown in Figure 5–19.

Figure 5–19 The Adapter Configuration Wizard File Messages Page



13. Click **Next**. The Protocol page is displayed, as shown in Figure 5–20.

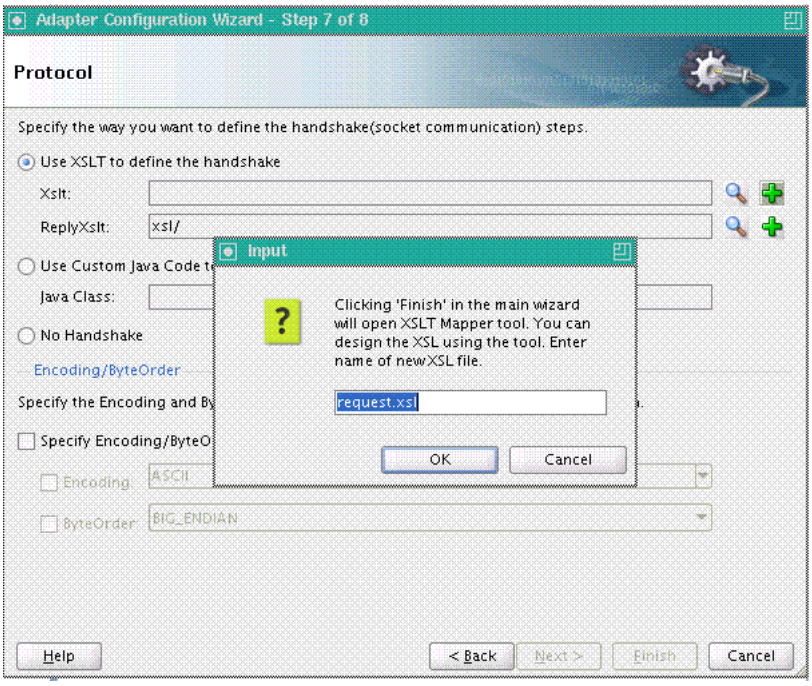
Figure 5–20 The Adapter Configuration Wizard - Protocol Page



14. Select **Use XSLT to define the handshake**.

15. Click the **create new xsl file** icon that appears at the end of the Xslt field. The Input dialog appears, as shown in Figure 5–21.

Figure 5–21 The input Dialog of the Protocol Page

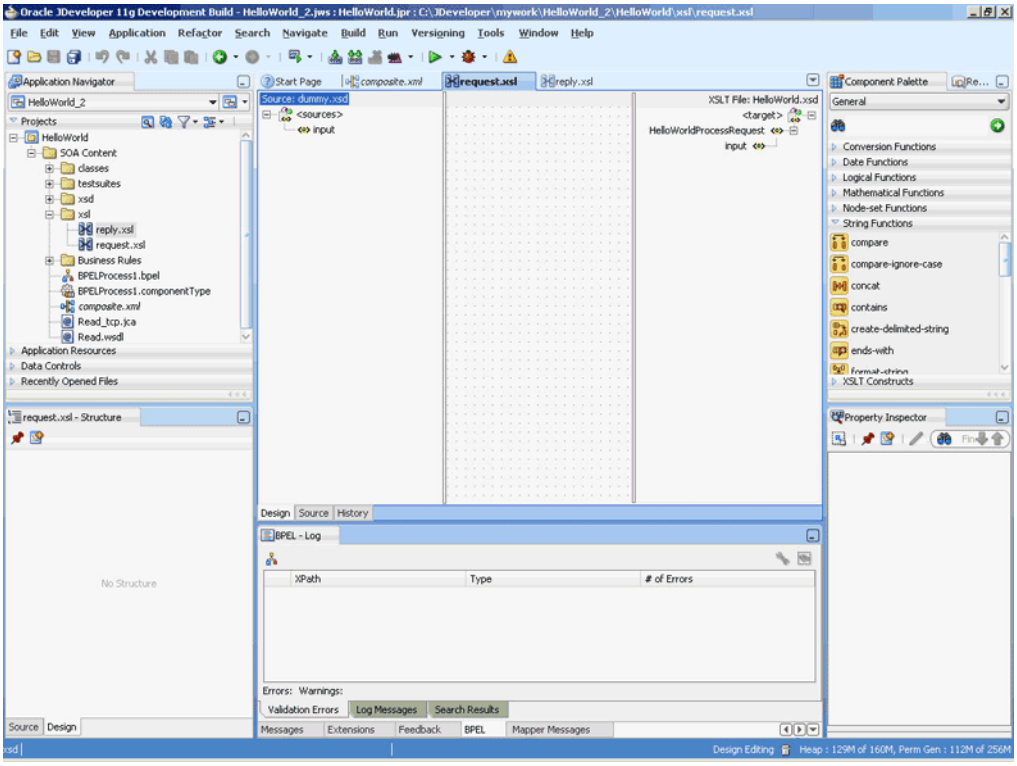


16. Use the default value, **request.xsl**, as the name of the XSL file, as shown in Figure 5–21 and click **OK**.

17. Click the **create new xsl file** icon that appears at the end of the ReplyXslt field. The Input dialog appears.
18. Use the default value, **reply.xsl**, as the name of the XSL file, and click **OK**.
19. Click **Finish**. The request.xsl and the reply.xsl files are created.

    Figure 5–22 shows the request.xsl page.

Figure 5–22 The JDeveloper - request.xsl Page

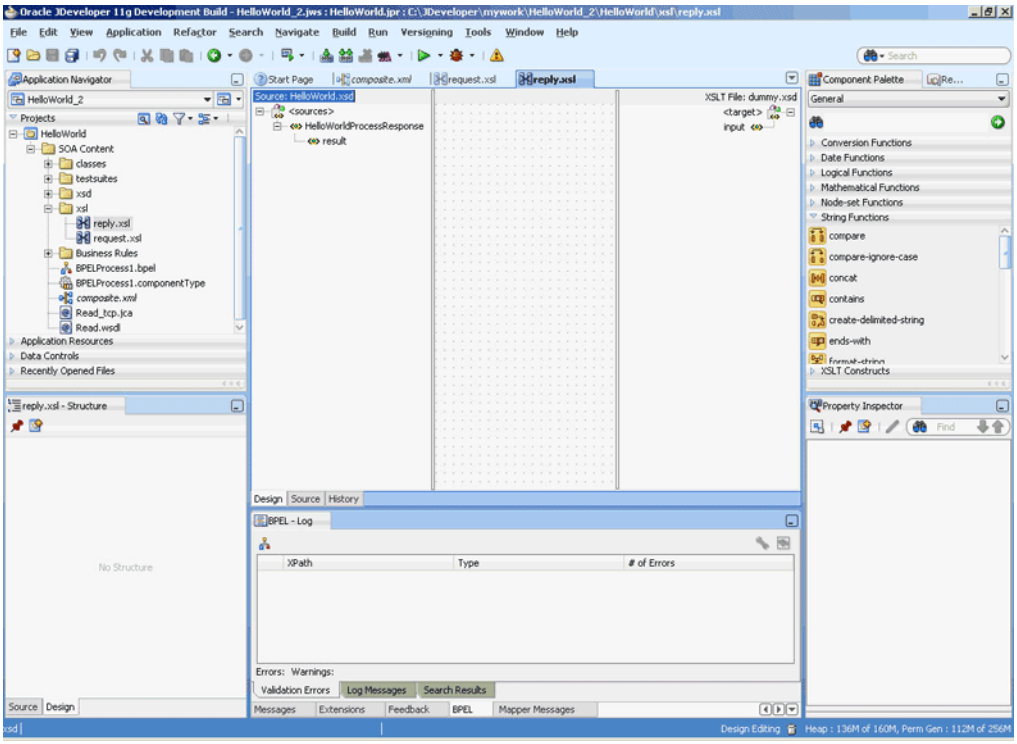


Note: A `dummy.xsd` file appears in the left Source pane of the `request.xsl` page, which is used as the source for the XSL mapper tool.

In an inbound request scenario, Oracle Socket Adapter reads native data that is received by the socket and converts it to an XML format. That is, on the source side there is no XML file. Because the XSLT mapper always needs source and target XSD files, a dummy XSD file appears in the mapper tool.

Figure 5–23 shows the reply.xsl page.

Figure 5–23 The JDeveloper - reply.xsl Page



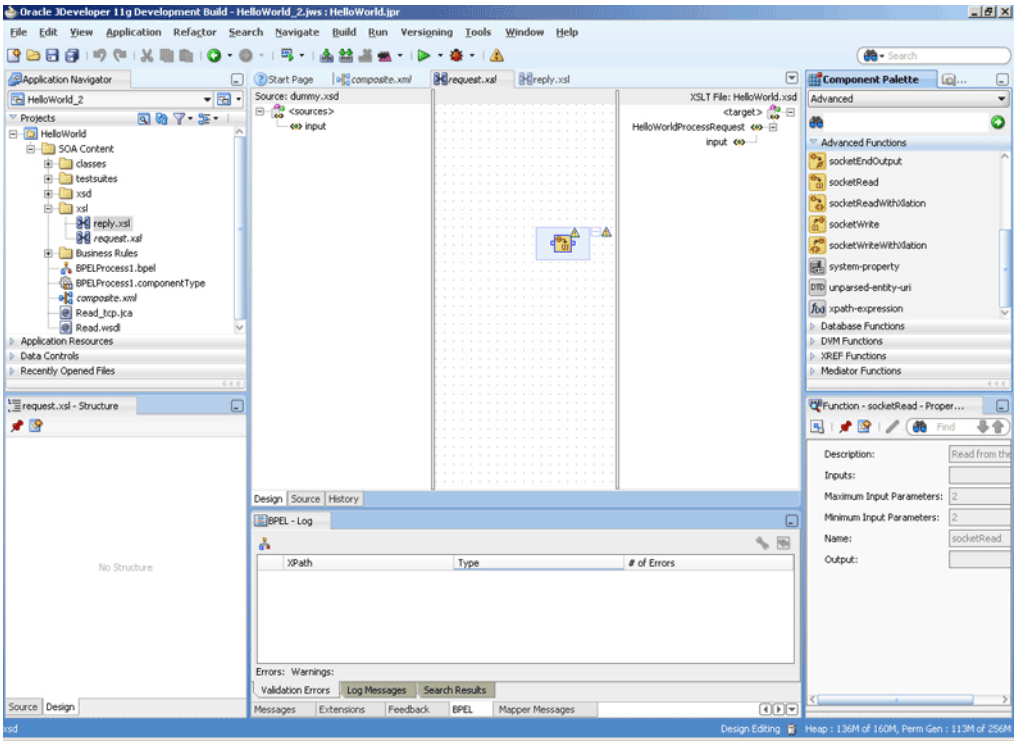
Note: A `dummy.xsd` file appears in the right target pane of the `reply.xsl` page. This `dummy.xsd` file is used as the target for the XSL mapper tool.

---

20. Define the request part of the inbound synchronous request/reply operation as follows:

a. In the request.xsl page, drag and drop **socketRead** from the Advanced Functions list of the Components Palette to the middle pane, as shown in Figure 5–24.

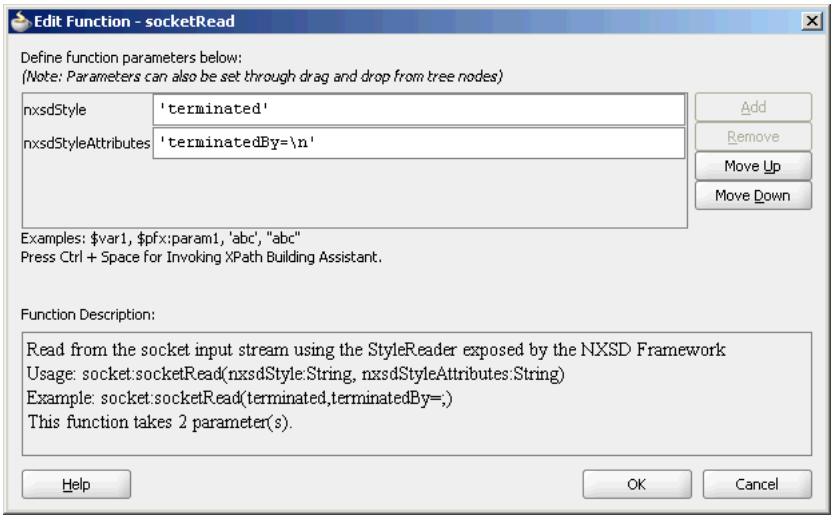
Figure 5–24 The JDeveloper - request.xsl Page



b. Double-click the **socketRead** advanced function. The Edit Function - socketRead dialog appears.

c. Enter the function parameters in the **nxsdStyle** and **nxsdStyleAttributes** fields, as shown in Figure 5–25.

Figure 5–25 The Edit Function - socketRead Dialog

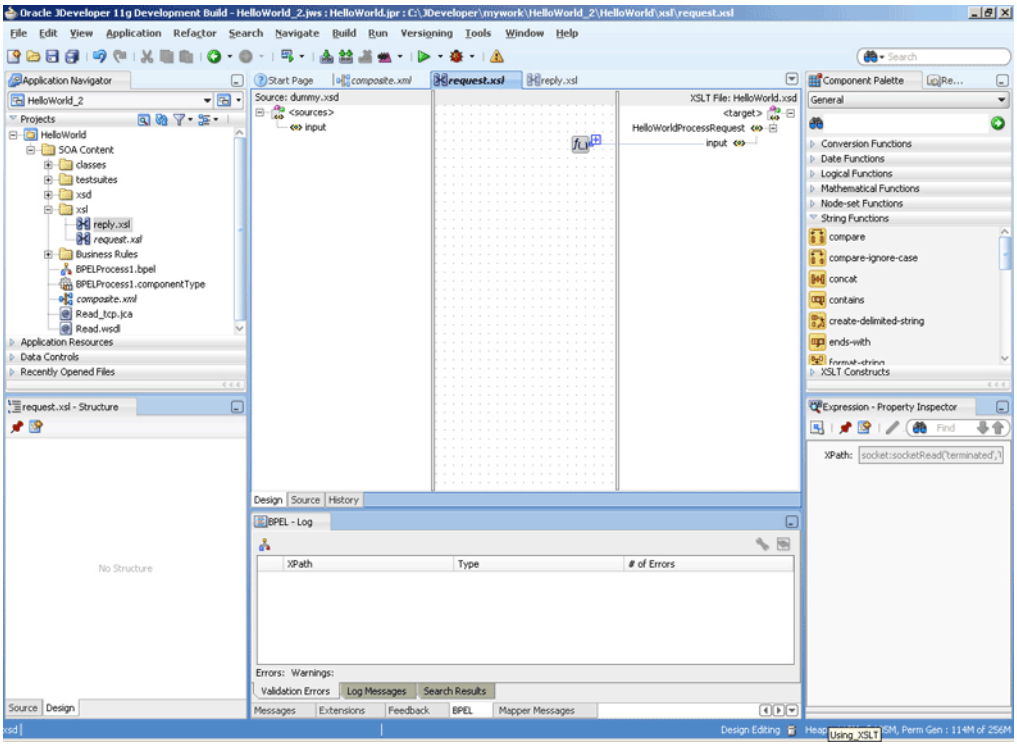


Note: The `socketRead` function reads from the socket input stream by using the `StyleReader` exposed by the NXSD framework.

d. Click **OK**. The request.xsl (XSL mapper tool) page is displayed.

e. Link the `sockRead` function in the middle pane to the target `input` node on the right pane. The request.xsl (XSL mapper tool) with the XSL mapping is displayed, as shown in Figure 5–26.

Figure 5–26 The JDeveloper - request.xsl Page



21. Define the reply part of the inbound synchronous request/reply operation as follows:

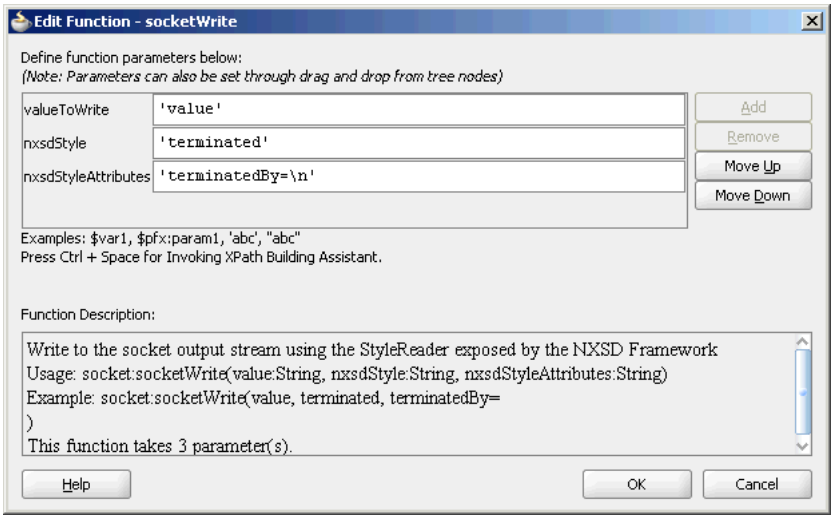
a. From the Component Palette list, select **Advanced**, and then select **Advanced Functions**. A list of advanced functions are displayed.

b. In the reply.xsl page, drag and drop **socketWrite** from the Advanced Functions list of the Component Palette to the middle pane.

c. Double-click the **socketWrite** advanced function. The Edit Function - socketWrite dialog appears.

d. Enter the function parameters in the **valueToWrite**, **nxsdStyle**, and **nxsdStyleAttributes** fields, as shown in Figure 5–27.

Figure 5–27 The Edit Function - socketWrite Dialog

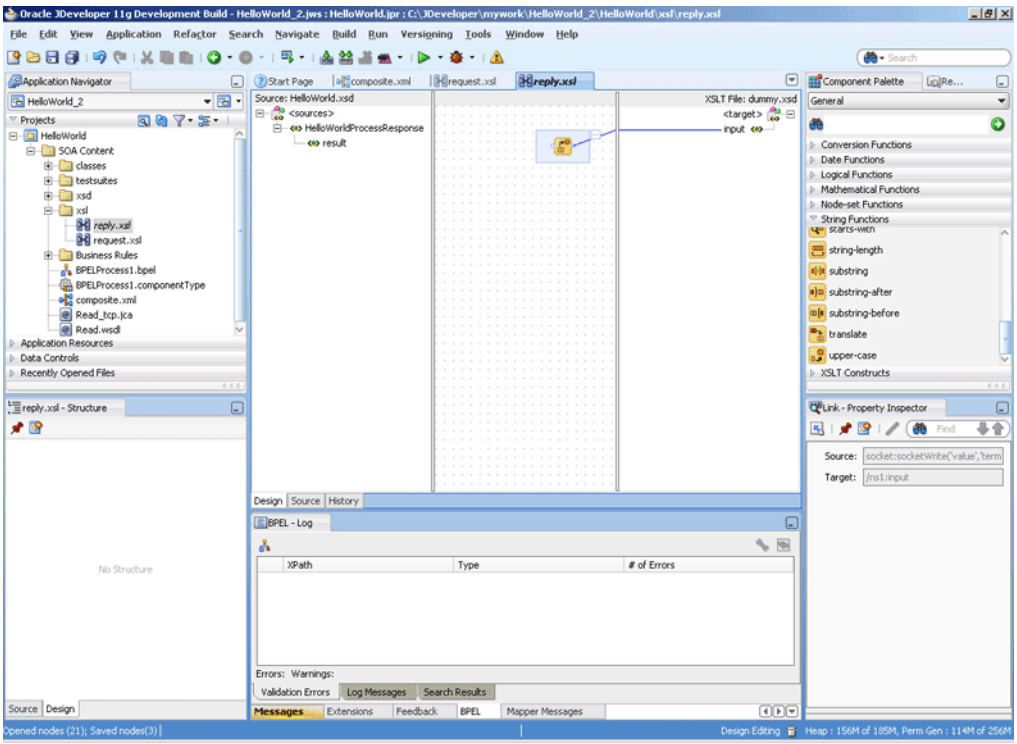


Note: The `socketWrite` function writes to the socket output stream by using the `StyleReader` exposed by the NXSD framework.

e. Click **OK**. The reply.xsl (XSL mapper tool) page is displayed.

f. Link the `sockWrite` function in the middle pane to the target `input` node on the right pane. The reply.xsl (XSL mapper tool) with the XSL mapping is displayed, as shown in Figure 5–28.

Figure 5–28 The JDeveloper - reply.xsl Page



22. Click **File**, **Save All**. The request.xsl and reply.xsl files for the inbound Oracle Socket Adapter are created.

5.4.3.2 Designing XSL for Outbound Synchronous Request/Reply

This section describes the procedure for designing XSL for an outbound synchronous request/reply scenario by using the XSL mapper tool:

Note: To perform this use case, you require the following files from the `artifacts.zip` file contained in the `Adapters-101SocketAdapterHelloWorld` sample:

- `artifacts/schemas/HellowWorld.xsd`

You can access the `Adapters-101SocketAdapterHelloWorld` sample on the Oracle SOA Sample Code site.

Copy the `HelloWorld.xsd` file to `HelloWorldComposite\xsd` under the `HelloWorldComposite` project:

Design an SOA Composite

To design an SOA composite, perform the steps described in Section 5.5.1.2, "Designing the SOA Composite.".

Note: The steps provided in Section 5.5.1.2, "Designing the SOA Composite" are applicable to a composite with Oracle BPEL PM. Alternatively, you can create a composite with Mediator.

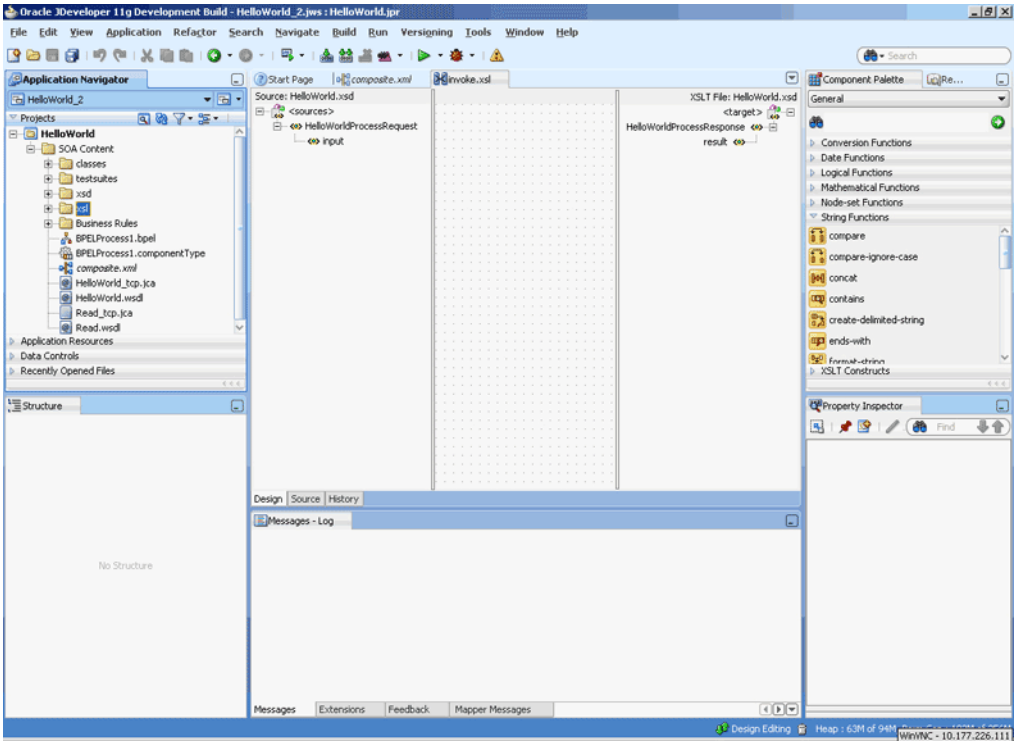
Create an Outbound Oracle Socket Adapter Reference

To create an outbound Oracle Socket Adapter reference, perform the following steps:

1. Drag and drop **Socket Adapter** from the Components Palette to the External References swim lane. The Welcome page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter the service name, `HelloWorld` in the **Service Name** field and then click **Next**. The Adapter Interface page is displayed.
4. Select **Define from operation and schema (specified later)**, as shown in the Figure 5–16 and click **Next**. The Operation page is displayed.
5. Select **Outbound Synchronous Request/Reply** as the Operation Type and then click **Next**. The Socket Connection page is displayed.
6. Enter `eis/socket/OutboundSocketAdapter` in the **Socket Connection JNDI Name** field and click **Next**. The Messages page is displayed.
7. Click **Browse For Schema File** that appears at the end of the URL field in the Request Message Schema box. The Type Chooser dialog is displayed.
8. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessRequest**, as shown in Figure 5–18.
9. Click **OK**. The URL field in the Messages page is populated with the `HelloWorld.xsd` file.

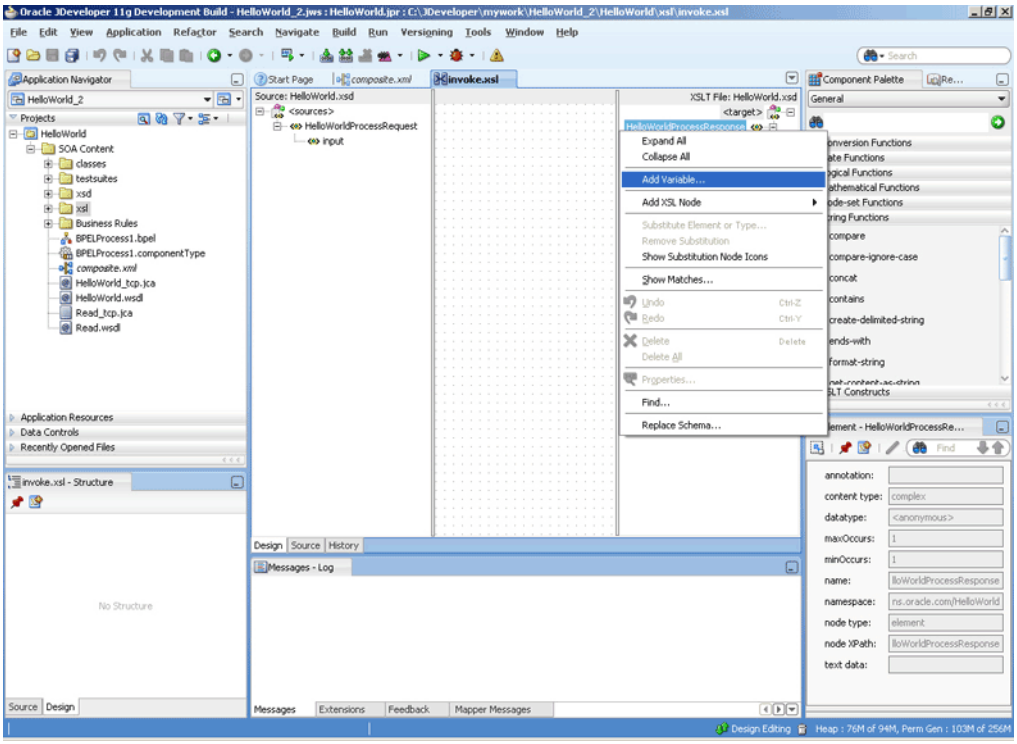
10. Click **Browse For Schema File** that appears at the end of the URL field in the Reply Message Schema box. The Type Chooser dialog is displayed.
11. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessResponse**.
12. Click **OK**. The URL fields in the Messages page are populated with the `HelloWorld.xsd` files, as shown in Figure 5–19.
13. Click **Next**. The Protocol page is displayed.
14. Select **Use XSLT to define the handshake**.
15. Click the **create new xsl file** icon that appears at the end of the Xslt field. The Input dialog appears.
16. Use the default value, `invoke.xsl`, as the name of the XSL file and click **OK**.
17. Click **Finish**. The invoke.xsl file appears in the XSL mapper tool, as shown in Figure 5–29.

Figure 5–29 The JDeveloper - invoke.xsl Page



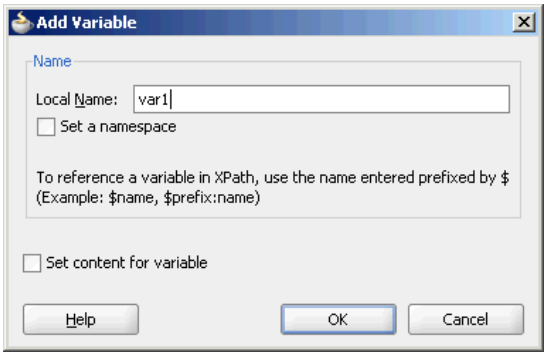
18. Right-click the **HelloWorldProcessResponse** element on the target side. A menu is displayed, as shown in Figure 5–30.

Figure 5–30 The JDeveloper - invoke.xsl Page



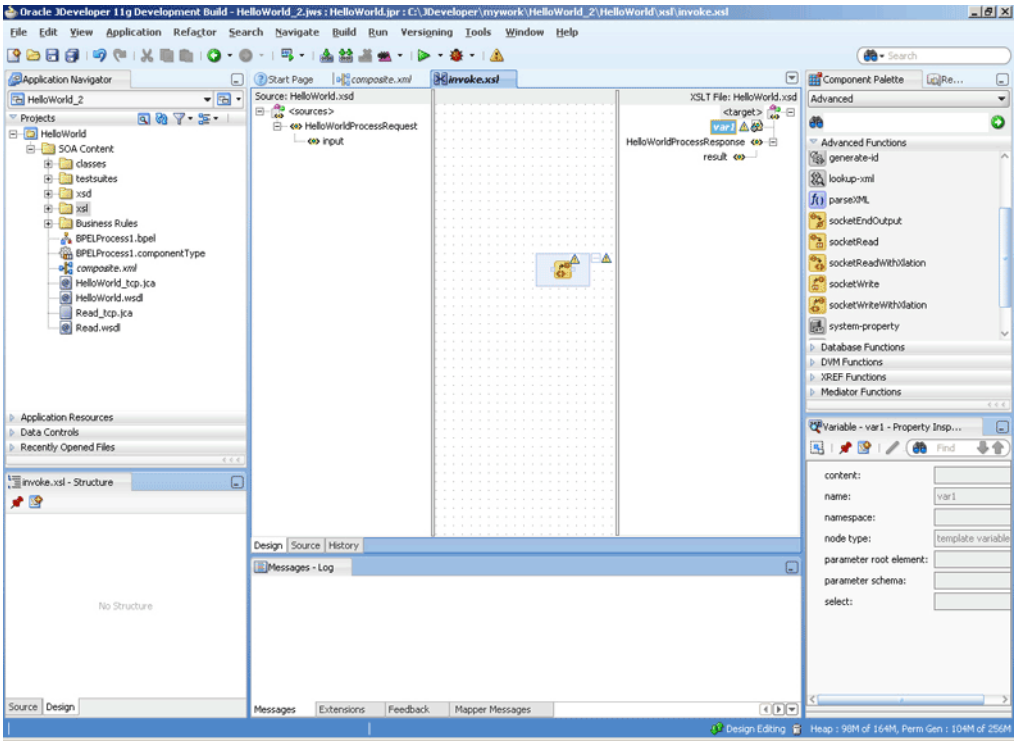
19. Click **Add Variable...**. The **Add Variable** dialog is displayed, as shown in Figure 5–31.

Figure 5–31 The Add Variable Dialog



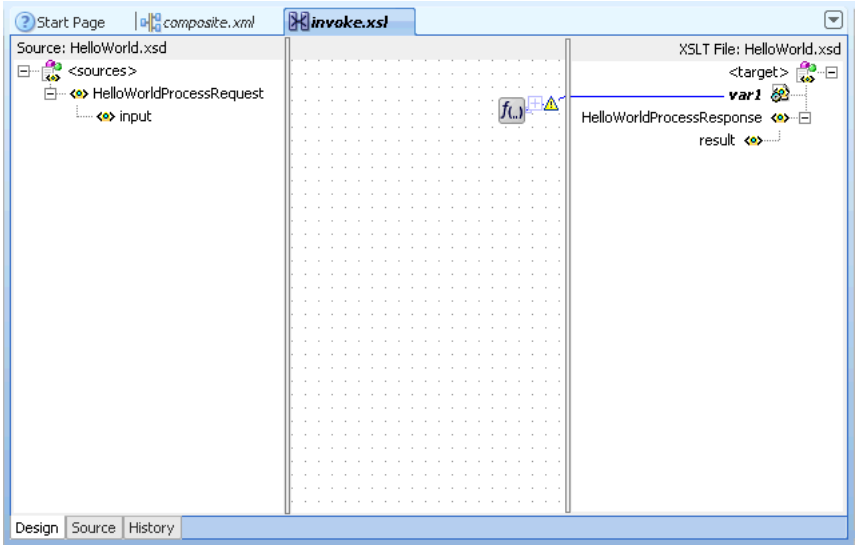
20. Enter `var1` in the **Local Name** field, and click **OK**. The var1 variable is added to the target pane of the XSL mapper tool.
21. From the Component Palette list, select **Advanced**; then, select **Advanced Functions**. A list of advanced functions is displayed.
22. Define the request part of the outbound synchronous request/reply operation, to write the data to the socket server, as follows:
    a. Drag and drop **socketWriteWithXlation** from the Advanced Functions list of the Component Palette to the middle pane, as shown in Figure 5–32.

Figure 5–32 The JDeveloper - invoke.xsl Page



b. Drag the **var1** node to the socketWriteWithXlation function. A link is created, as shown in Figure 5–33.

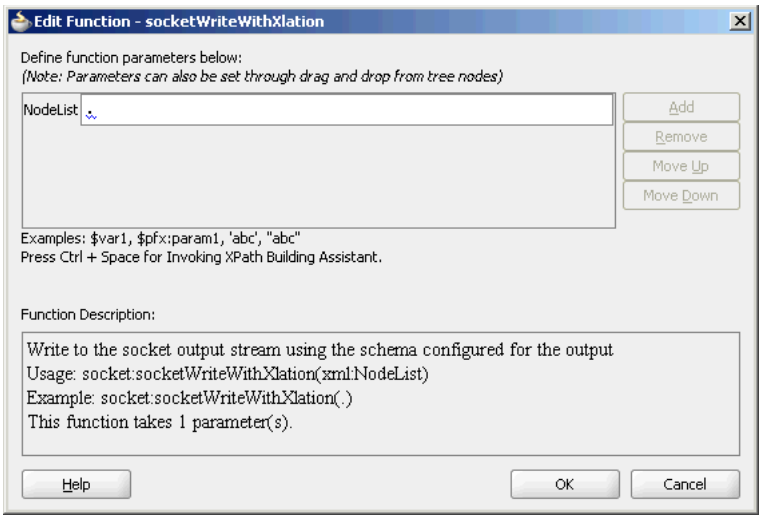
Figure 5–33 The JDeveloper - invoke.xsl Page



c. Double-click the **socketWriteWithXlation** advanced function. The Edit Function - socketWriteWithXlation dialog appears.

d. Enter a dot (.) in the **NodeList** field, as shown in Figure 5–34.

Figure 5–34 The Edit Function - socketWriteWithXlation Dialog



Note: The `socketWriteWithXlation` function writes to the socket output stream using the schema configured for the output.

The dot (.) specified in the NodeList field signifies writing the HelloWorldProcessRequest to the top level node.

---

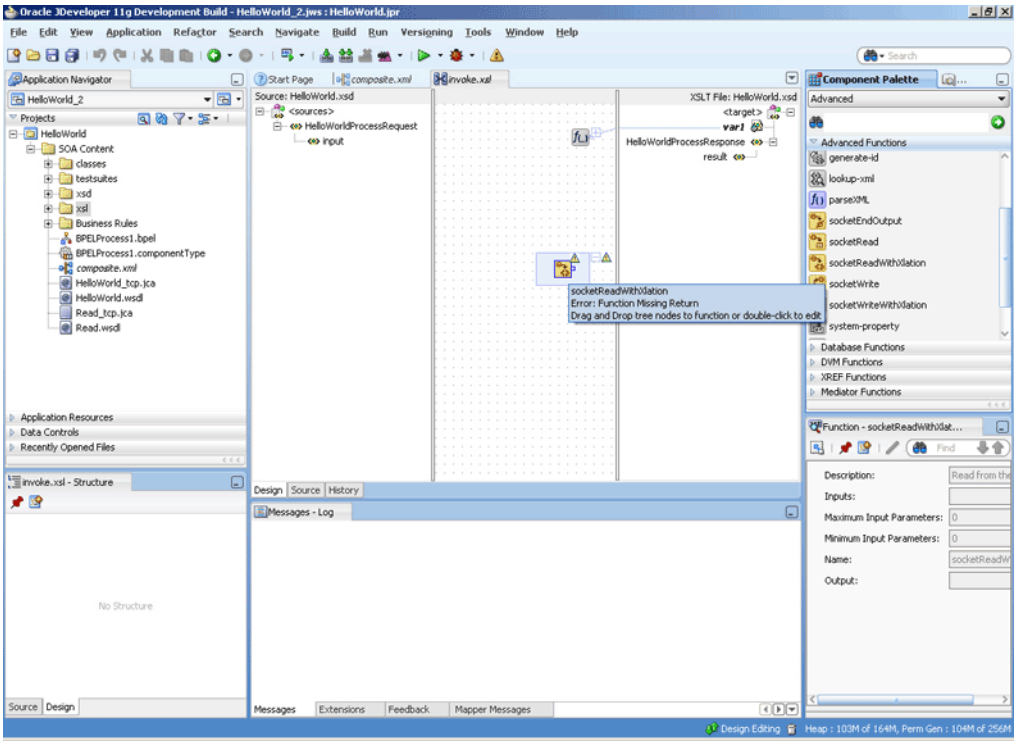
e. Click **OK**. A Warning dialog appears.

f. Click **Yes**. The invoke.xsl page is displayed. The request part of the Synchronous Request/Reply operation is defined.

23. Define the reply part of the outbound synchronous request/reply operation as follows:

a. Drag and drop **socketReadWithXlation** from the Advanced Functions list of the Component Palette to the middle pane, as shown in Figure 5–35.

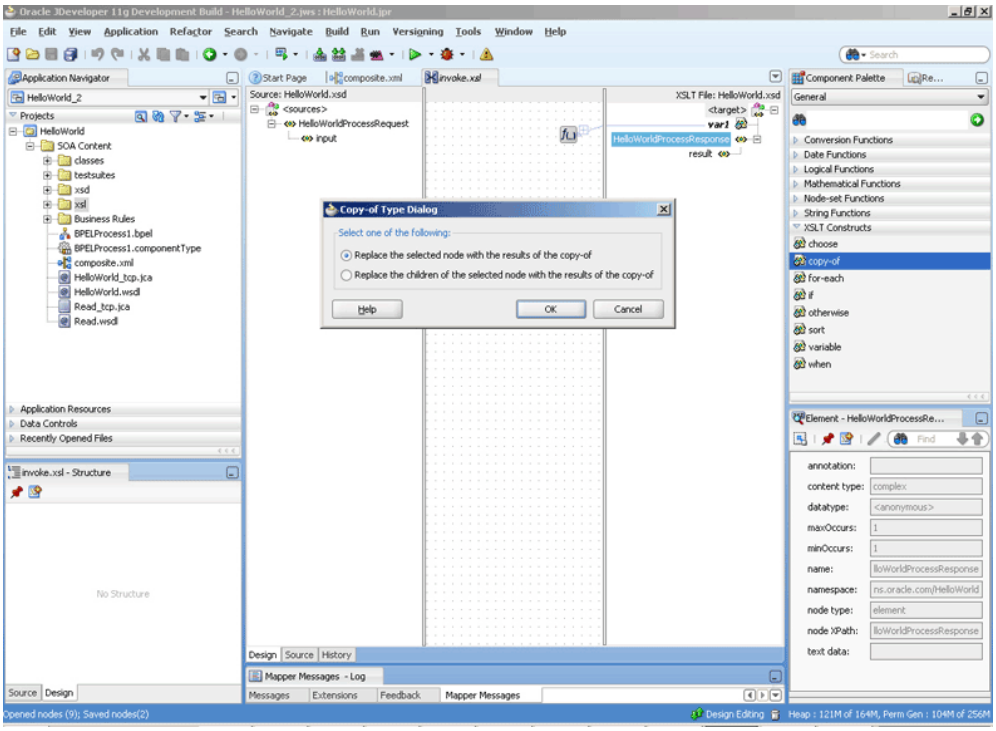
Figure 5–35 The JDeveloper - invoke.xsl Page



b. From the Component Palette list, select **General**; then, select **XSLT Constructs**. A list of XSLT constructs is displayed.

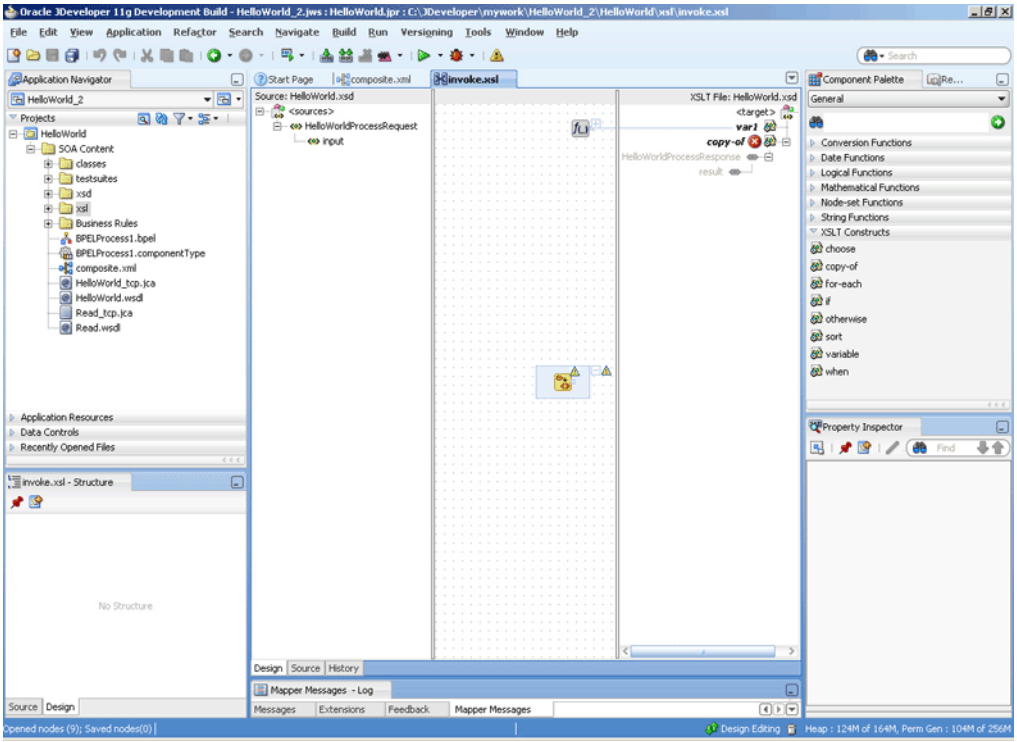
c. Drag **copy-of** from the Component Palette to HelloWorldProcessResponse in the target pane. The Copy-of Type Dialog appears, as shown in Figure 5–36.

Figure 5–36 The JDeveloper - invoke.xsl Page with Copy-of Type Dialog



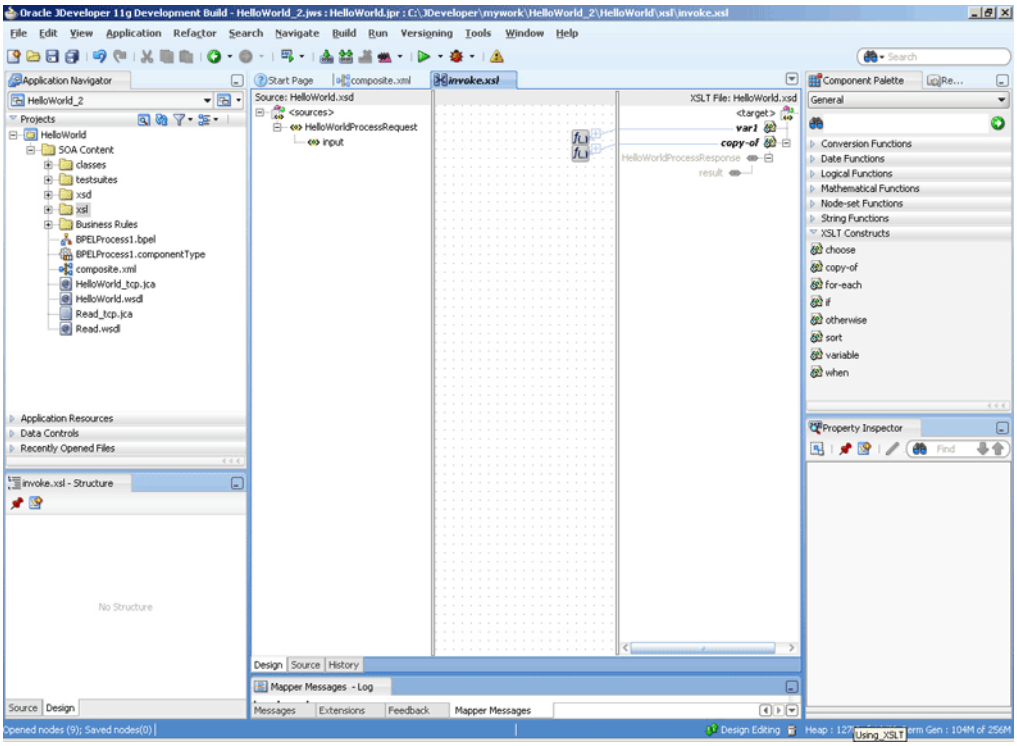
d. Click **OK**. The invoke.xsl (XSL mapper tool) page is displayed with the copy-of XSLT construct added to the target pane, as shown in Figure 5–37.

Figure 5–37 The JDeveloper - invoke.xsl Page



e. Drag the **copy-of** XSLT construct to the socketReadWithXlation function. A link is created, as shown in Figure 5–38.

Figure 5–38 The JDeveloper - invoke.xsl Page



24. Click **File**, **Save All**. The Outbound Synchronous Request/Reply handshake is defined.

5.4.4 Specifying a TCP Port in a Configuration Plan For an Oracle Socket Adapter

To specify a TCP port in a configuration plan for an Oracle Socket Adapter, perform the following steps (where <service-name> is Service name):

1. Add a port property to your <service-name>_tcp.jca file:

```
<property name="Port" value="Port"/>
```

2. Add the following code to your configuration plan XML file:

```
<service_name=<XXXXXX">
 <property>
  <property name="Port">
   <replace>2222</replace>
 </property>
 <binding type'"jca"/>
</service>
```

3. Add the port property to your .xml file under the service element and specify a default value, in this example, 1111

```
<service name="XXXXX" ui:wsdlLocation
  <interface.wsdl.interface="..."/>
  <binding.jca config="XXXX_tcp.jca"/>
    <property name="Port" type=xs:string" many="false"
override="may">1111</property>
```

```
</service>
```

4. Deploy your composite with the configuration plan.

When deployed, the Oracle Socket Adapter listens on port 2222, as provided in the configuration plan.

If you deploy the composite without a configuration plan or if the configuration plan does not override the Port property, then the Oracle Socket Adapter listens on the socket that the composite.xml file's default Port property specifies (in this example, port 1111).

5.5 Oracle Socket Adapter Use Cases

This section includes the following Oracle Socket Adapter use cases:

- Section 5.5.1, "Oracle Socket Adapter Hello World"
- Section 5.5.2, "Flight Information Display System"

5.5.1 Oracle Socket Adapter Hello World

This is a simple HelloWorld use case, which demonstrates the synchronous inbound request/response and synchronous outbound request/response modes of communication using Oracle Socket Adapter. The HelloWorld business process takes an input string from the Oracle Socket Adapter inbound service and publishes the message to the BPEL process. The BPEL process invokes the Oracle Socket Adapter outbound service (a simple HelloWorld Server, which adds a prefix ?Hello? to the input string and returns it) and returns the received string using a synchronous reply.

This use case includes the following sections:

- Section 5.5.1.1, "Prerequisites"
- Section 5.5.1.2, "Designing the SOA Composite"
- Section 5.5.1.3, "Creating the Inbound Oracle Socket Adapter Service"
- Section 5.5.1.4, "Creating the Outbound Oracle Socket Adapter Service"
- Section 5.5.1.5, "Wiring Services and Activities"
- Section 5.5.1.6, "Deploying with JDeveloper"
- Section 5.5.1.7, "Monitoring Using the Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console)"

5.5.1.1 Prerequisites

To perform this use case, you require the following files from the `artifacts.zip` file contained in the `Adapters-101SocketAdapterHelloWorld` sample:

- `artifacts/schemas/HelloWorld.xsd`
- `artifacts/xsl/request.xsl`
- `artifacts/xsl/reply.xsl`
- `artifacts/xsl/invoke.xsl`

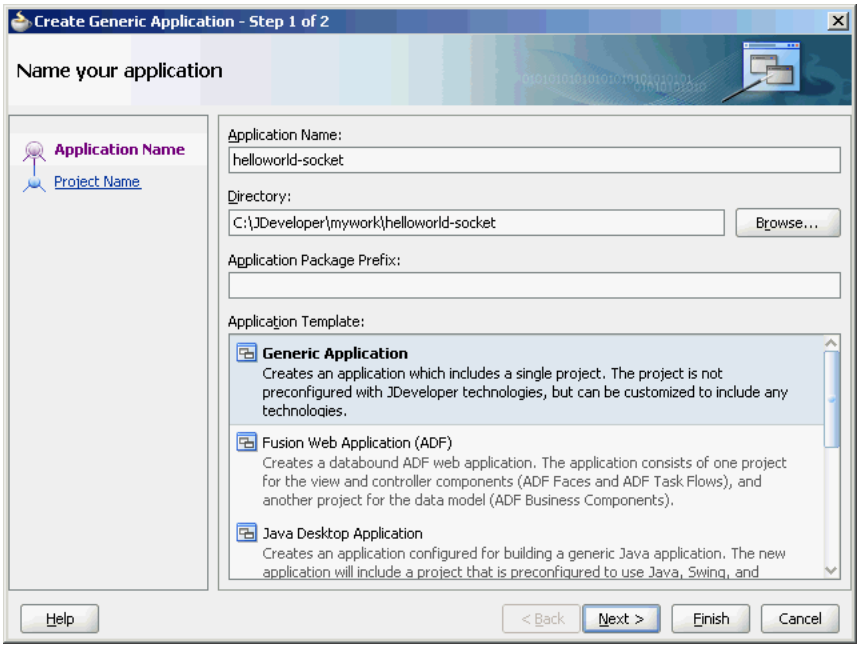
You can access the `Adapters-101SocketAdapterHelloWorld` sample on the Oracle SOA Sample Code site.

5.5.1.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In **Application Navigator** of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `helloworld-socket` in the **Application Name** field, as shown in Figure 5–39, and then click **Next**. The Name your project page is displayed.

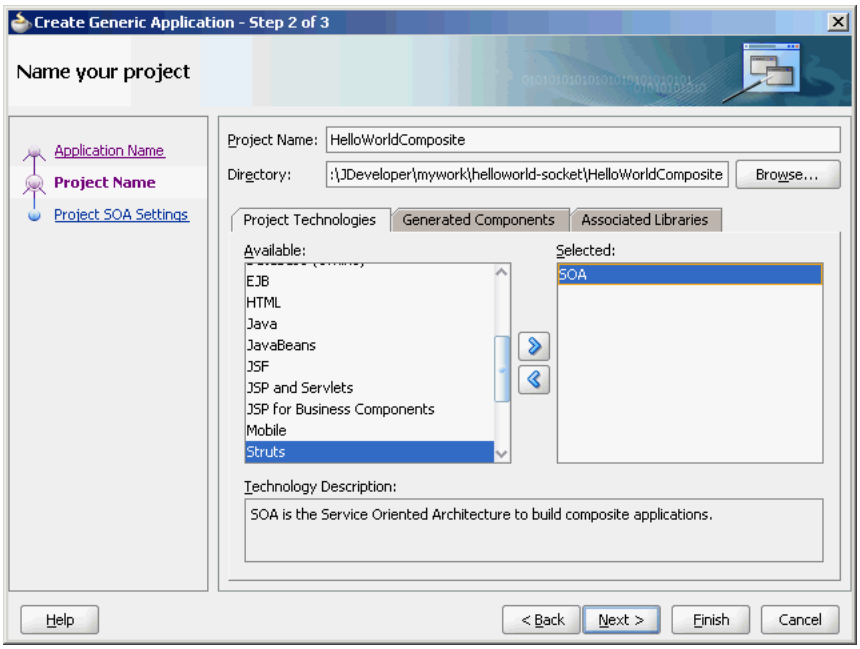
Figure 5–39 The Create SOA Application Dialog



3. Click **OK**. The Name Your Project dialog is displayed.
4. Enter `HelloWorldComposite` in the **Project Name** field, and then select **SOA** under Project Technologies and move it to the **Selected** box by clicking the right-arrow, as shown in Figure 5–40.

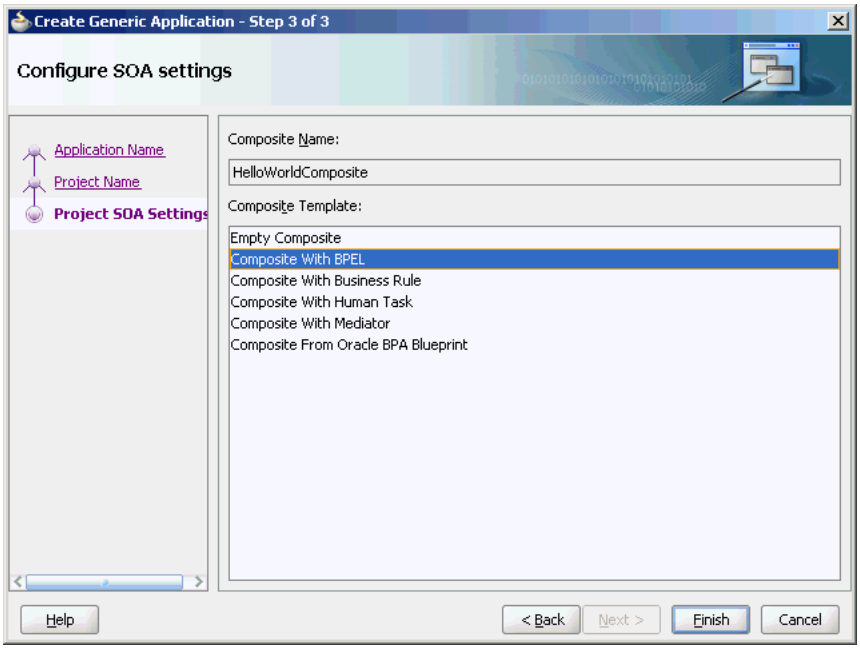
   The HelloWorld application and the HelloWorldComposite project appear in the Application Navigator.

Figure 5–40 The Create Project Dialog



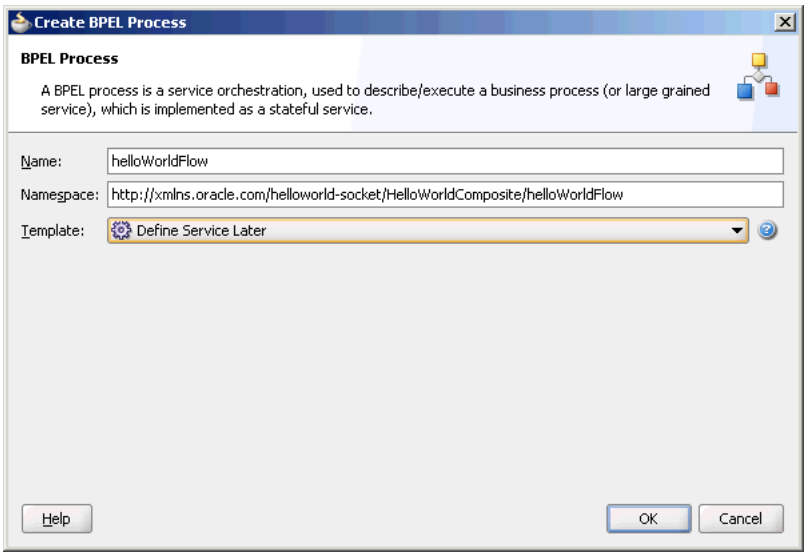
5. Click **Next**. The Configure SOA settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, as shown in Figure 5–41, and click **Finish**. The Create BPEL Process dialog is displayed.

Figure 5–41 The Configure SOA Settings Dialog



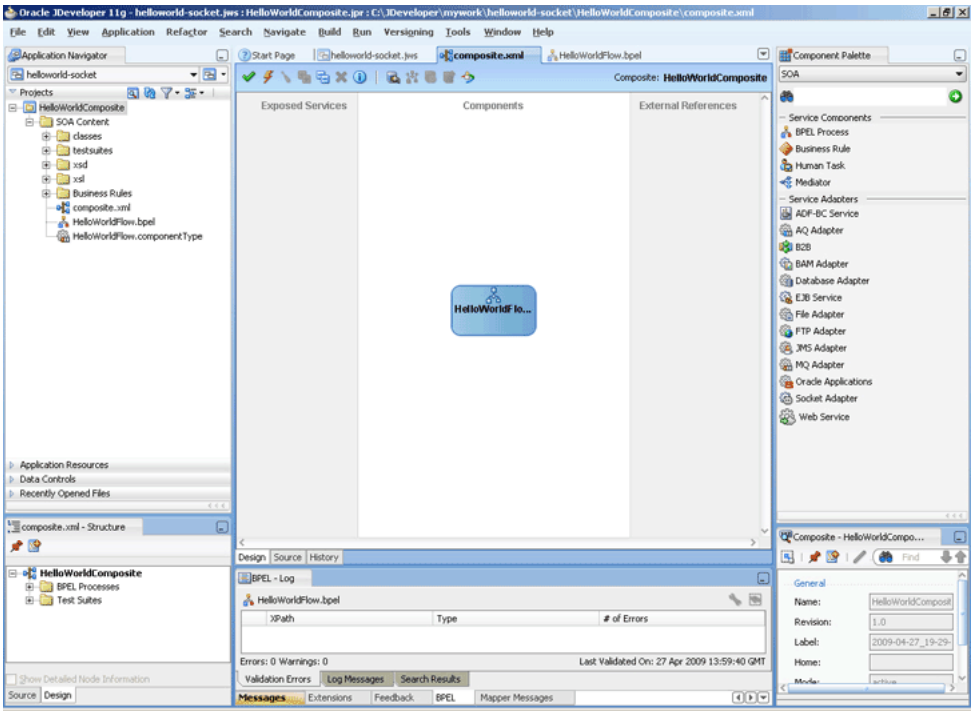
7. Enter `HelloWorldFlow` in the **Name** field and select **Define Service Later** from the Template box, as shown in Figure 5–42.

Figure 5–42 The Create BPEL Process Dialog



8. Click **OK**. The HelloWorld application and the HelloWorldComposite project appear in the design area, as shown in Figure 5–43.

Figure 5–43 The JDeveloper - composite.xml



9. Copy the `HelloWorld.xsd` file to the xsd directory in your project (see Section 5.5.1.1, "Prerequisites" for the location of this file).

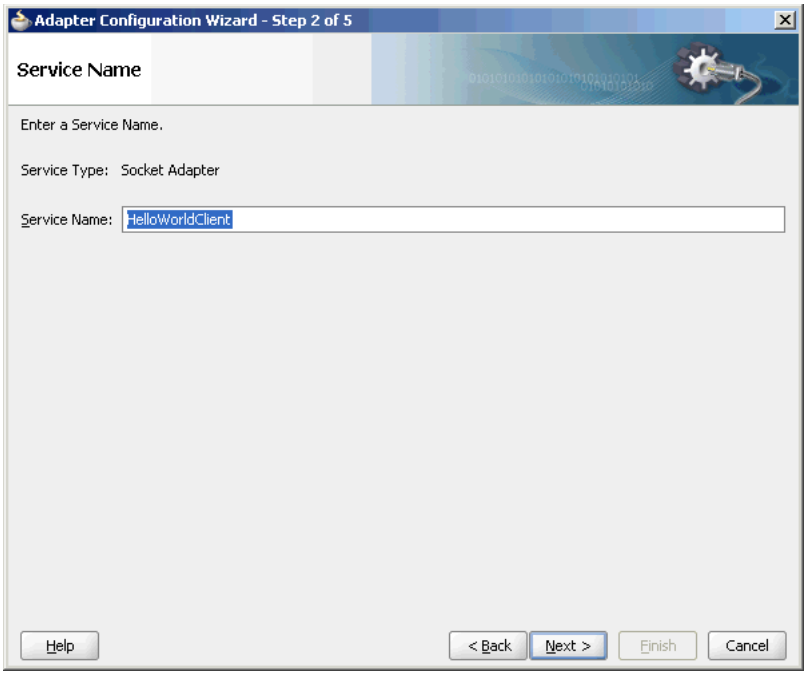
10. Copy the `request.xsl`, `reply.xsl`, and `invoke.xsl` files to the xsl directory in your project (see Section 5.5.1.1, "Prerequisites" for the location of these files).

5.5.1.3 Creating the Inbound Oracle Socket Adapter Service

Perform the following steps to create an inbound Oracle Socket Adapter service:

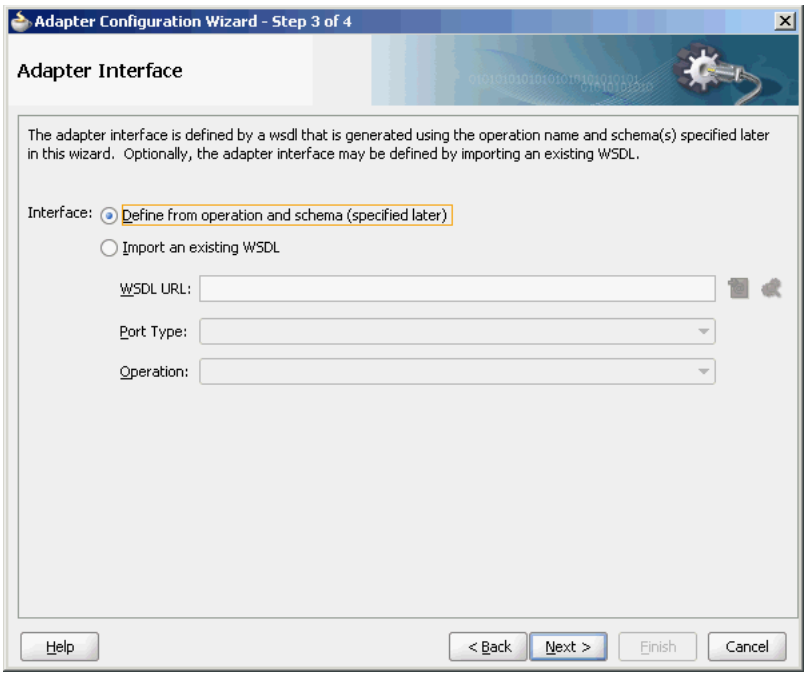
1. Drag and drop Socket Adapter from the Components Palette to the Exposed Services swim lane. The Welcome page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `HelloWorldClient` in the **Service Name** field, as shown in Figure 5–44.

Figure 5–44 The Adapter Configuration Wizard Service Name Page



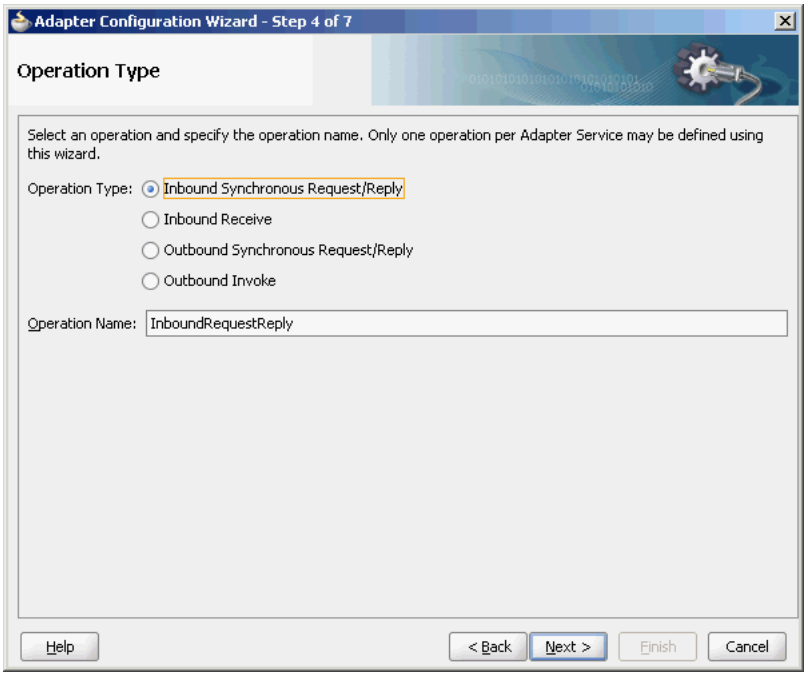
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, as shown in the Figure 5–45 and click **Next**. The Operation page is displayed.

Figure 5–45 The Adapter Configuration Wizard - Adapter Interface Page



6. Select **Inbound Synchronous Request/Reply** as the Operation Type, as shown in Figure 5–46.

Figure 5–46 The Adapter Configuration Wizard Operation Page

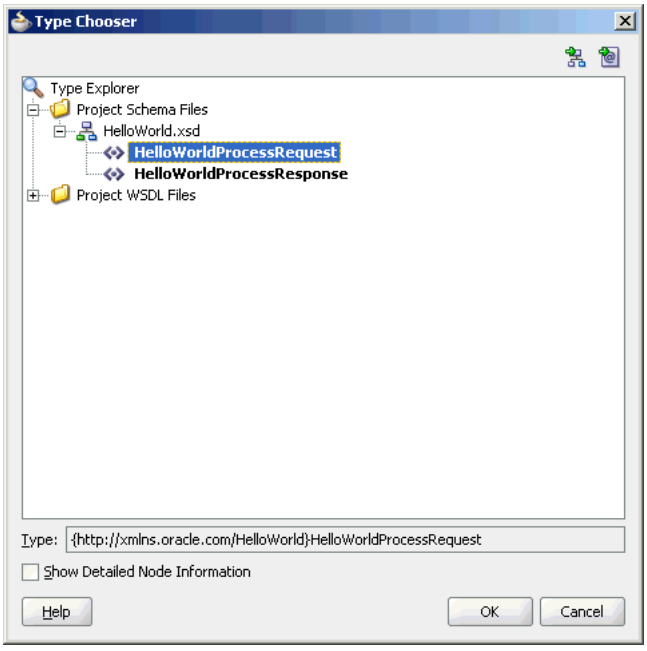


7. Click **Next**. The Socket Connection page is displayed.
8. Enter `eis/socket/InboundSocketAdapter` in the **Socket Connection JNDI Name** field, as shown in Figure 5–47, and click **Next**. The Messages page is displayed.

Figure 5–47 The Adapter Configuration Wizard Socket Connection Page

9. Click **Browse For Schema File** that appears at the end of the URL field in the Request Message Schema box. The Type Chooser dialog is displayed.
10. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessRequest**, as shown in Figure 5–48.

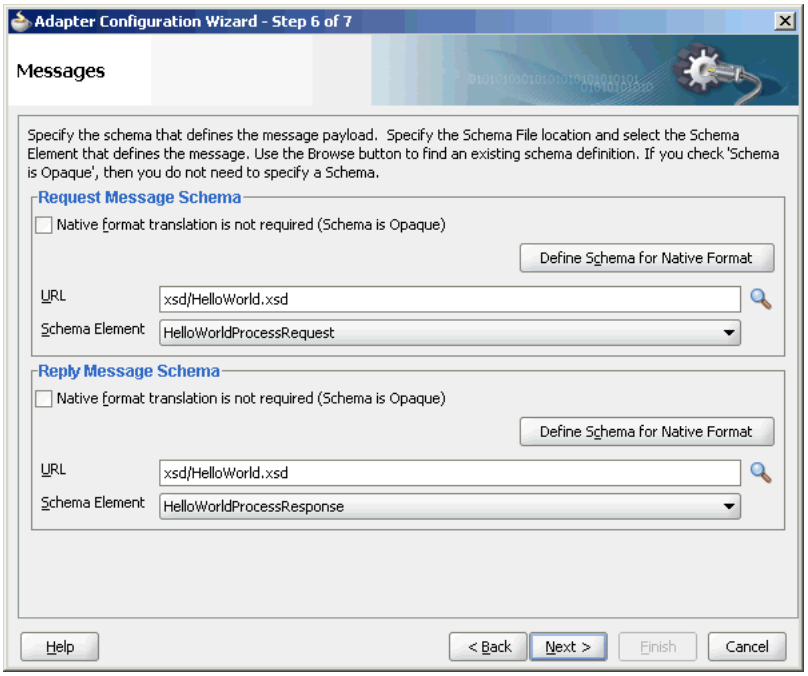
Figure 5–48 The Type Chooser Dialog



11. Click **OK**. The URL field in the Messages page is populated with the `HelloWorld.xsd` file.

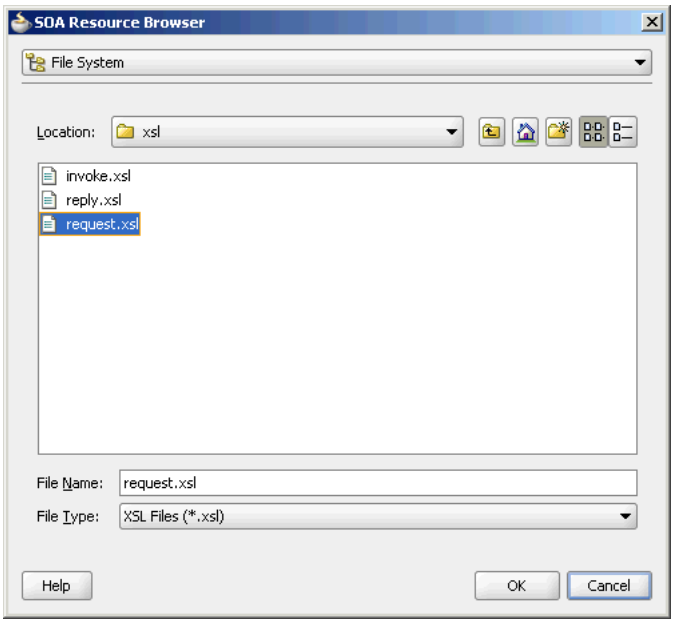
12. Click **Browse For Schema File** that appears at the end of the URL field in the Reply Message Schema box. The Type Chooser dialog is displayed.
13. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessRequest**.
14. Click **OK**. The URL fields in the Messages page are populated with the `HelloWorld.xsd` files, as shown in Figure 5–49.

Figure 5–49 The Adapter Configuration Wizard File Messages Page



15. Click **Next**. The Protocol page is displayed.
16. Select **Use XSLT to define the handshake**.
17. Click **Browse to select the XSL file** that appears at the end of the Xslt field. The SOA Resource Browser dialog is displayed.
18. Select **request.xsl** as the file name, as shown in Figure 5–50, and click **OK**. The Xslt field is populated.

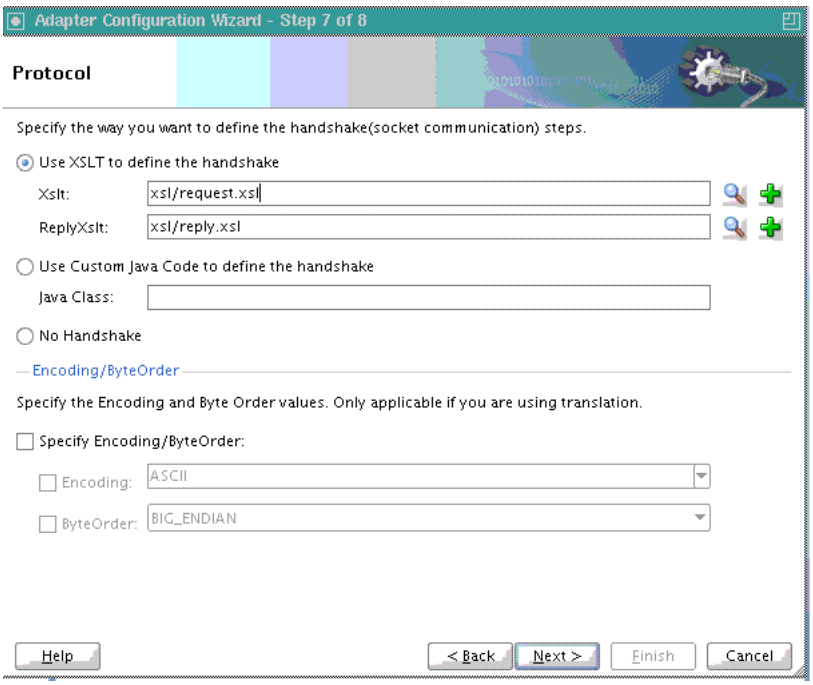
Figure 5–50 The SOA Resource Browser Dialog



19. Click **Browse to select the XSL file** that appears at the end of the ReplyXslt field. The SOA Resource Browser dialog is displayed.

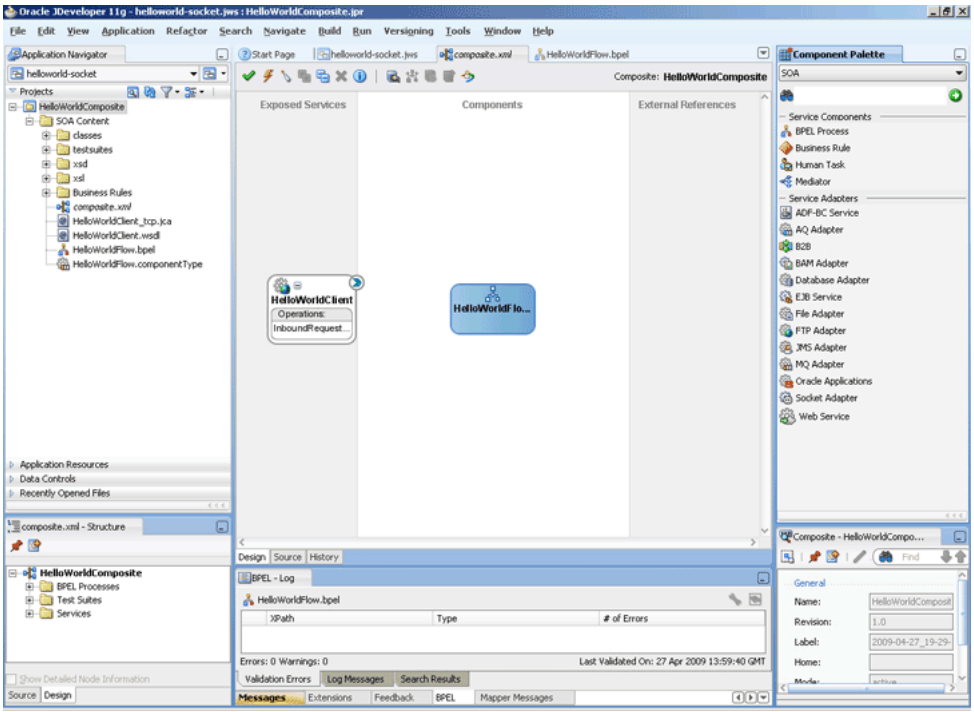
20. Select **reply.xsl** as the file name and click **OK**. The Xslt field is populated, as shown in Figure 5–51.

Figure 5–51 The Adapter Configuration Wizard Protocol Page



21. Click **Finish**. The composite.xml page appears, as shown in Figure 5–52.

Figure 5–52 The JDeveloper - composite.xml Page

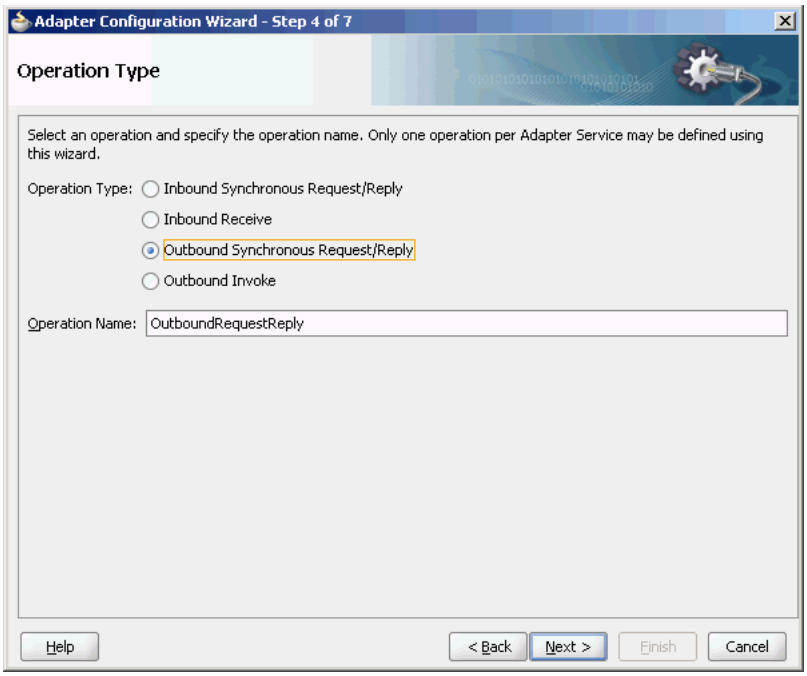


5.5.1.4 Creating the Outbound Oracle Socket Adapter Service

Perform the following steps to create an outbound Oracle Socket Adapter service:

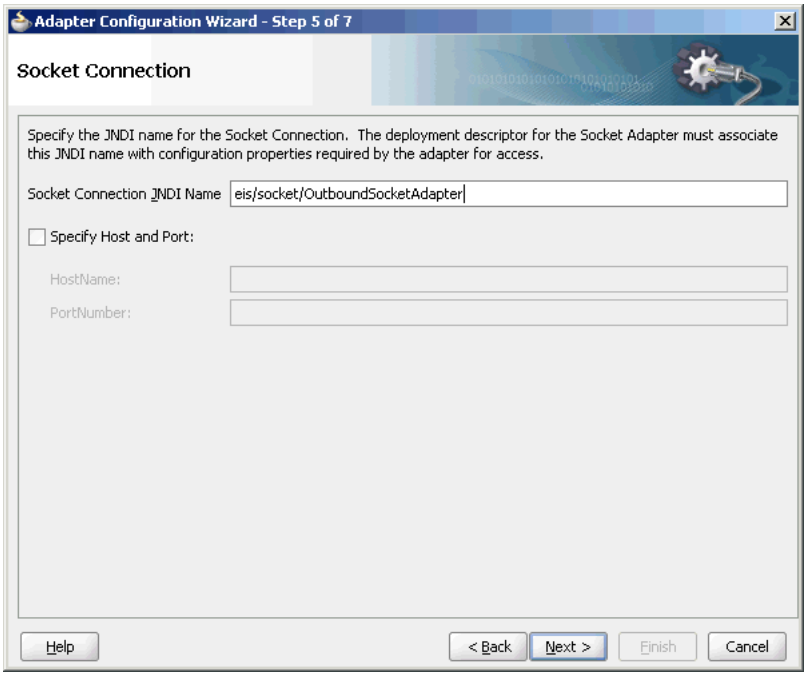
1. Drag and drop **Socket Adapter** from the Component Palette to the External References swim lane. The Welcome page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `HelloWorldServer` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Outbound Synchronous Request/Reply** as the operation type, as shown in Figure 5–53.

Figure 5–53 The Adapter Configuration Wizard Operation Type Page



7. Click **Next**. The Socket Connection page is displayed.
8. Enter `eis/socket/OutboundSocketAdapter` in the **Socket Connection JNDI Name** field, as shown in Figure 5–54, and click **Next**. The Messages page is displayed.

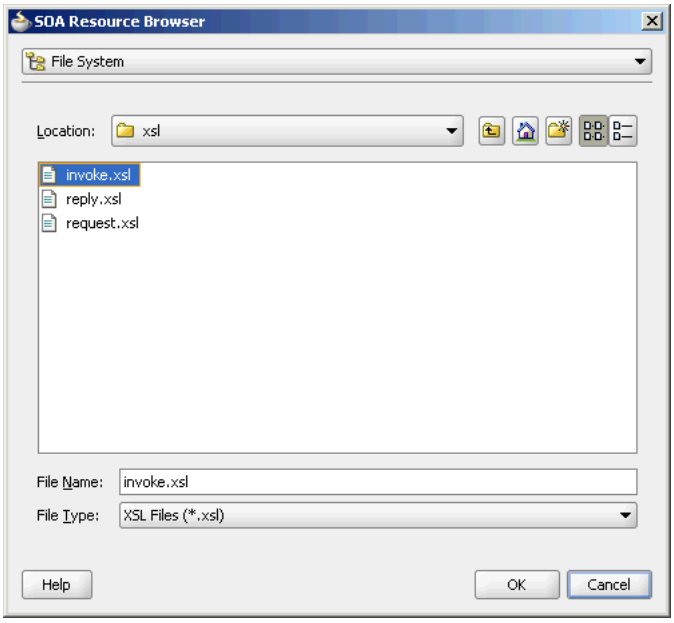
Figure 5–54 The Adapter Configuration Wizard Socket Connection Page



9. Click **Browse For Schema File** that appears at the end of the URL field in the Request Message Schema box. The Type Chooser dialog is displayed.

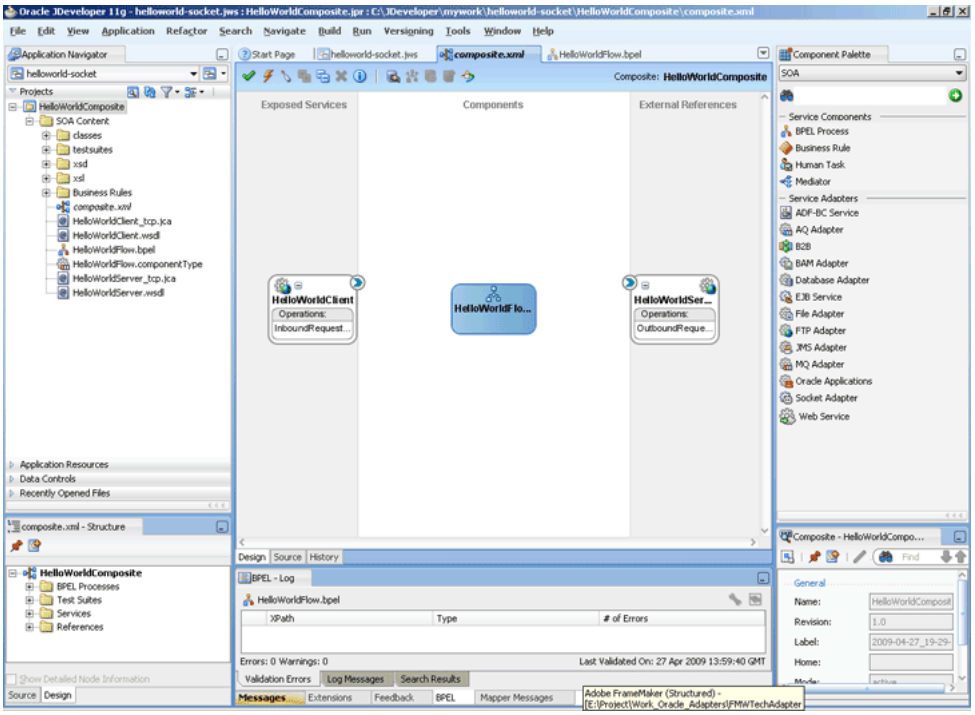
10. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessRequest**, as shown in Figure 5–48.
11. Click **OK**. The URL field in the Messages page is populated with the `HelloWorld.xsd` file.
12. Click **Browse For Schema File** that appears at the end of the URL field in the Reply Message Schema box. The Type Chooser dialog is displayed.
13. Click **Project Schema Files**, **HelloWorld.xsd**, and **HelloWorldProcessResponse**.
14. Click **OK**. The URL fields in the Messages page are populated with the `HelloWorld.xsd` files, as shown in Figure 5–49.
15. Click **Next**. The Protocol page is displayed.
16. Select **Use XSLT to define the handshake**.
17. Click **Browse to select the XSL file** that appears at the end of the Xslt field. The SOA Resource Browser dialog is displayed.
18. Select **invoke.xsl** as the file name, as shown in Figure 5–55, and click **OK**. The Xslt field is populated.

Figure 5–55 The SOA Resource Browser Dialog



19. Click **Finish**. The Composite.xml page appears, as shown in Figure 5–56.

Figure 5–56 The JDeveloper - composite.xml Page



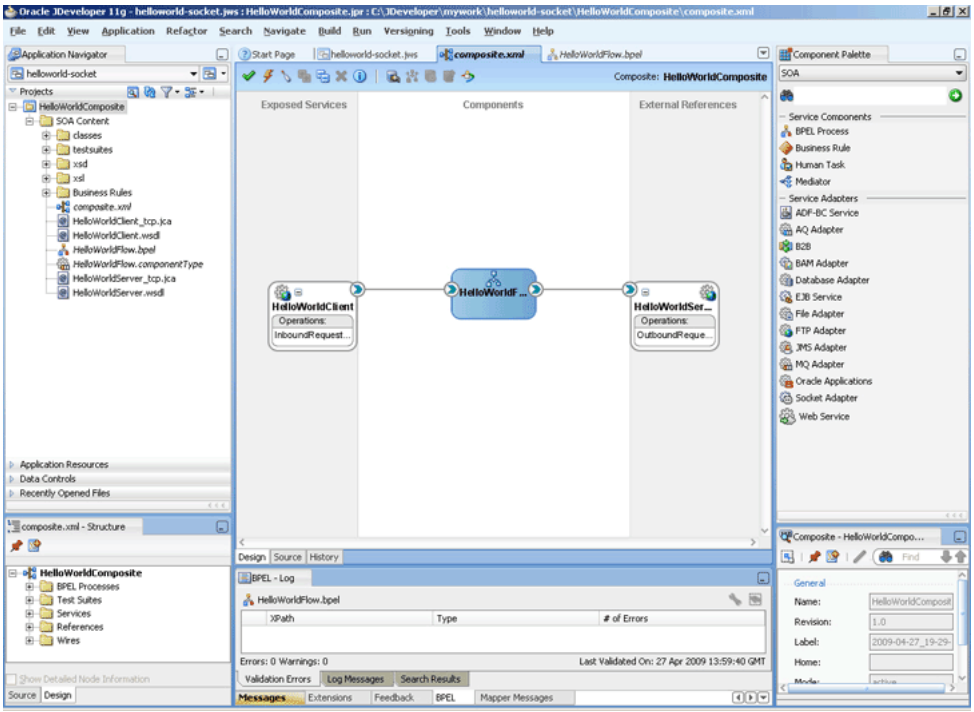
5.5.1.5 Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the HelloWorldClient in the Exposed Services area to the drop zone that appears as a green triangle in the HelloWorldFlow BPEL process in the Components area.
2. Drag the small triangle in the HelloWorldFlow BPEL process in the Components area to the drop zone that appears as a green triangle in the HelloWorldServer in the External References area.

   The JDeveloper composite.xml appears, as shown in Figure 5–57.

Figure 5–57 The JDeveloper - composite.xml

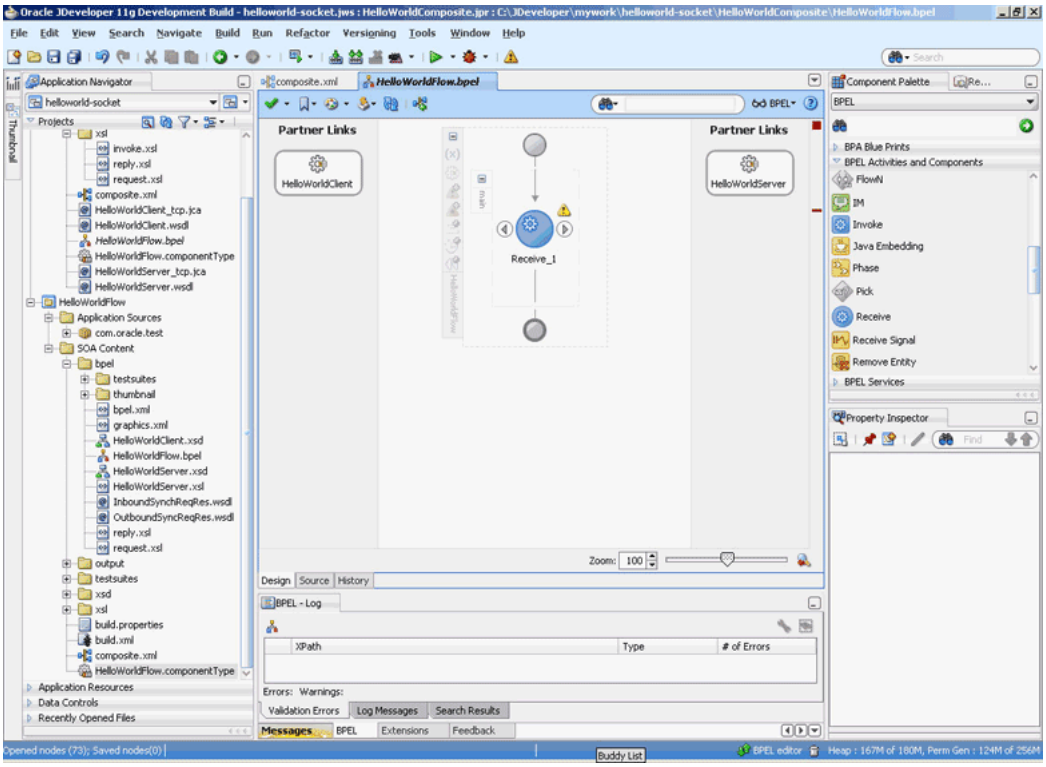


3. Click **File**, **Save All**.

Add a Receive Activity

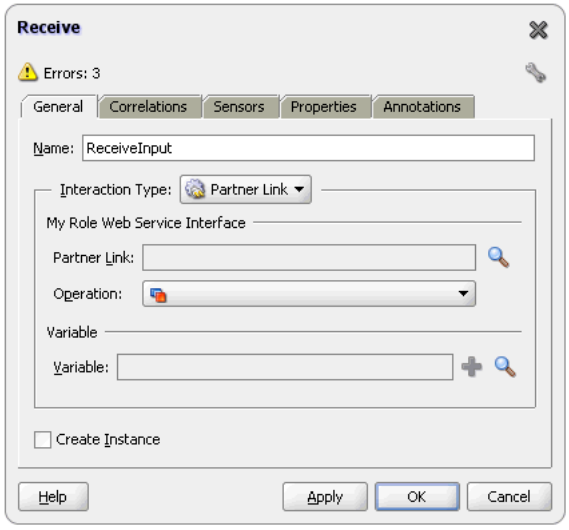
1. Double-click **HelloWorldFlow**. The BPELHelloWorld.bpel page is displayed.
2. Drag and drop a **Receive** activity from the Component Palette to the design area, as shown in Figure 5–58.

Figure 5–58 The JDeveloper - HelloWorldFlow.bpel



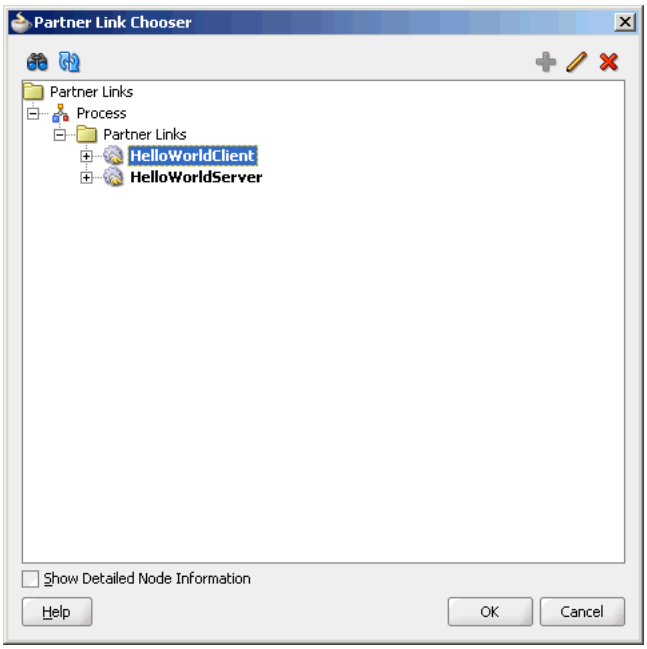
3. Double-click the **Receive** activity. The Receive dialog is displayed.
4. Enter `ReceiveInput` in the **Name** field, as shown in Figure 5–59.

Figure 5–59 The Receive Dialog



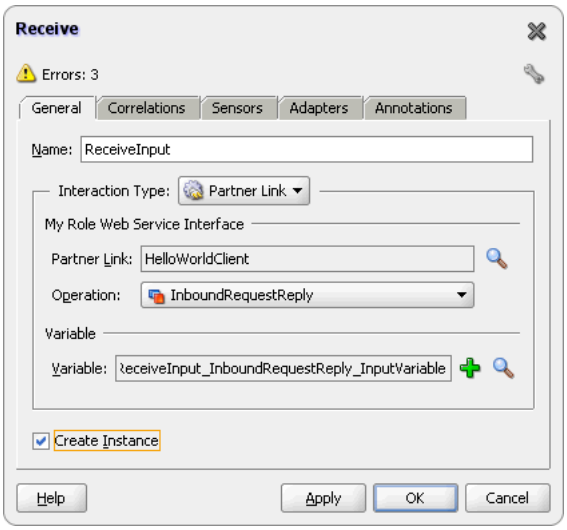
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
6. Select **HelloWorldClient**, as shown in Figure 5–60, and click **OK**.

Figure 5–60 The Partner Link Chooser Dialog



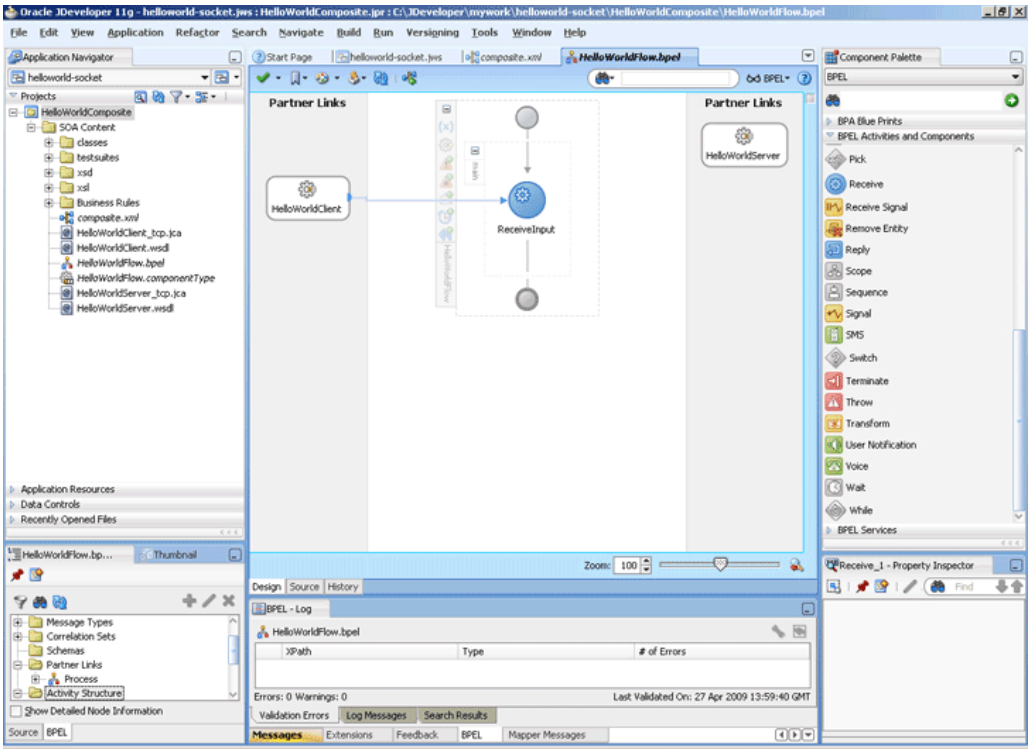
7. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog, as shown in Figure 5–61. The Create Variable dialog is displayed.

Figure 5–61 The Receive Dialog



8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper HelloWorldFlow.bpel page appears, as shown in Figure 5–62.

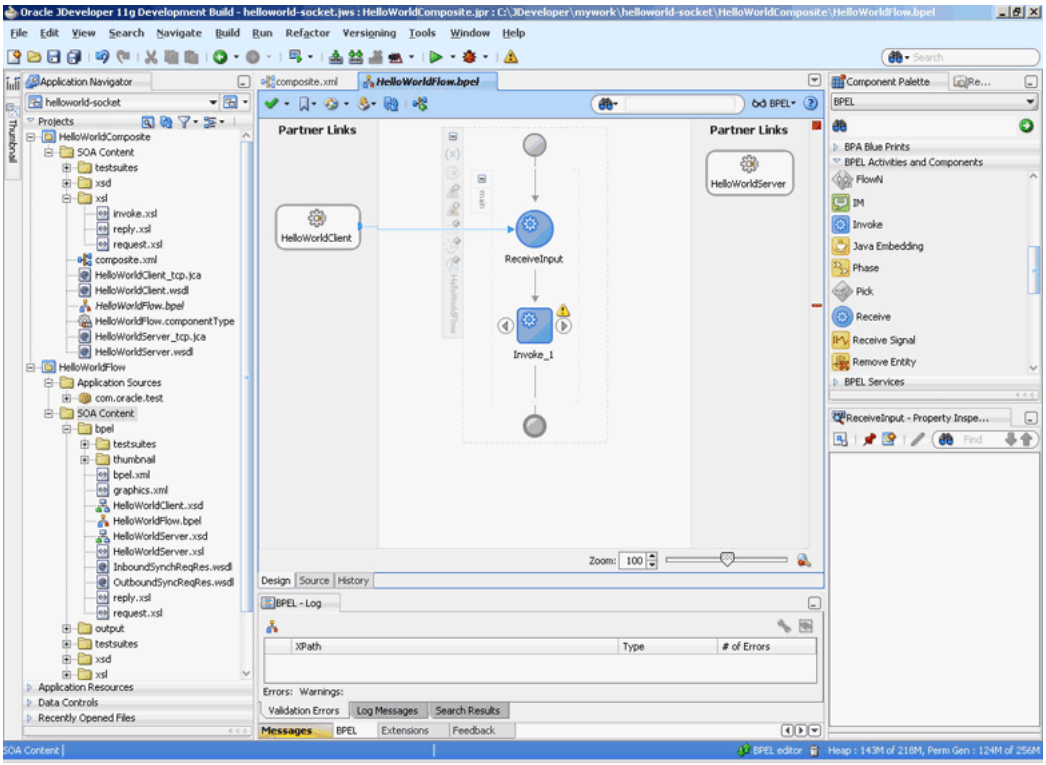
Figure 5–62 The JDeveloper - HelloWorldFlow.bpel



Add an Invoke Activity

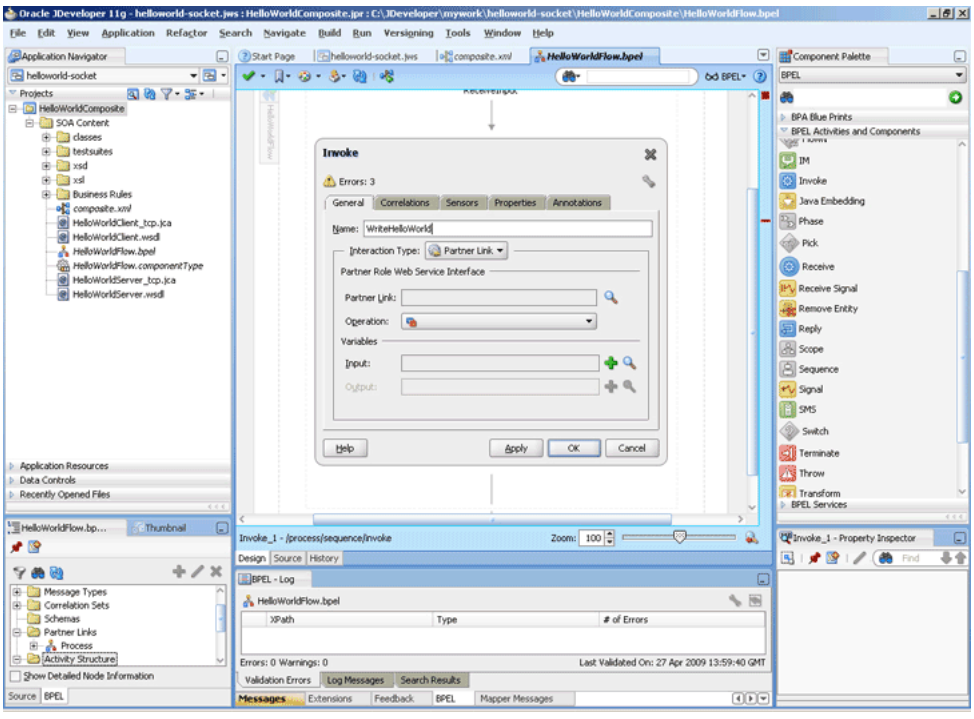
1. Drag and drop an **Invoke** activity after the ReceiveInput activity from the Component Palette to the design area, as shown in Figure 5–63.

Figure 5–63 The JDeveloper - HelloWorldFlow.bpel



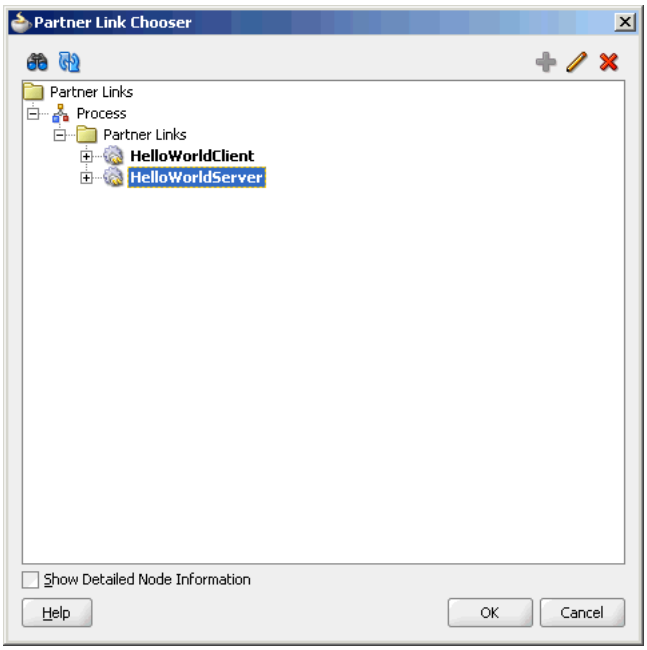
2. Double-click the **Invoke** activity. The Invoke dialog is displayed.
3. Enter `WriteHelloWorld` in the **Name** field, as shown in Figure 5–64.

Figure 5–64 The JDeveloper - HelloWorldFlow.bpel



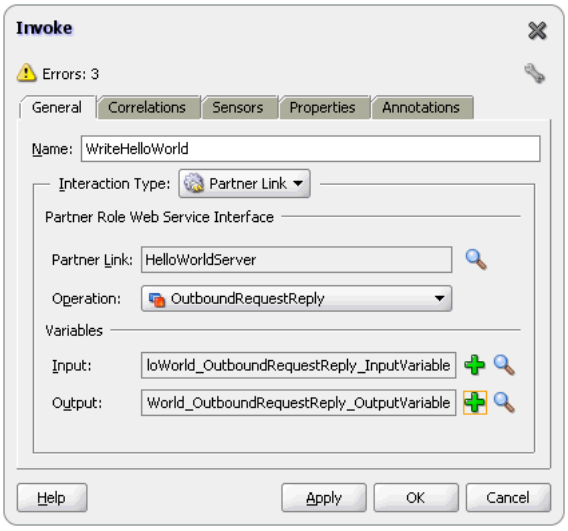
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **HelloWorldServer**, as shown in Figure 5–65, and click **OK**.

Figure 5–65 The Partner Link Chooser Dialog



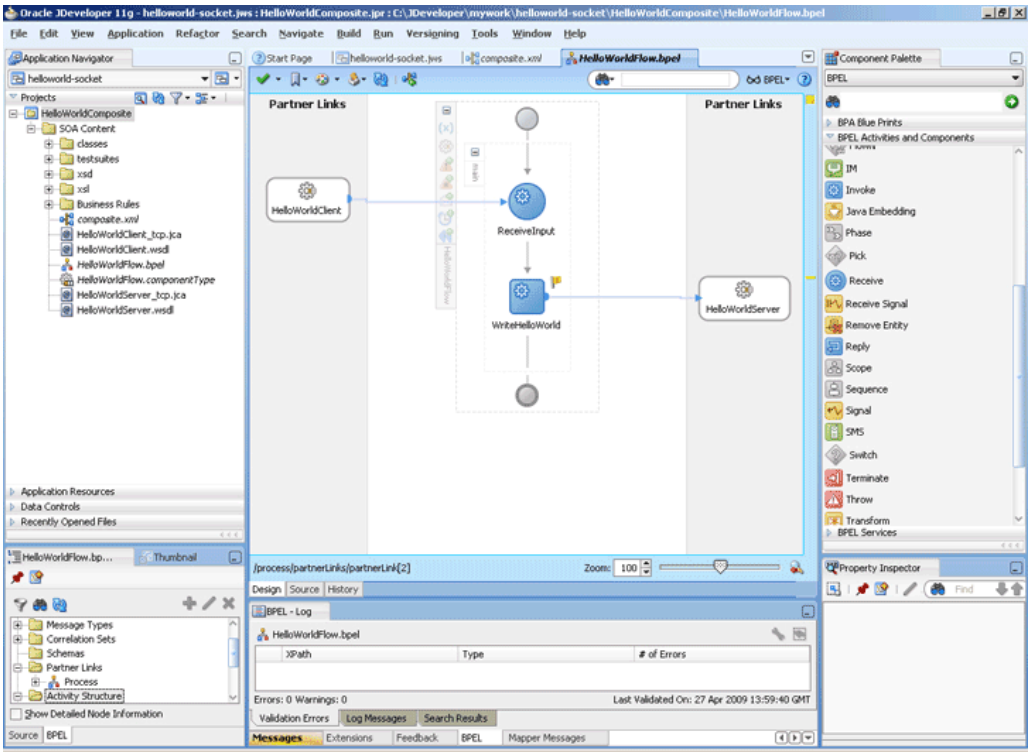
6. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
8. Repeat the same for selecting the output variable. The Invoke dialog is displayed, as shown in Figure 5–66.

Figure 5–66 The Invoke Dialog



9. Click **OK**. The JDeveloper HelloWorldFlow.bpel page appears, as shown in Figure 5–67.

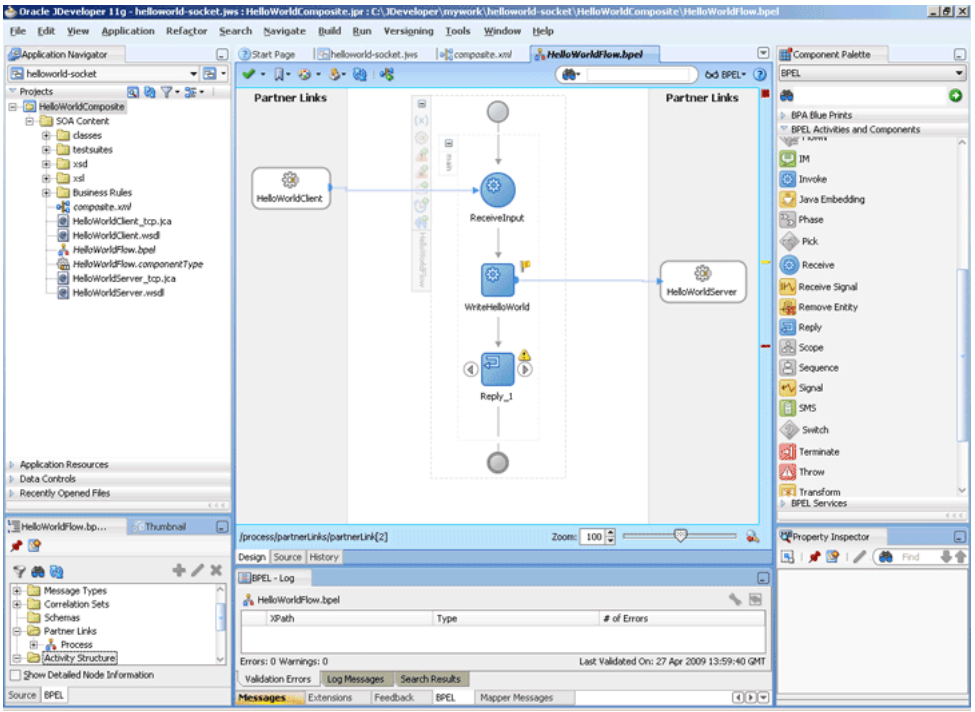
Figure 5–67 The JDeveloper - HelloWorldFlow.bpel



Add a Reply Activity

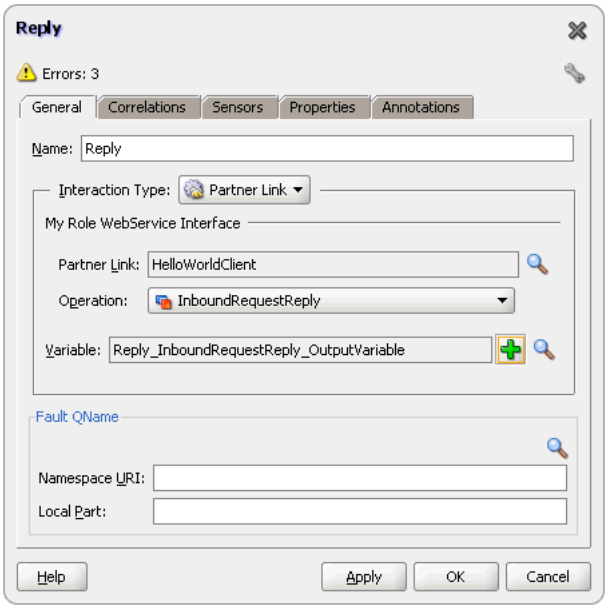
1. Drag and drop a **Reply** activity from the Component Palette to the design area, as shown in Figure 5–68.

Figure 5–68 The JDeveloper - HelloWorldFlow.bpel



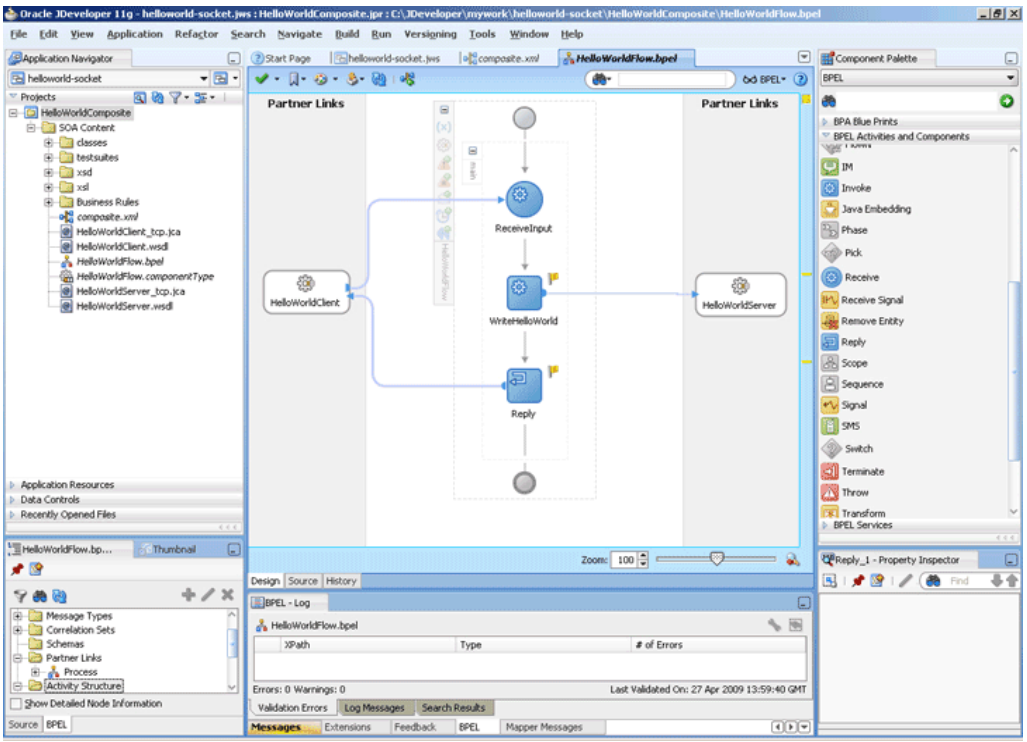
2. Double-click the **Reply** activity. The Reply dialog is displayed.
3. Enter `Reply` in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **HelloWorldClient**, as shown in Figure 5–60, and click **OK**.
6. Click the **Auto-Create Variable** icon to the right of the Variable field in the Reply dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name, as shown in Figure 5–69.

Figure 5–69 The Reply Dialog



8. Click **OK**. The JDeveloper HelloWorldFlow.bpel page appears, as shown in Figure 5–70.

Figure 5–70 The JDeveloper - HelloWorldFlow.bpel

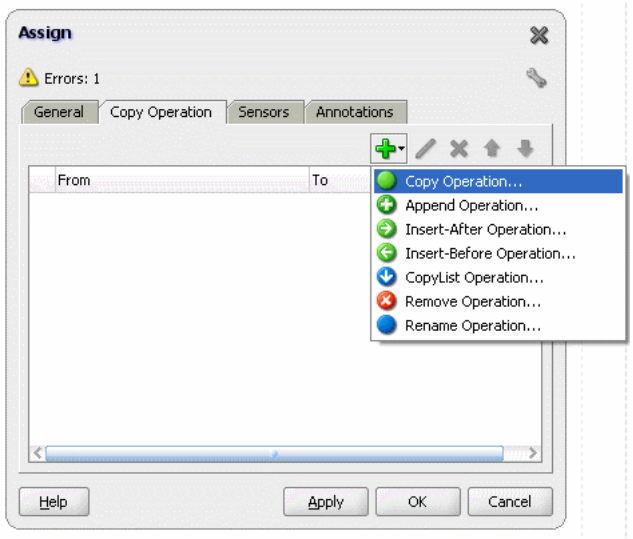


Add Assign Activities

1. Drag and drop an **Assign** activity from the Component Palette in between the Receive and Invoke activities in the design area.
2. Double-click the **Assign** activity. The Assign dialog is displayed.

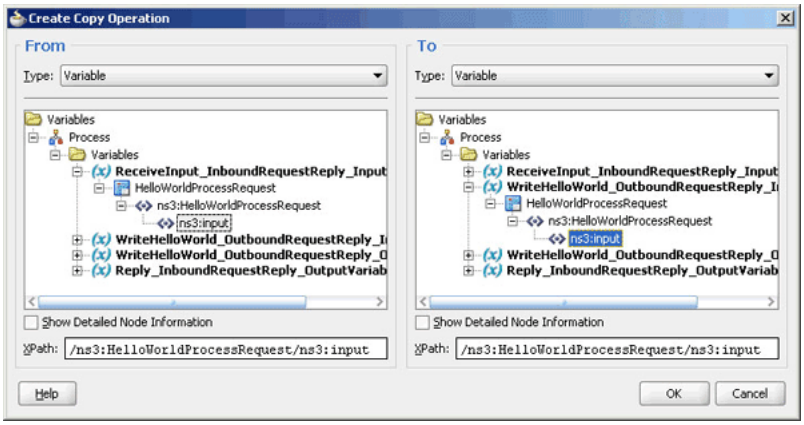
3. Click the **Copy Operation** tab. The Assign dialog is displayed, as shown in Figure 5–71.

Figure 5–71 The Assign Dialog - Copy Operation Tab



4. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
5. In the left pane, under the ReceiveInput_InboundRequestReply_InputVariable variable select, **ns3:input**.
6. In the right pane, under the WriteHelloWorld_OutboundRequestReply_ InputVariable variable select, **ns3:input**, as shown in Figure 5–72.

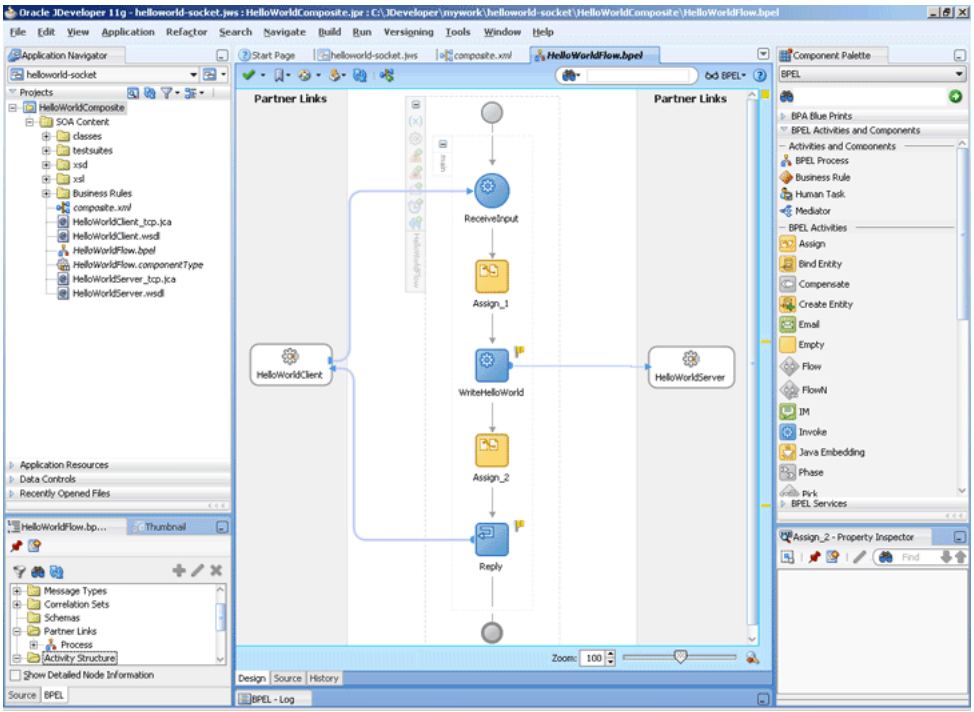
Figure 5–72 The Create Copy Operation Dialog



7. Click **OK**. The Assign dialog is displayed.
8. Click **OK**. The JDeveloper HelloWorldFlow.bpel page is displayed.
9. Add another Assign activity in between the Invoke and the Reply activities.
10. Double-click the assign activity.
11. Click the **Copy Operation** tab, and select **Copy Operation**.
12. In the left pane, select **ns3:result** under WriteHelloWorld_ OutboundRequestReply_OutputVariable.

13. In the right pane, select **ns3:result** under Reply_InboundRequestReply_ OutputVariable and click **OK**.
14. Click **OK**, the JDeveloper HelloWorldFlow.bpel page is displayed, as shown in Figure 5–73.

Figure 5–73 The JDeveloper - HelloWorldFlow.bpel



15. Click **File**, **Save All**.

5.5.1.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application. For more information, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

5.5.1.7 Monitoring Using the Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console)

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://`*`servername`*`:`*`portnumber`*`/em`. The composite you deployed appears in the application navigator.
2. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the Recent Instances area.
3. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.

4. Click the Instance ID that you noted in Step 2. The Flow Trace page is displayed.
5. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
6. Expand a payload node to view payload details.
7. Click the **Flow** tab to view the process flow. Additionally, click an activity (such as invoke, receive) to view the details of an activity.

5.5.2 Flight Information Display System

The flight information display system use case demonstrates the various modes of defining handshakes by using Oracle Socket Adapter.A flight information display server (FIDS) is started by an FIDS client requesting information on flight status for flights originating from a particular source, JFK, or SFO. The FIDS, in turn, invokes flight data requests for three airlines, Airline1, Airline 2, and Airline 3. The FIDS then collates the information received and replies to the FIDS client by using the HTTP protocol.

This use case includes the following sections:

- Section 5.5.2.1, "Prerequisites"
- Section 5.5.2.2, "Designing the SOA Composite"
- Section 5.5.2.3, "Creating the Inbound Oracle Socket Adapter Service"
- Section 5.5.2.4, "Creating Outbound Oracle Socket Adapter Services"
- Section 5.5.2.5, "Wiring Services and Activities"
- Section 5.5.2.6, "Deploying with JDeveloper"
- Section 5.5.2.7, "Monitoring Using the Fusion Middleware Control Console"

5.5.2.1 Prerequisites

To perform this use case, you require the following files from the `artifacts.zip` file contained in the `Adapters-102SocketAdapterFlightInformationDisplaySystem` sample:

- `artifacts/schemas/Airline1.xsd`
- `artifacts/schemas/Airline2.xsd`
- `artifacts/schemas/Airline3.xsd`
- `artifacts/schemas/FIDS.xsd`
- `artifacts/xsl/request.xsl`
- `artifacts/xsl/reply.xsl`
- `artifacts/xsl/invoke.xsl`

To obtain the `Adapters-102SocketAdapterFlightInformationDisplaySystem` sample, access the Oracle SOA Sample Code site.

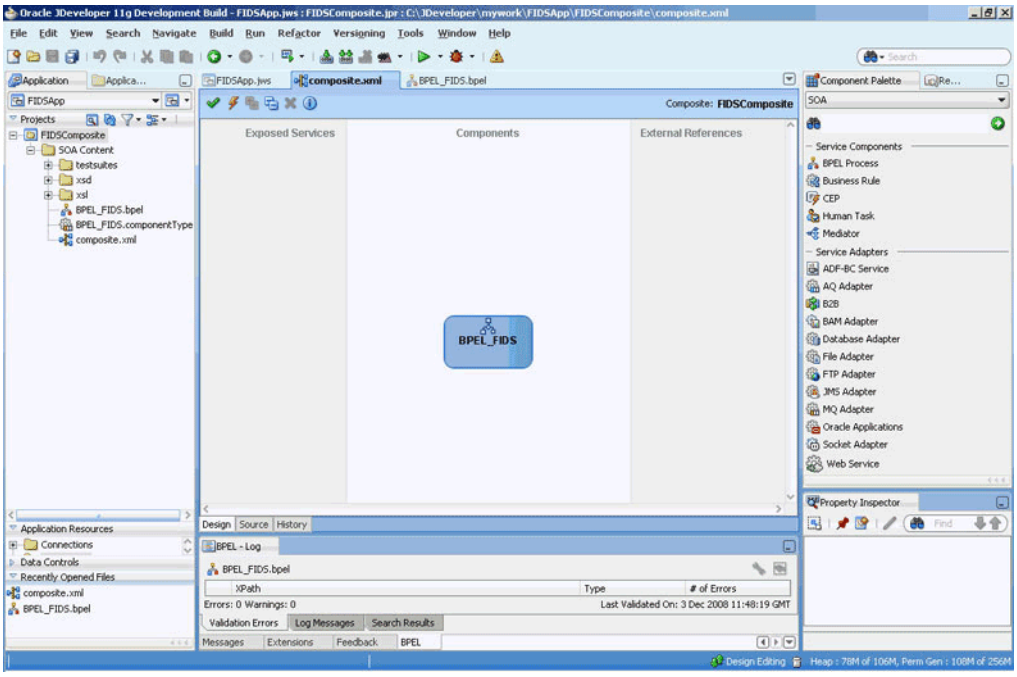
5.5.2.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following steps:

1. In the Application Navigator of JDeveloper, click **New Application**. The Create Generic Application - Name your application page is displayed.
2. Enter `FIDSApp` in the **Application Name** field, and then click **Next**. The Name your project page is displayed.
3. Click **OK**. The Name Your Project dialog is displayed.
4. Enter `FIDSComposite` in the **Project Name** field, and then select **SOA** under Project Technologies and move it to the **Selected** box by clicking the right-arrow.

   The FIDSApp application and the FIDSComposite project appear in the Application Navigator.
5. Click **Next**. The Configure SOA Settings dialog appears.
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process dialog is displayed.
7. Enter `BPEL_FIDS` in the **Name** field and select **Define Service Later** from the Template box.
8. Click **OK**. The FIDSApp application and the FIDSComposite project appear in the design area, as shown in Figure 5–74.

Figure 5–74 The JDeveloper - composite.xml



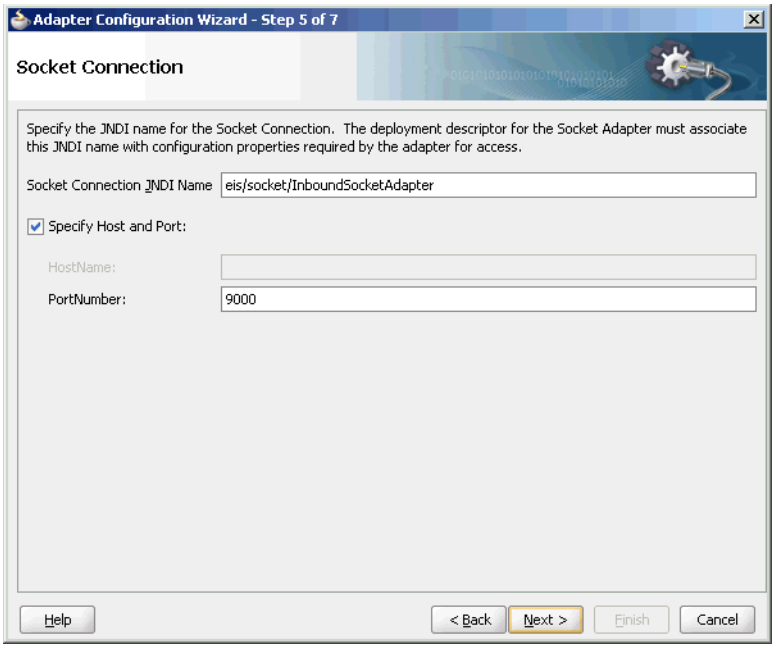
9. Copy the `Airline1.xsd`, `Airline2.xsd`, `Airline3.xsd`, and `FIDS.xsd` files to `FIDSComposite\xsd` under the project `FIDSComposite` (see Section 5.5.2.1, "Prerequisites" for the location of these files).
10. Copy `invoke.xsl`, `request.xsl`, and `reply.xsl` to `FIDSComposite\xsl` under the project `FIDSComposite` (see Section 5.5.2.1, "Prerequisites" for the location of these files).

5.5.2.3 Creating the Inbound Oracle Socket Adapter Service

Perform the following steps to create an inbound Oracle Socket Adapter service that would be used to expose the FIDSApp application:

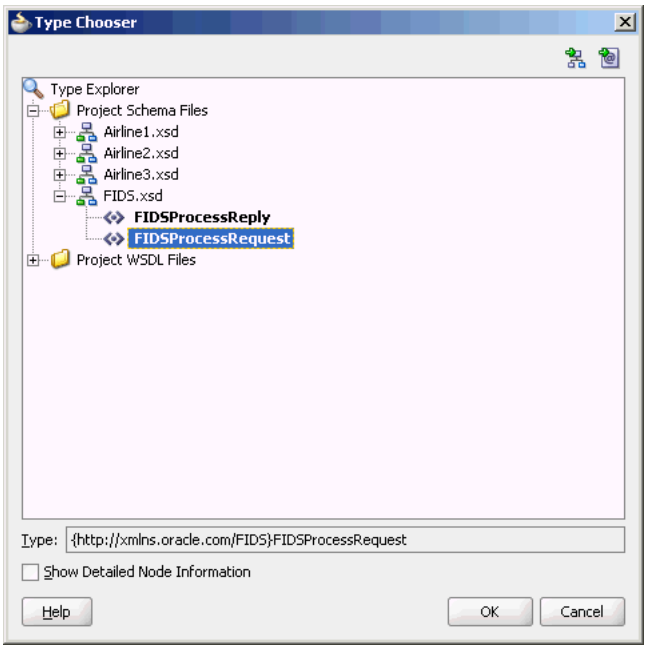
1. Drag and drop **Socket Adapter** from the Components Palette to the Exposed Services swim lane. The Welcome page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `FIDS` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Inbound Synchronous Request/Reply** as the operation type.
7. Click **Next**. The Socket Connection page is displayed.
8. Enter `eis/socket/InboundSocketAdapter` in the **Socket Connection JNDI Name** field and then select **Specify Host and Port**, as shown in Figure 5–75.

Figure 5–75 The Adapter Configuration Wizard Socket Connection Page



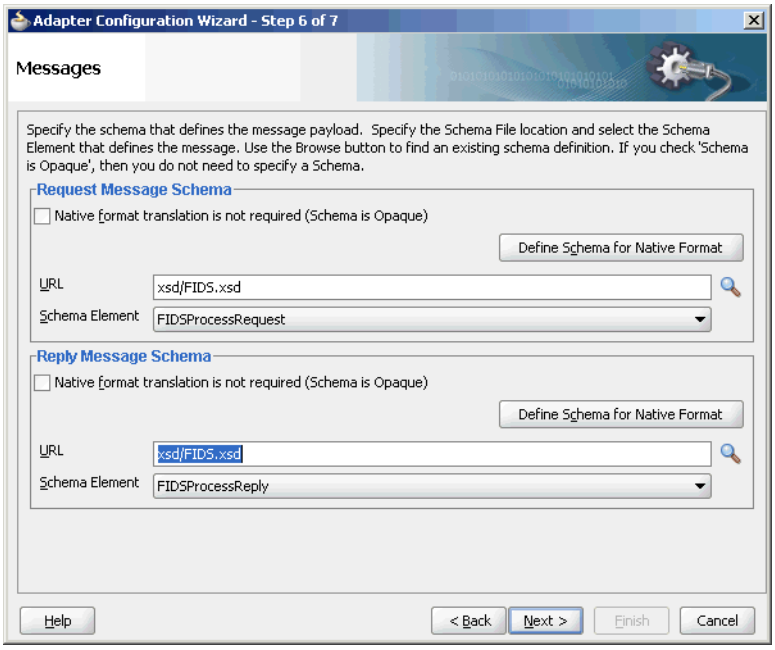
9. Enter `9000` in the PortNumber field and click **Next**. The Messages page is displayed.
10. Click **Browse For Schema File** that appears at the end of the URL field in the Request Message Schema box. The Type Chooser dialog is displayed.
11. Click **Project Schema Files**, **FIDS.xsd**, and **FIDSProcessRequest**, as shown in Figure 5–76.

Figure 5–76 The Type Chooser Dialog



12. Click **OK**. The URL field in the Messages page is populated with the FIDS.xsd file.
13. Click **Browse For Schema File** that appears at the end of the URL field in the Reply Message Schema box. The Type Chooser dialog is displayed.
14. Click **Project Schema Files**, **FIDS.xsd**, and **FIDSProcessReply**.
15. Click **OK**. The URL fields in the Messages page are populated with the FIDS.xsd files, as shown in Figure 5–77.

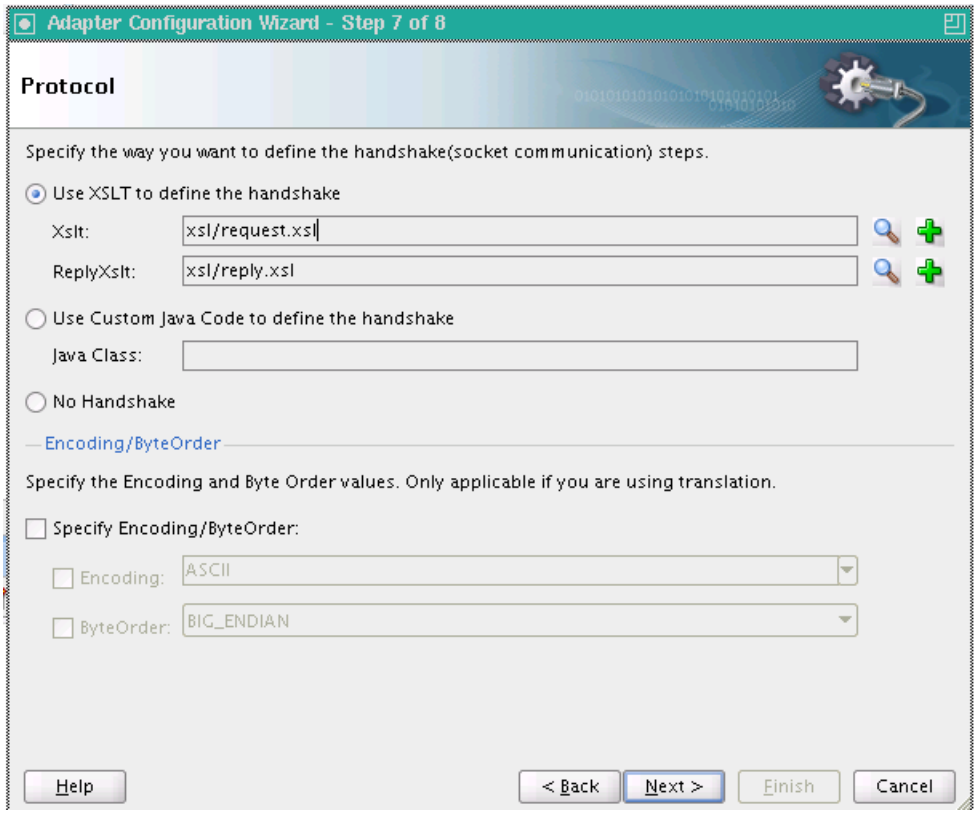
Figure 5–77 The Adapter Configuration Wizard - Messages Page



16. Click **Next**. The Protocol page is displayed.

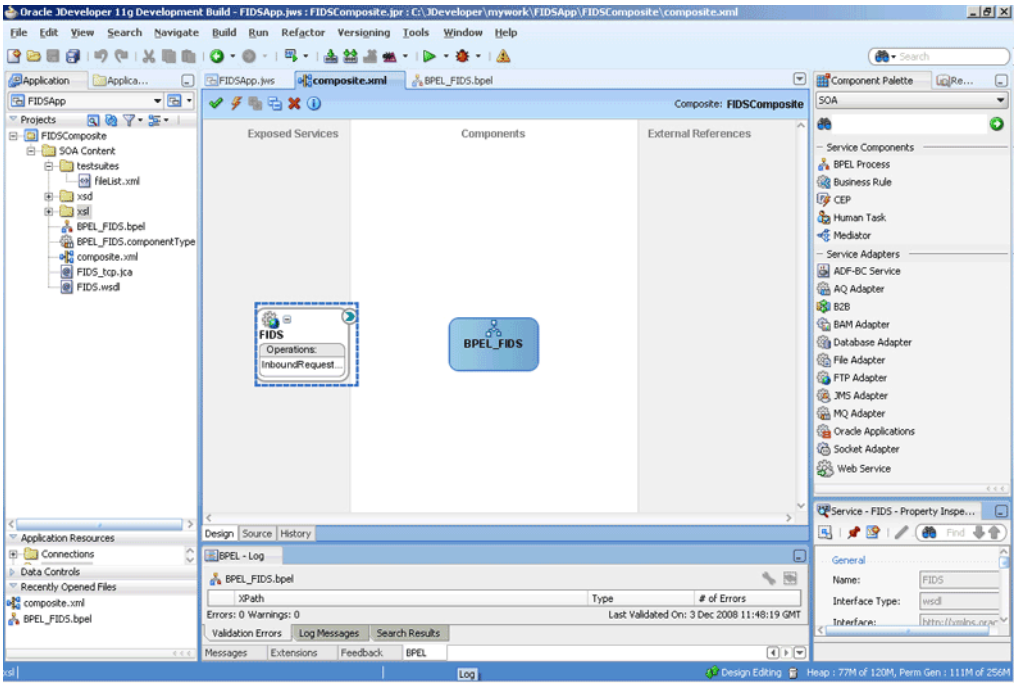
17. Select **Use XSLT to define the handshake**.
18. Click the **Browse to select the XSL file** icon that appears at the end of the Xslt field. The SOA Resource Browser dialog is displayed.
19. Select **request.xsl** as the file name and click **OK**. The Xslt field is populated.
20. Click the **Browse to select the XSL file** icon that appears at the end of the ReplyXslt field. The SOA Resource Browser dialog is displayed.
21. Select **reply.xsl** as the file name and click **OK**. The Xslt field is populated, as shown in Figure 5–78.

Figure 5–78 The Adapter Configuration Wizard - Protocol Page



22. Click **Finish**. The composite.xml page appears, as shown in Figure 5–79.

Figure 5–79 The JDeveloper - composite.xml Page



5.5.2.4 Creating Outbound Oracle Socket Adapter Services

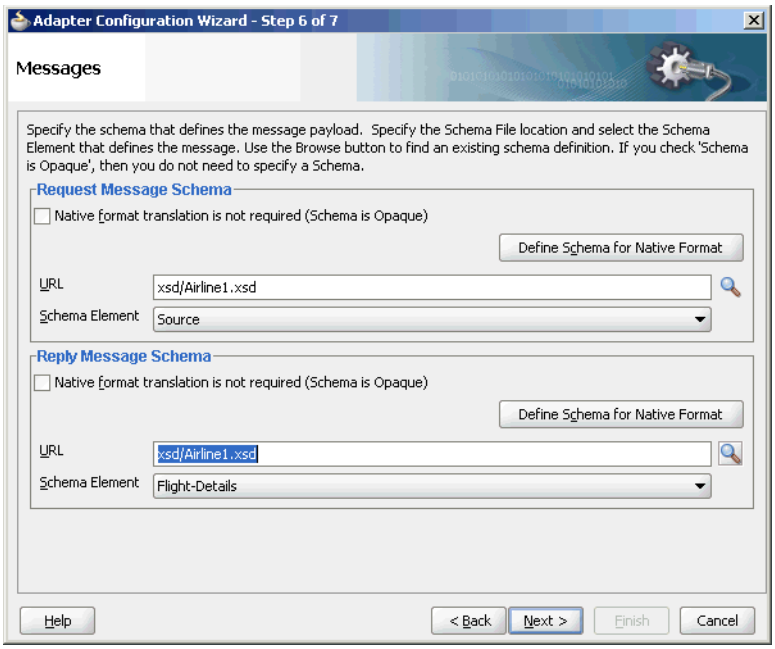
Perform the following steps to create an outbound Oracle Socket Adapter service for the Airline1 server socket:

1. Drag and drop Socket Adapter from the Component Palette to the External References swim lane. The Welcome page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `Airline1` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Outbound Synchronous Request/Reply** as the Operation Type.
7. Click **Next**. The Socket Connection page is displayed.
8. Enter `eis/socket/OutboundSocketAdapter` in the **Socket Connection JNDI Name** field, as shown in Figure 5–80, and then select **Specify Host and Port**.

Figure 5–80 The Adapter Configuration Wizard - Socket Connection Page

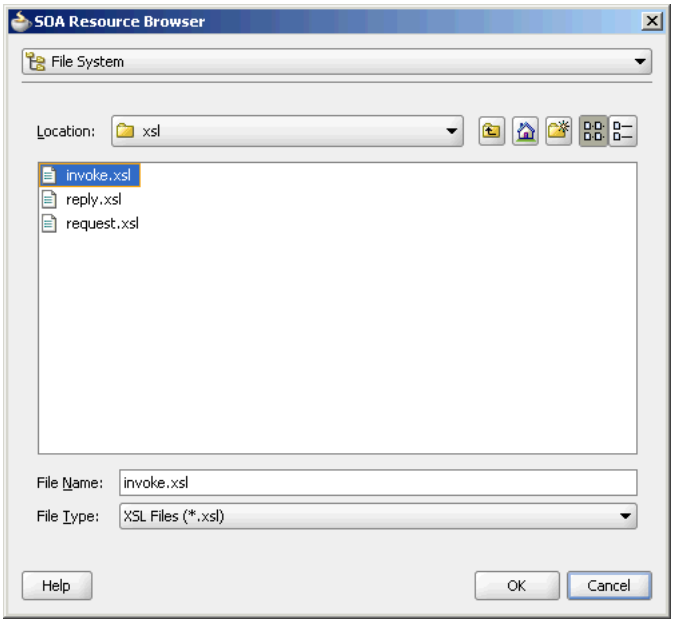
9. Enter the name of the system where the Airline1 socket server program must run in the **HostName** field and `9001` in the **PortNumber** field, and click **Next**. The Messages page is displayed.
10. Click **Browse For Schema File** that appears at the end of the URL field in the Request Message Schema box. The Type Chooser dialog is displayed.
11. Click **Project Schema Files**, **Airline1.xsd**, and **Source**.
12. Click **OK**. The URL field in the Messages page is populated with the **Airline1.xsd** file.
13. Click **Browse For Schema File** that appears at the end of the URL field in the Reply Message Schema box. The Type Chooser dialog is displayed.
14. Click **Project Schema Files**, **Airline1.xsd**, and **Flight-Details**.
15. Click **OK**. The URL fields in the Messages page are populated with the `Airline1.xsd` files, as shown in Figure 5–81.

Figure 5–81 The Adapter Configuration Wizard - Messages Page



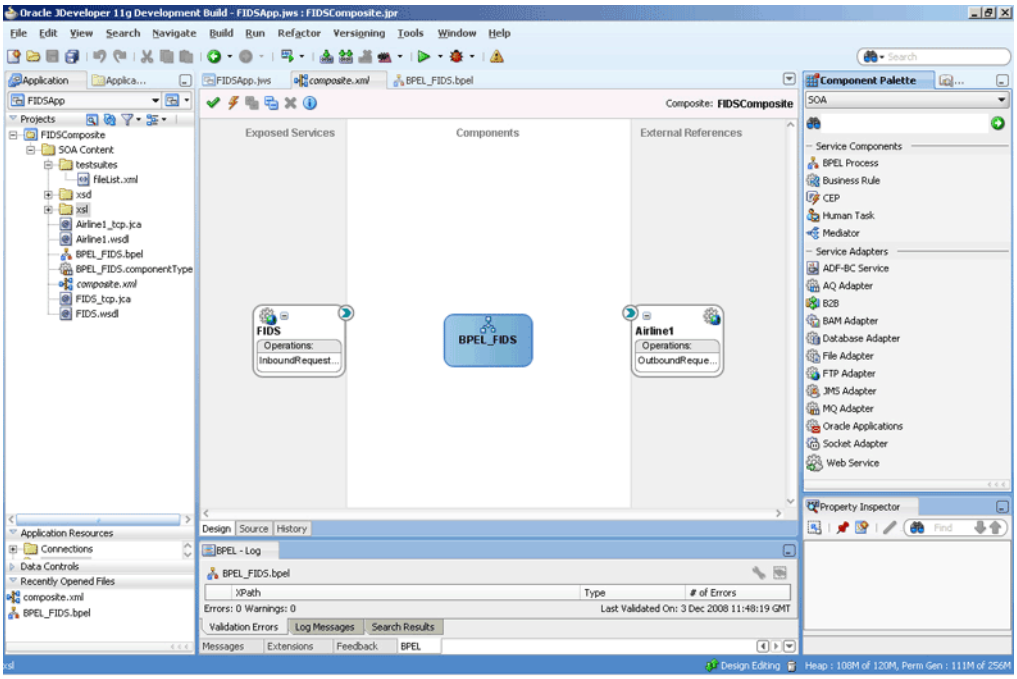
16. Click **Next**. The Protocol page is displayed.
17. Select **Use XSLT to define the handshake**.
18. Click **Browse to select the XSL file** that appears at the end of the Xslt field. The SOA Resource Browser dialog is displayed.
19. Select **invoke.xsl** as the file name, as shown in Figure 5–82, and click **OK**. The Xslt field is populated.

Figure 5–82 The SOA Resource Browser Dialog



20. Click **Finish**. The composite.xml page appears, as shown in Figure 5–83.

Figure 5–83 The JDeveloper - composite.xml Page

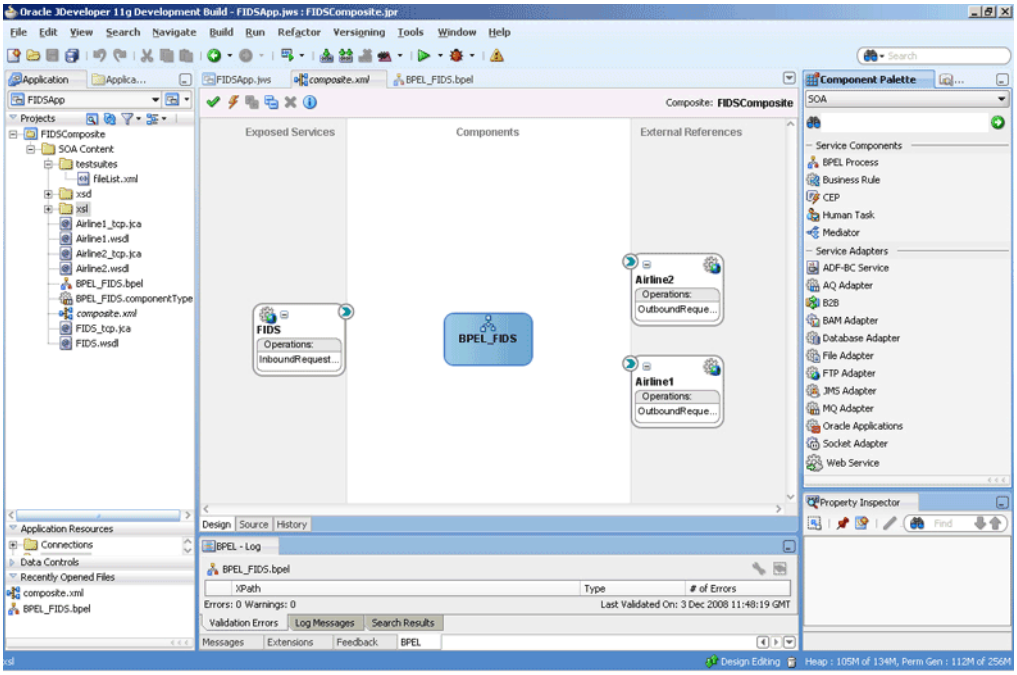


Perform the following steps to create an outbound Oracle Socket Adapter service for the Airline2 server socket:

1. Drag and drop **Socket Adapter** from the Component Palette to the External References swim lane. The Welcome page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `Airline2` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Outbound Synchronous Request/Reply** as the operation type.
7. Click **Next**. The Socket Connection page is displayed.
8. Enter `eis/socket/OutboundSocketAdapter` in the **Socket Connection JNDI Name** field and then select **Specify Host and Port**.
9. Enter the name of the system where the Airline2 socket server program must run in the **HostName** field and `9002` in the **PortNumber** field, and click **Next**. The Messages page is displayed.
10. Click **Browse For Schema File** that appears at the end of the URL field in the Request Message Schema box. The Type Chooser dialog is displayed.
11. Click **Project Schema Files**, **Airline2.xsd**, and **Source**.
12. Click **OK**. The URL field in the Messages page is populated with the `Airline2.xsd` file.
13. Click **Browse For Schema File** that appears at the end of the URL field in the Reply Message Schema box. The Type Chooser dialog is displayed.
14. Click **Project Schema Files**, **Airline2.xsd**, and **flight-details**.

15. Click **OK**. The URL fields in the Messages page are populated with the `Airline2.xsd` files.
16. Click **Next**. The Protocol page is displayed.
17. Select **No Handshake**.
18. Click **Finish**. The composite.xml page appears, as shown in Figure 5–84.

Figure 5–84 The JDeveloper - composite.xml Page

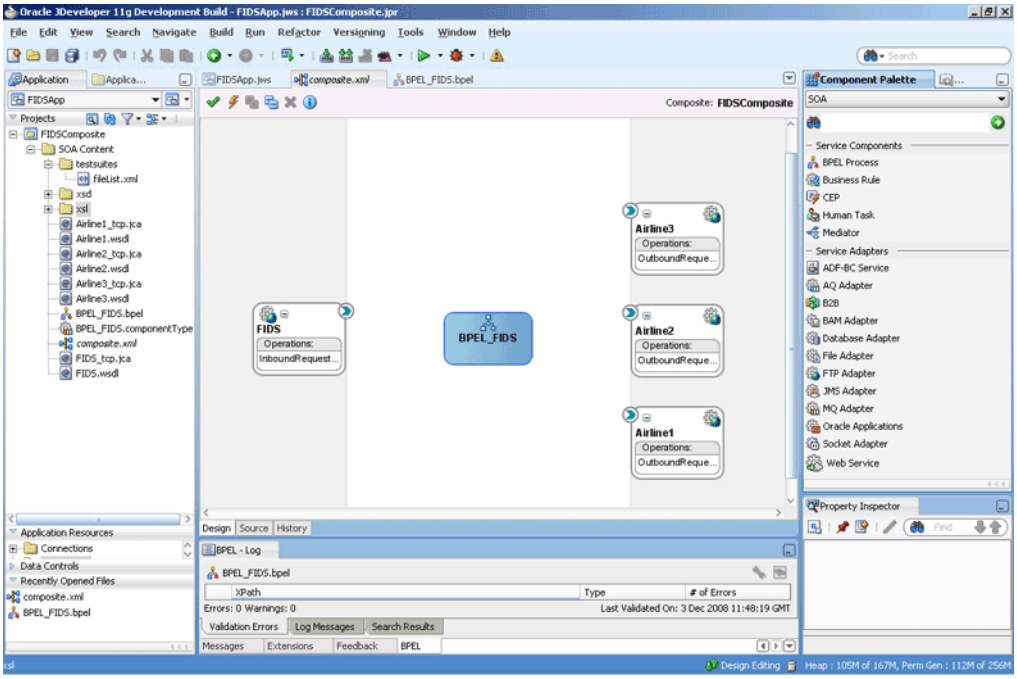


Perform the following steps to create an outbound Oracle Socket Adapter service for the Airline3 server socket:

1. Drag and drop **Socket Adapter** from the Component Palette to the External References swim lane. The Welcome page of the Adapter Configuration Wizard is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `Airline3` in the **Service Name** field.
4. Click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
6. Select **Outbound Synchronous Request/Reply** as the operation type.
7. Click **Next**. The Socket Connection page is displayed.
8. Enter `eis/socket/OutboundSocketAdapter` in the **Socket Connection JNDI Name** field and then select **Specify Host and Port**.
9. Enter the name of the system where the Airline3 socket server program must run in the **HostName** field and `9003` in the **PortNumber** field, and click **Next**. The Messages page is displayed.
10. Click **Browse For Schema File** that appears at the end of the URL field in the Request Message Schema box. The Type Chooser dialog is displayed.

11. Click **Project Schema Files**, **Airline3.xsd**, and **src**.
12. Click **OK**. The URL field in the Messages page is populated with the `Airline3.xsd` file.
13. Click **Browse For Schema File** that appears at the end of the URL field in the Reply Message Schema box. The Type Chooser dialog is displayed.
14. Click **Project Schema Files**, **Airline3.xsd**, and **airline**.
15. Click **OK**. The URL fields in the Messages page are populated with the `Airline3.xsd` files.
16. Click **Next**. The Protocol page is displayed.
17. Select **Use Custom Java Code to define the handshake**.
18. Enter `com.oracle.socket.fids.custom.Airline3Custom` in the Java Class field.
19. Click **Finish**. The composite.xml page appears, as shown in Figure 5–85.

Figure 5–85 The JDeveloper - composite.xml Page



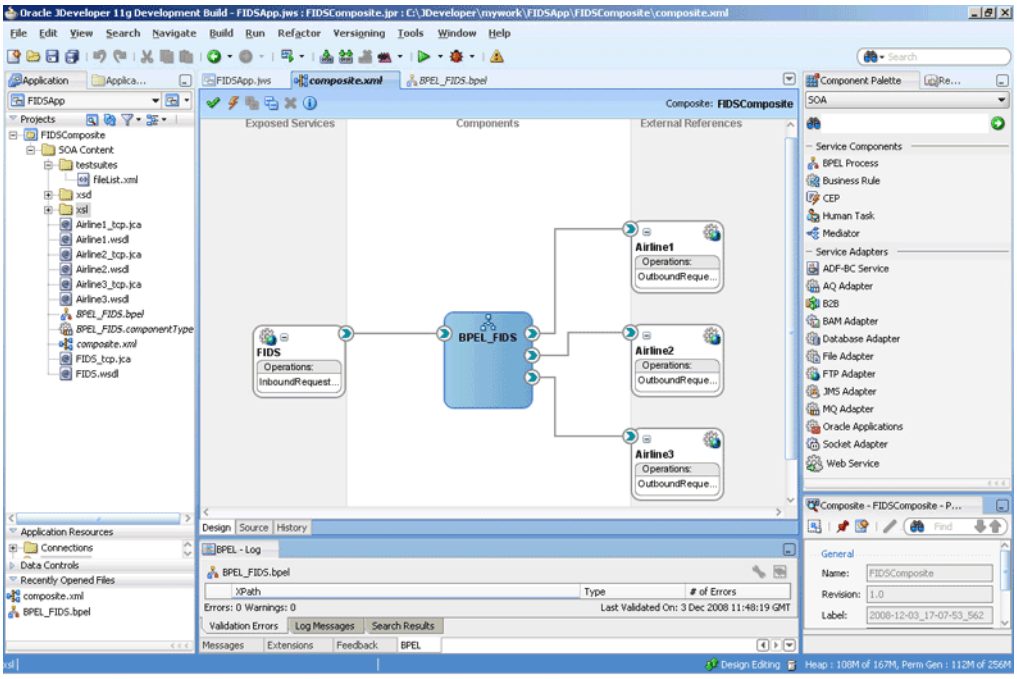
5.5.2.5 Wiring Services and Activities

You have to assemble or wire the components that you have created: Inbound adapter service, BPEL process, Outbound adapter references. Perform the following steps to wire the components:

1. Drag the small triangle in the FIDS client in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL_FIDS process in the Components area.
2. Drag the small triangle in the BPEL_FIDS process in the Components area to the drop zone that appears as a green triangle in the Airline1, Airline2, and Airline3 servers in the External References area.

   The JDeveloper composite.xml file appears, as shown in Figure 5–86.

Figure 5–86 The JDeveloper - composite.xml

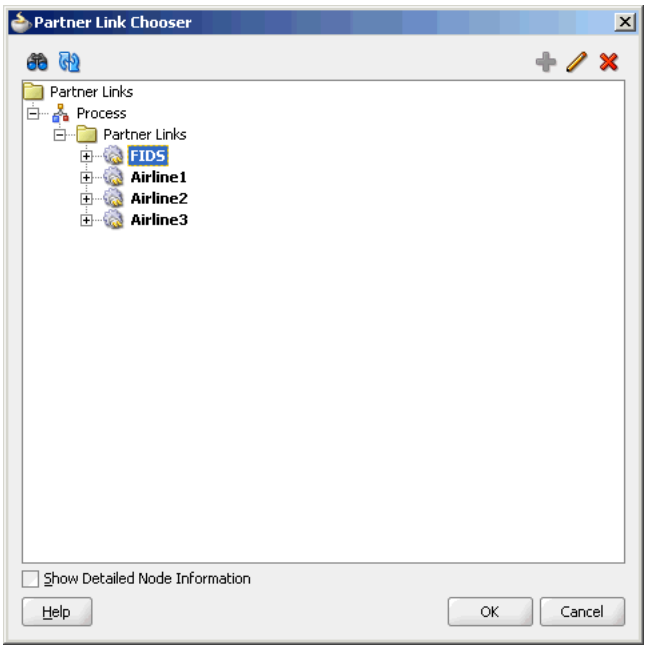


3. Click **File**, **Save All**.

Add a Receive Activity

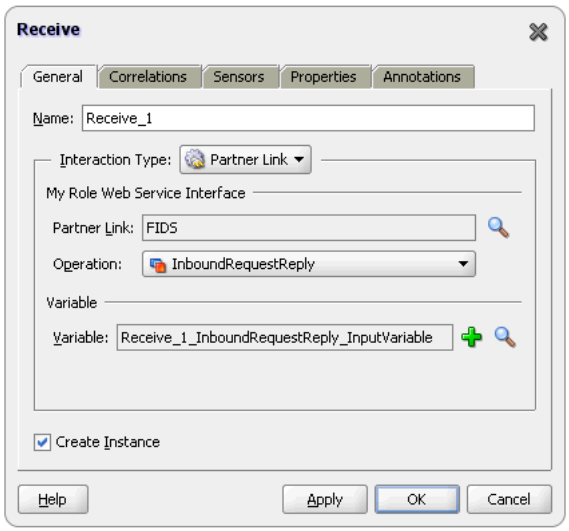
1. Double-click **BPEL_FIDS**. The BPELFIDS.bpel page is displayed.
2. Drag and drop a **Receive** activity from the Component Palette to the design area.
3. Double-click the **Receive** activity. The Receive dialog is displayed.
4. Retain the default name `Receive_1` in the **Name** field.
5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
6. Select **FIDS**, as shown in Figure 5–87, and click **OK**.

Figure 5–87 The Partner Link Chooser Dialog



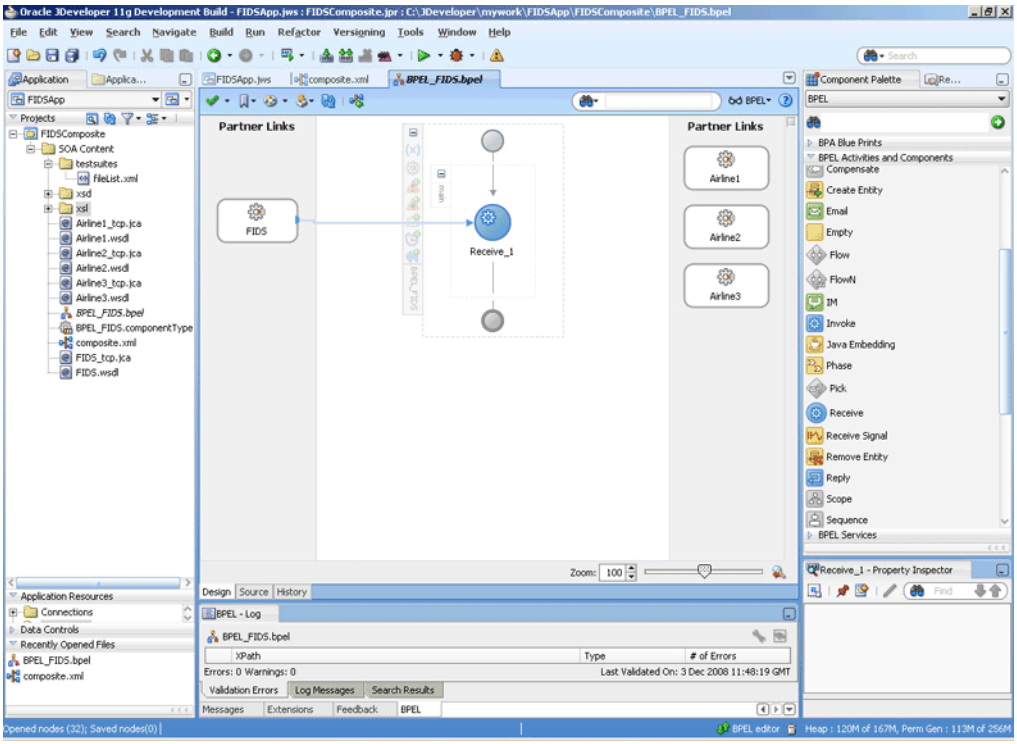
7. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog, as shown in Figure 5–88. The Create Variable dialog is displayed.

Figure 5–88 The Receive Dialog



8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
9. Check **Create Instance**, and click **OK**. The JDeveloper BPEL_FIDS.bpel page appears, as shown in Figure 5–89.

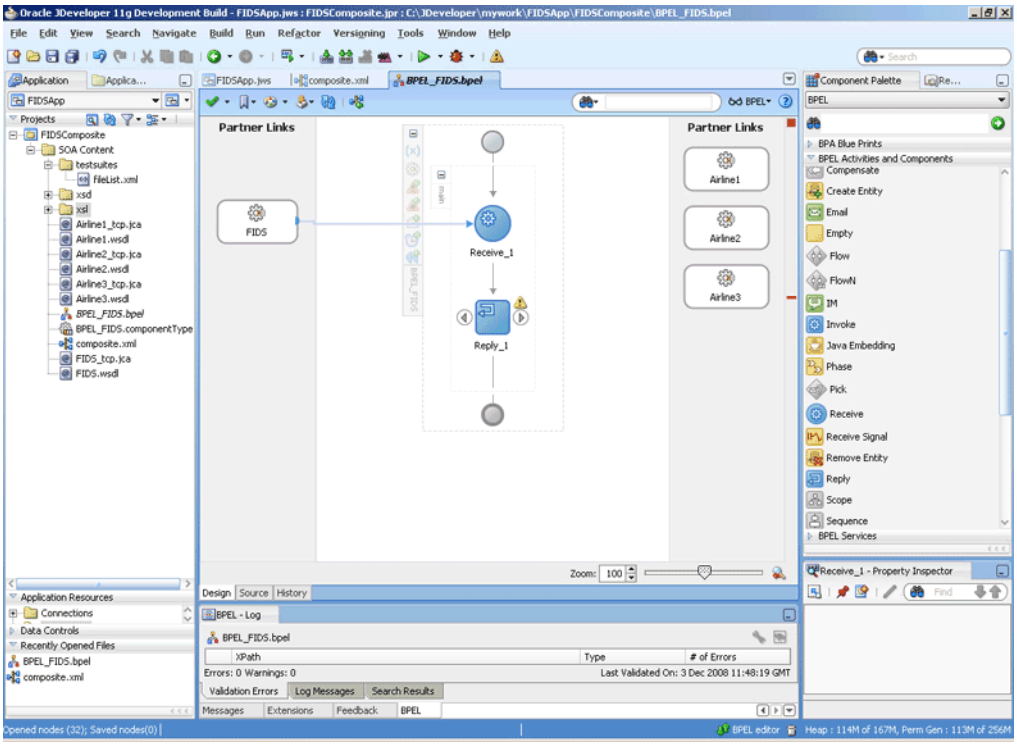
Figure 5–89 The JDeveloper - BPEL_FIDS.bpel



Add a Reply Activity

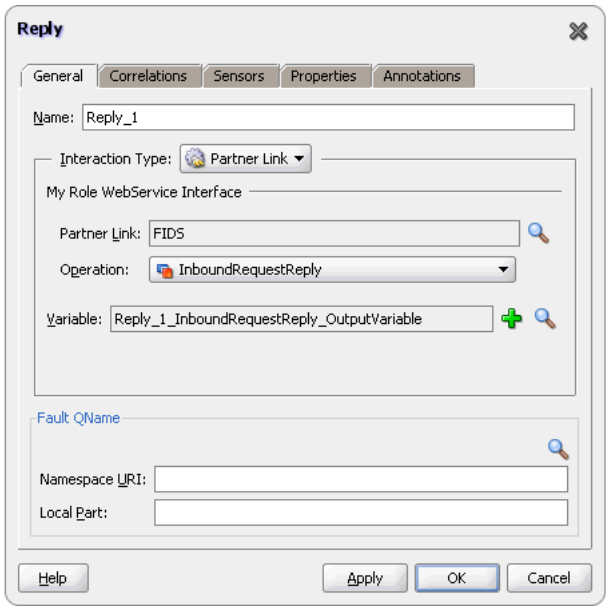
1. Drag and drop an **Reply** activity from the Component Palette to the design area, as shown in Figure 5–90.

Figure 5–90 The JDeveloper - BPEL_FIDS.bpel



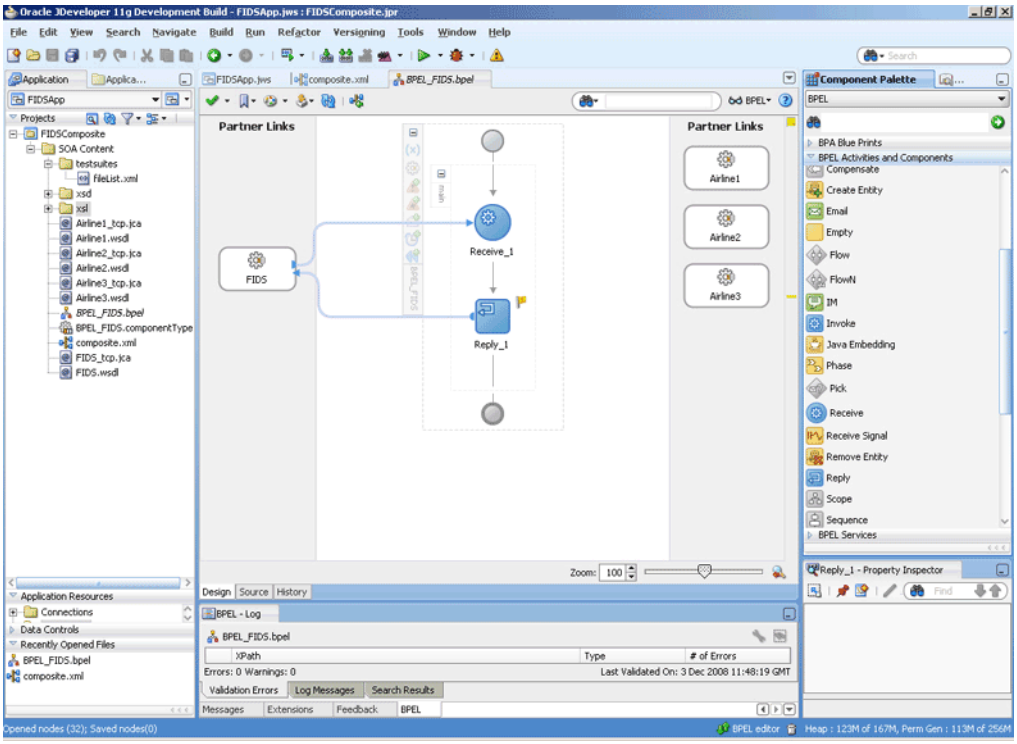
2. Double-click the **Reply** activity. The Reply dialog is displayed.
3. Retain the default name `Reply_1` in the Name field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **FIDS**, as shown in Figure 5–87, and click **OK**.
6. Click the **Auto-Create Variable** icon to the right of the Variable field in the Reply dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name, as shown in Figure 5–91.

Figure 5–91 The Reply Dialog



8. Click **OK**. The JDeveloper BPEL_FIDS.bpel page appears, as shown in Figure 5–92.

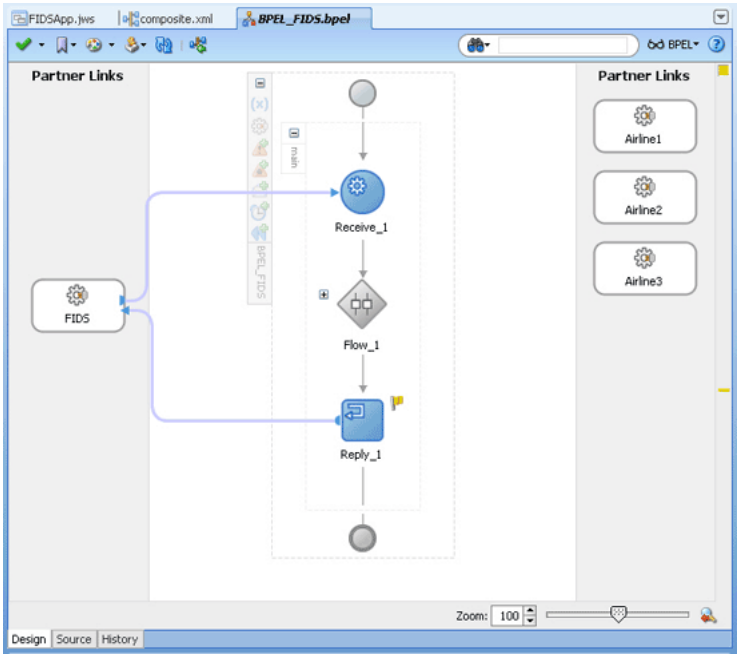
Figure 5–92 The JDeveloper - BPEL_FIDS.bpel



Add a Flow Activity

1. Drag and drop a **Flow** activity from the Component Palette in between the Receive and the Reply activities in the design area, as shown in Figure 5–93.

Figure 5–93 The JDeveloper - BPEL_FIDS.bpel

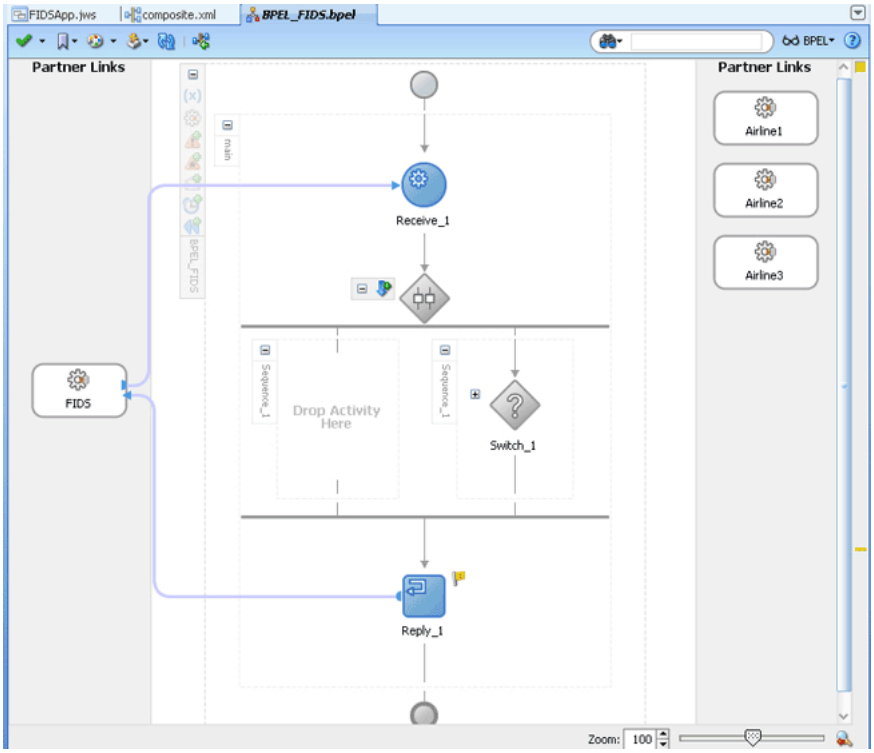


2. Expand the Flow_1 activity. This displays a screen to create sequences.

Design the Flow for Airline1 Server

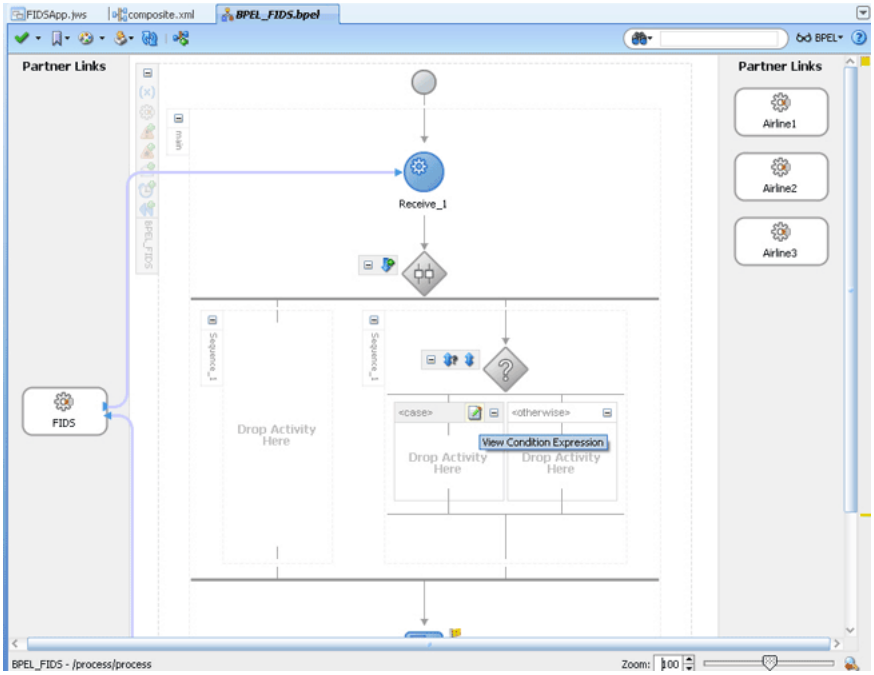
1. Drag and drop a **Switch** activity from the Component Palette to Sequence_1, as shown in Figure 5–94.

Figure 5–94 The JDeveloper - BPEL_FIDS.bpel Page



2. Expand the **Switch** activity. This displays a screen to enter the values for <case> and <otherwise>.
3. In the `<case>` section, click the **View Condition Expression** icon, as shown in Figure 5–95. The Condition Expression pop-up window is displayed.

Figure 5–95 The JDeveloper - BPEL_FIDS.bpel Page

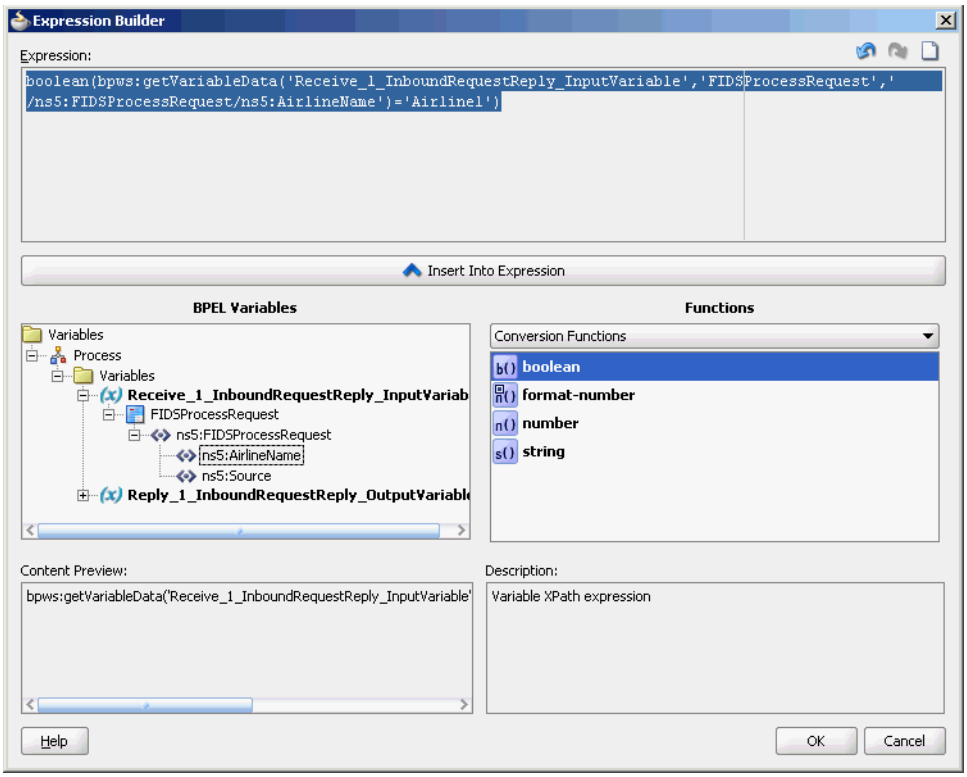


4. Click the **Xpath Expression Builder** icon in the pop-up window. The Expression Builder dialog is displayed.

5. Enter `boolean(bpws:getVariableData('Receive_1_InboundRequestReply_InputVariable','FIDSProcessRequest','/ns5:FIDSProcessRequest/ns5:AirlineName')='Airline1')` as the expression, as shown in Figure 5–96, and click **OK**. The screen returns to the Condition Expression pop-up window.

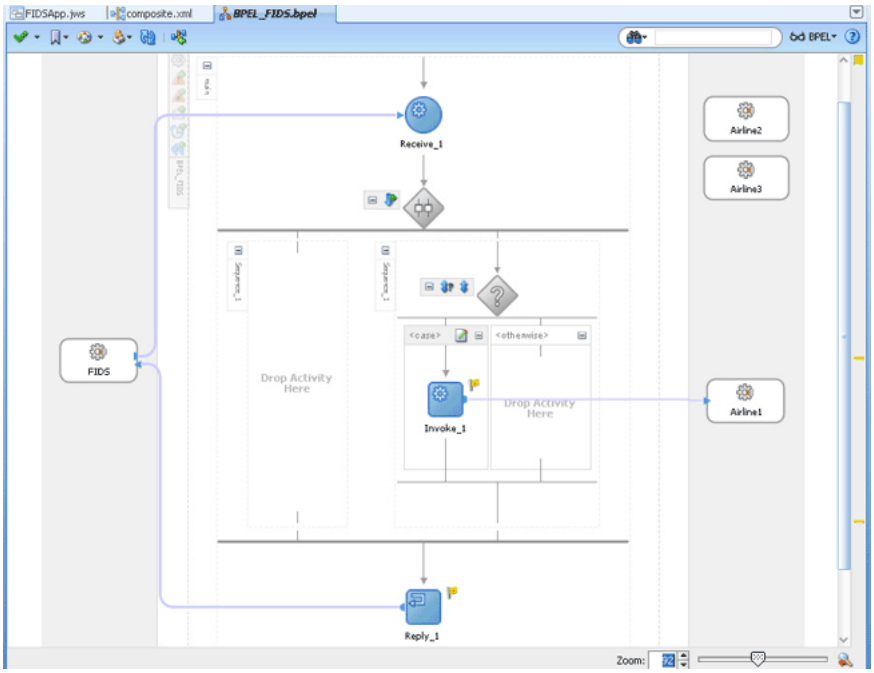
> **Note:** This expression ensures that this flow is executed only when information for Airline1 is requested.

Figure 5–96 The Expression Builder Dialog



6. Add an invoke activity to the <case> section.
   a. Drag and drop an **Invoke** activity in the <case> section.
   b. Double-click the **Invoke** activity. The Invoke dialog is displayed.
   c. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
   d. Select **Airline1**, and click **OK**.
   e. Click the **Automatically Create Input Variable** and the **Automatically Create Output Variable** icons to the right of the Input and Output Variable fields in the Invoke dialog. The Create Variable dialogs are displayed.
   f. Select the default variable names and click **OK**. The Variable fields are populated with the default variable name. The Invoke dialog is displayed.
   g. Click **OK**. The JDeveloper BPEL_FIDS.bpel page is displayed, as shown in Figure 5–97.

Figure 5–97 The JDeveloper - BPEL_FIDS.bpel



7. Add an assign activity to the <case> section.

   a. Drag and drop an **Assign** activity from the Component Palette before the Invoke_1 activity in the <case> section.

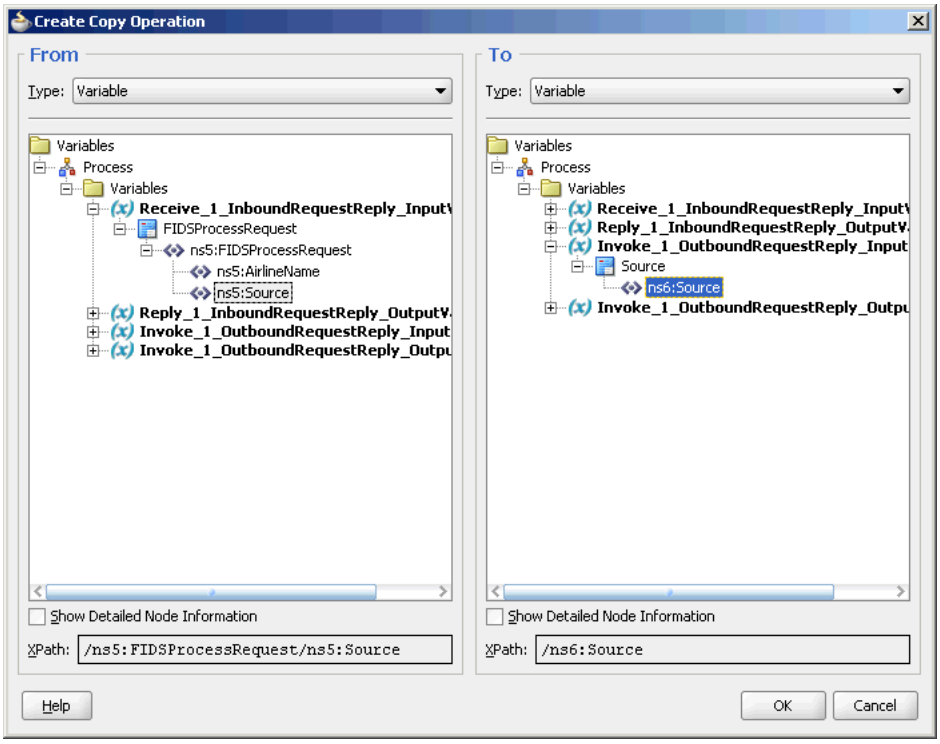
   b. Double-click the **Assign_1** activity. The Assign dialog is displayed.

   c. Click the **Copy Operation** tab. The Assign dialog is displayed.

   d. Select **Copy Operation**. The Create Copy Operation dialog is displayed.

   e. Create the copy operation between the source from the input variable of the Receive_1 activity and the source from the input variable of the Invoke_1 activity, as shown in Figure 5–98.

Figure 5–98 The Create Copy Operation Dialog

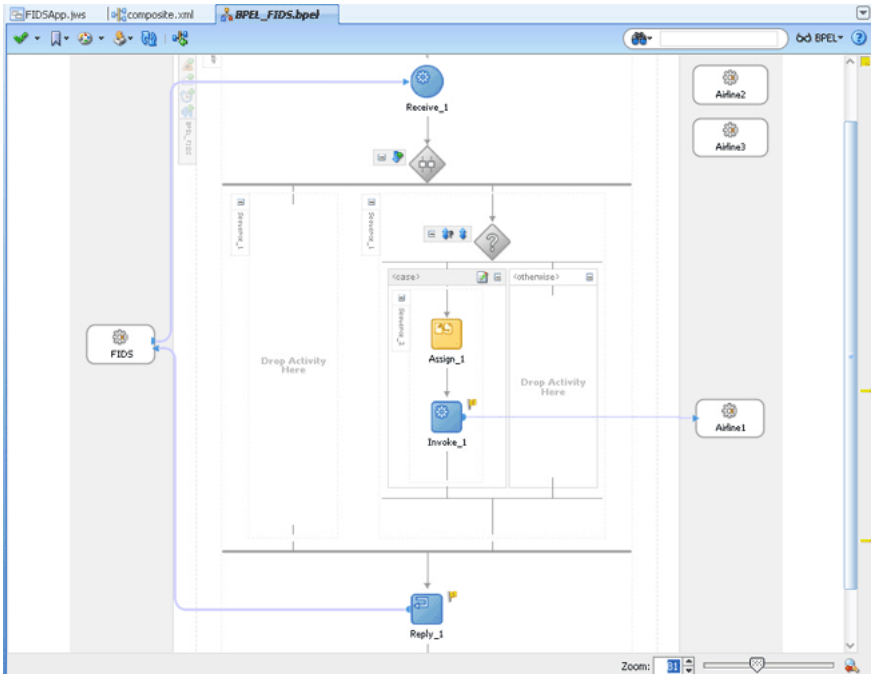


f. Click **OK** in the Create Copy Operation dialog.

g. Click **OK**.

The BPEL_FIDS.bpel page is displayed, as shown in Figure 5–99.

Figure 5–99 The JDeveloper - BPELFIDS.bpel



8. Add a Transform activity to the <case> section.

a. Drag and drop a **Transform** activity in the <case> section, after the Invoke_1 activity.

b. Double-click the **Transform** activity.

c. Click the **Transformation** tab.

d. Click the **Create...(Alt+N)** icon. The Source Variable dialog is displayed.

e. Select **Invoke_1_OutboundRequestReply_OutputVariable** from the Source Variable list and click **OK**.

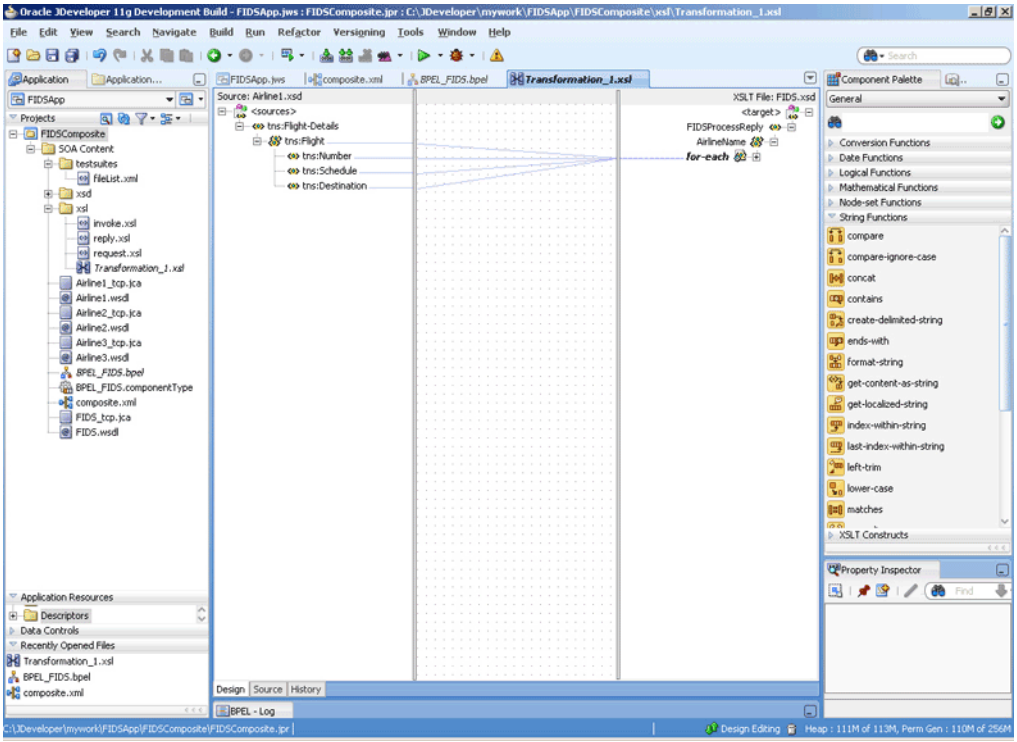
f. Select **Reply_1_InboundRequestReply_OutputVariable** from the Target Variable list.

g. Click **OK**. The XSL mapper tool is displayed.

h. Link the `tns:Flight` node from the source, on the left pane to the target `FlightDetails` node on the right pane. The Auto Map Preferences dialog appears.

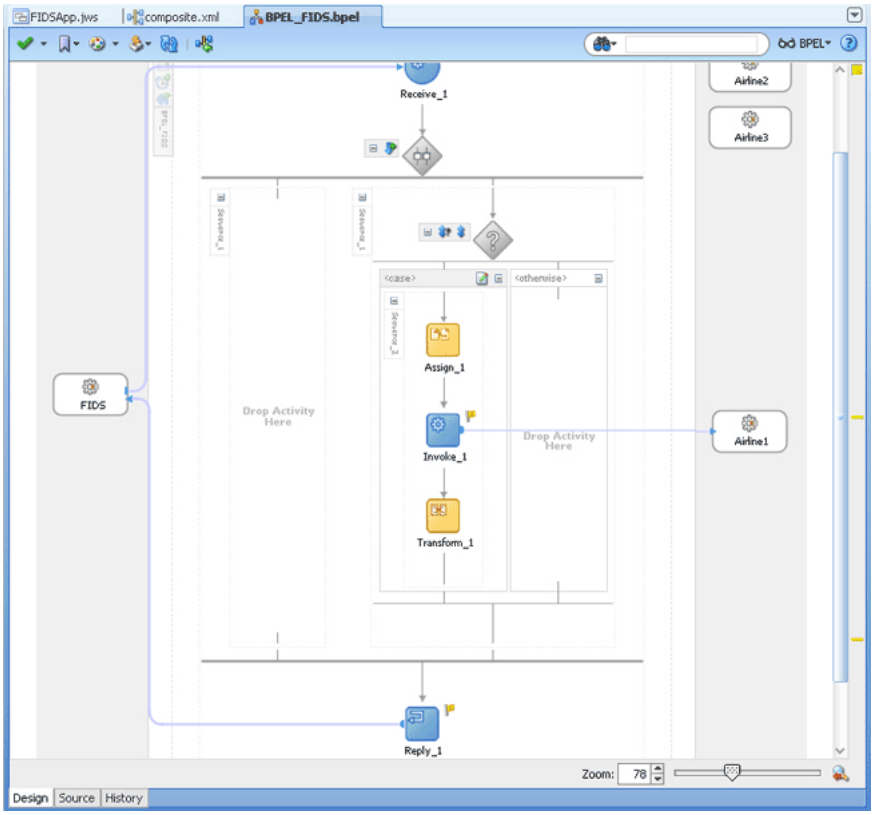
i. Click **OK**. The Transformation_1.xsl (XSL mapper tool) is displayed, as shown in Figure 5–100.

Figure 5–100 The JDeveloper - Transformation_1.XSL Page



9. Click **File**, **Save All**. The BPEL_FIDS.bpel page is displayed, as shown in Figure 5–101, with the flow defined for the Airline1 server.

Figure 5–101 The JDeveloper - BPEL_FIDS.bpel



Design the Flow for Airline2 Server

1. Double-click the empty Sequence activity and enter `Sequence_2` in the **Name** field.
2. Drag and drop a **Switch** activity from the Component Palette to Sequence_2.
3. Expand the **Switch** activity. This displays a screen to enter the values for <case> and <otherwise>.
4. In the `<case>` section, click the **View Condition Expression** icon. The Condition Expression pop-up window is displayed.
5. Click the **Xpath Expression Builder** icon in the pop-up window. The Expression Builder dialog is displayed.
6. Enter `boolean(bpws:getVariableData('Receive_1_InboundRequestReply_InputVariable','FIDSProcessRequest','/ns5:FIDSProcessRequest/ns5:AirlineName')='Airline2')` as the expression, and click **OK**. The screen returns to the Condition Expression pop-up window.

   Note: This expression ensures that this flow is executed only when information for Airline2 is requested.

7. Add an invoke activity to the <case> section.
   a. Drag and drop an **Invoke** activity in the <case> section.
   b. Double-click the **Invoke_2** activity. The Invoke dialog is displayed.

c. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.

d. Select **Airline2**, and click **OK**.

e. Click the **Automatically Create Input Variable** and the **Automatically Create Output Variable** icons to the right of the Input and Output Variable fields in the Invoke dialog. The Create Variable dialogs are displayed.

f. Select the default variable names and click **OK**. The Input and Output Variable fields are populated with the default variable names. The Invoke dialog is displayed.

g. Click **OK**. An Invoke activity is added to the JDeveloper BPEL_FIDS.bpel page under Sequence_2.

8. Add an assign activity to the <case> section.

a. Drag and drop an **Assign** activity from the Component Palette before the Invoke activity in the <case> section.

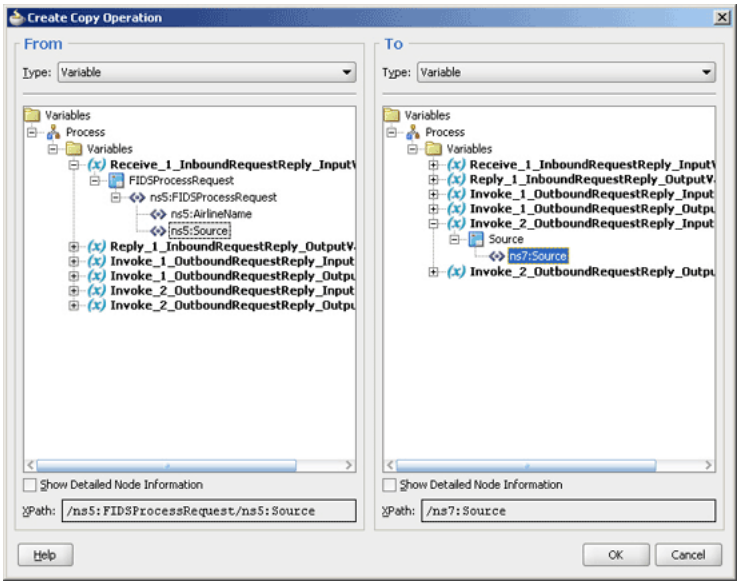
b. Double-click the **Assign_2** activity. The Assign dialog is displayed.

c. Click the **Copy Operation** tab. The Assign dialog is displayed.

d. Select **Copy Operation**. The Create Copy Operation dialog is displayed.

e. Create the copy operation between the source from the input variable of the Receive_1 activity and the source from the input variable of the Invoke_2 activity, as shown in Figure 5–102.

Figure 5–102 The Create Copy Operation Dialog



f. Click **OK** in the Create Copy Operation dialog.

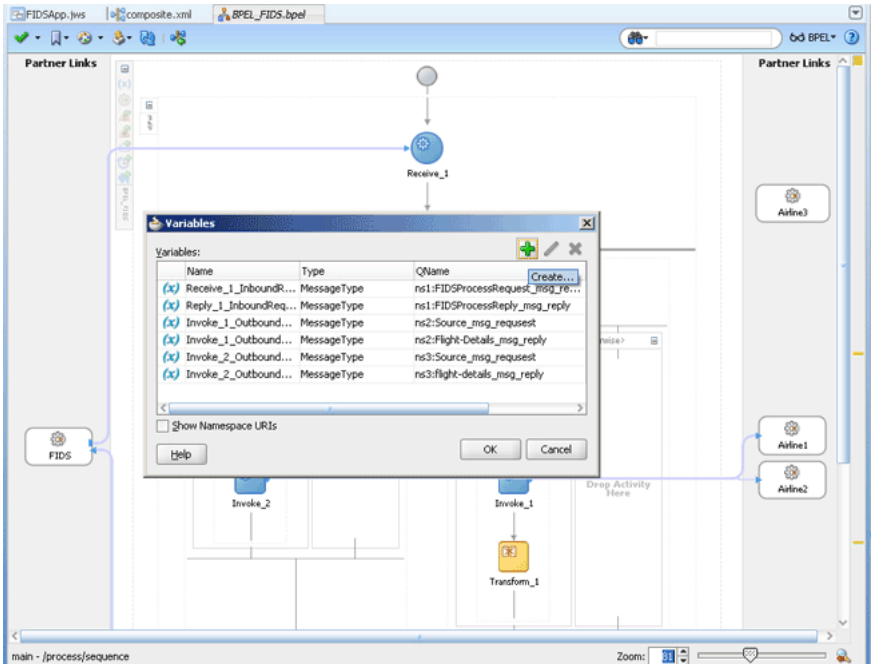
g. Click **OK**.

9. Create a temporary variable and add a Transform activity to the <case> section.

Note: The temporary variable is used for storing flight details from the Airline2 server, which would later be appended to the reply variable.

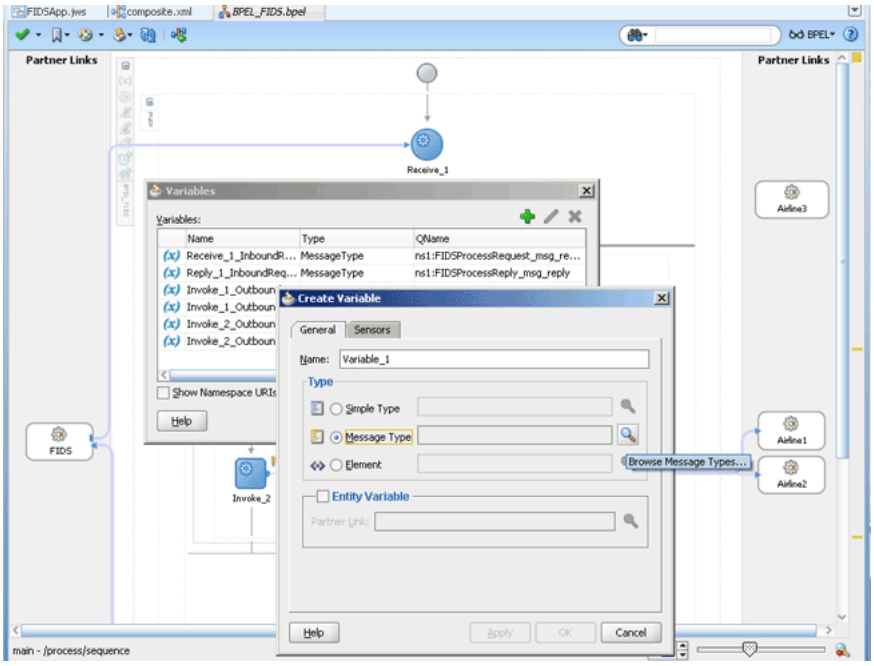
a. Click the **Variables...** icon represented by (x). The Variables dialog is displayed, as shown in Figure 5–103.

Figure 5–103 The Variables Dialog



b. Click the **Create...** icon. The Create Variable dialog is displayed, as shown in Figure 5–104.

Figure 5–104 The Create Variable Dialog

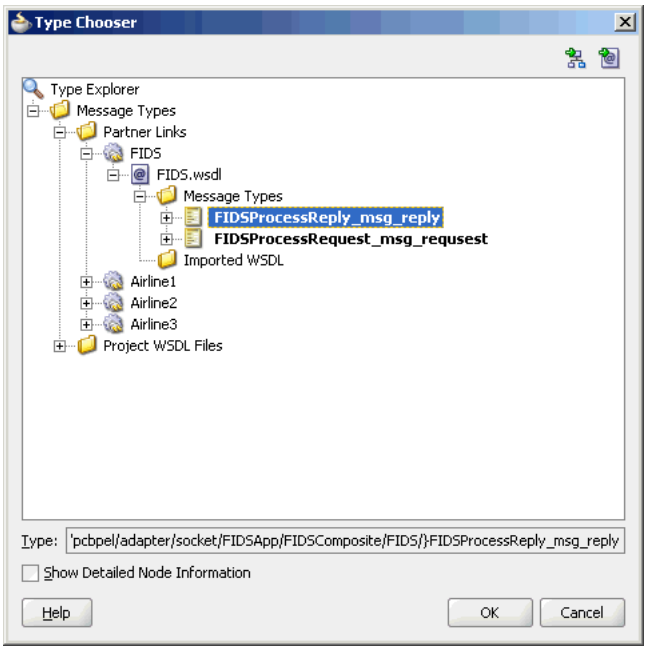


c. Select **Message Type** as the variable type.

d. Click the **Browse Message Types** icon at the end of the Message Type field. The Type Chooser dialog is displayed.

e. Click **Message Types**, **Partner Links**, **FIDS**, **FIDS.wsdl**, **Message Types**, and then **FIDSProcessReply_msg_reply**, as shown in Figure 5–105.

Figure 5–105 The Type Chooser Dialog



f. Click **OK**. The Message type field in the Create Variable dialog is populated with the FIDSProcessReply_msg_reply partner link.

g. Click **OK**. A variable, Variable_1, of type Message Type is added to the Variables list in the Variable dialog.

h. Click **OK** to return to the BPEL_FIDS.bpel page.

i. Drag and drop a **Transform** activity in the <case> section, after the Invoke_2 activity.

j. Double-click the **Transform_2** activity.

k. Click the **Transformation** tab.

l. Click the **Create...** icon. The Source Variable dialog is displayed.

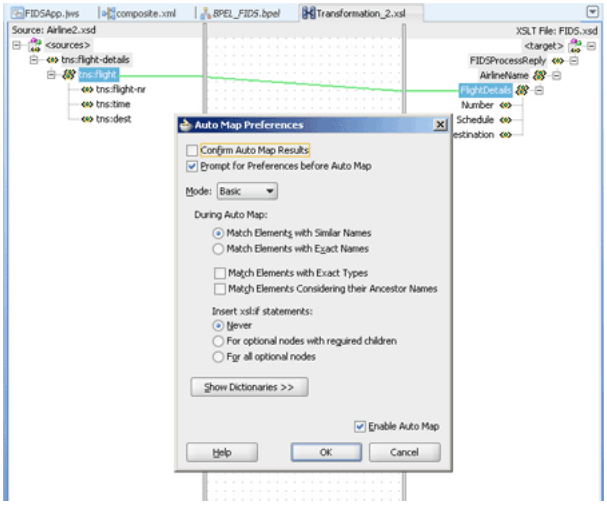
m. Select **Invoke_2_OutboundRequestReply_OutputVariable** from the Source Variable list and click **OK**.

n. Select **Variable_1** from the Target Variable list.

o. Click **OK**. The XSL mapper tool is displayed.

p. Link the `tns:flight` node from the source, on the left pane to the target `FlightDetails` node on the right pane. The Auto Map Preferences dialog appears, as shown in Figure 5–106.

Figure 5–106 The Transformation_2.XSL Page With Auto Map Preference Dialog

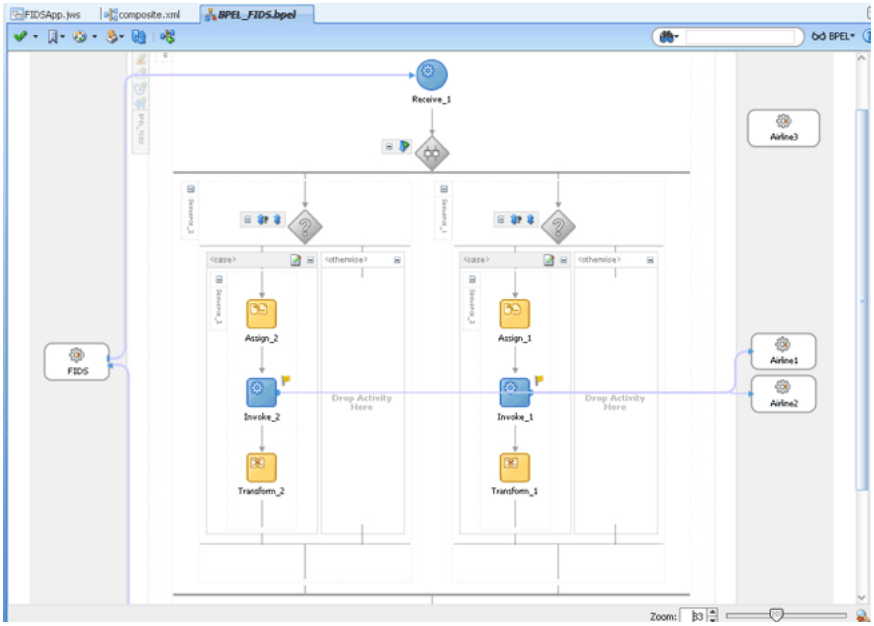


q. Click **OK**. The Transformation_2.xsl (XSL mapper tool) file with the XSL mapping is displayed.

10. Click **File**, **Save All**.

    The BPEL_FIDS.bpel page is displayed, as shown in Figure 5–107, with the flow defined for the Airline2 server.

Figure 5–107 The JDeveloper - BPEL_FIDS.bpel



Design the Flow for Airline3 Server

1. Right-click the **Flow_1** activity. Click **Add Sequence** from the menu. `Sequence_3` is added.
2. Drag and drop a **Switch** activity from the Component Palette to Sequence_3.
3. Expand the **Switch** activity. This displays a screen to enter the values for <case> and <otherwise>.
4. In the `<case>` section, click the **View Condition Expression** icon. The Condition Expression pop-up window is displayed.
5. Click the **Xpath Expression Builder** icon in the pop-up window. The Expression Builder dialog is displayed.
6. Enter `boolean(bpws:getVariableData('Receive_1_InboundRequestReply_InputVariable','FIDSProcessRequest','/ns5:FIDSProcessRequest/ns5:AirlineName')='Airline3')` as the expression, and click **OK**. The screen returns to the Condition Expression pop-up window.

   Note: This expression ensures that this flow is executed only when information for Airline3 is requested.

7. Add an invoke activity to the <case> section.
   a. Drag and drop an **Invoke** activity in the <case> section.
   b. Double-click the **Invoke_3** activity. The Invoke dialog is displayed.
   c. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
   d. Select **Airline3**, and click **OK**.

e. Click the **Automatically Create Input Variable** and **Automatically Create Output Variable** icon to the right of the Input and Output Variable field in the Invoke dialog. The Create Variable dialogs are displayed.

f. Select the default variable names and click **OK**. The Input and Output Variable fields are populated with the default variable names. The Invoke dialog is displayed.

g. Click **OK**. An Invoke activity is added to JDeveloper BPEL_FIDS.bpel page under Sequence_3.

8. Add an assign activity to the <case> section.

a. Drag and drop an **Assign** activity from the Component Palette before the Invoke activity in the <case> section.

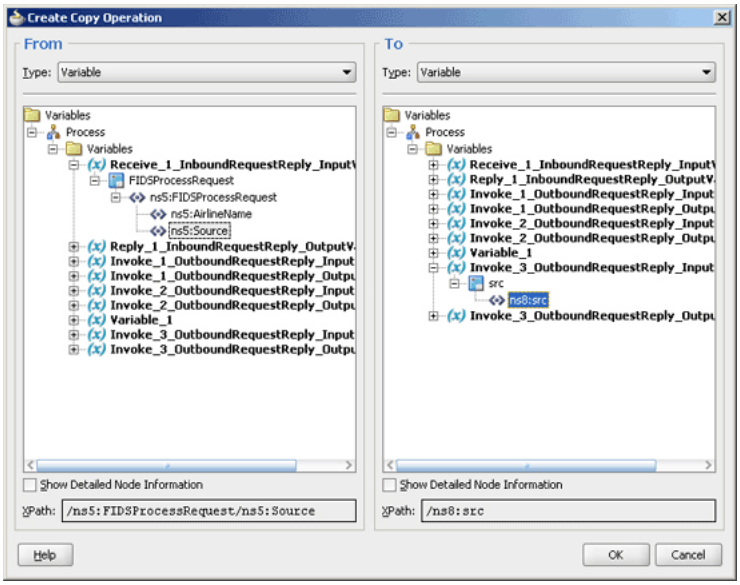
b. Double-click the **Assign_3** activity. The Assign dialog is displayed.

c. Click the **Copy Operation** tab. The Assign dialog is displayed.

d. Select **Copy Operation**. The Create Copy Operation dialog is displayed.

e. Create the copy operation between the source from the input variable of the Receive_1 activity and the source from the input variable of the Invoke_3 activity, as shown in Figure 5–108.

Figure 5–108 The Create Copy Operation Dialog



f. Click **OK** in the Create Copy Operation dialog.

g. Click **OK**.

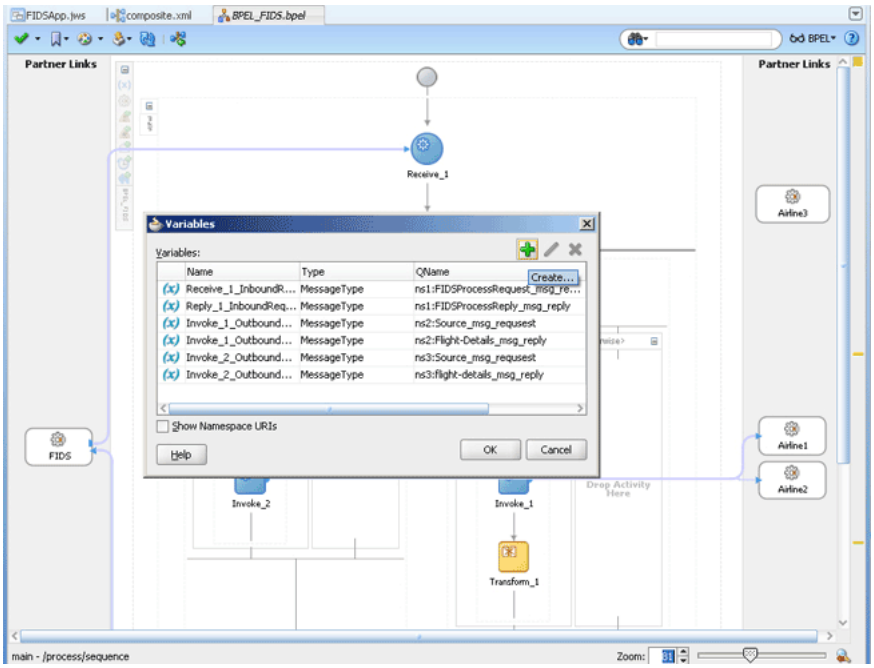
9. Create a temporary variable and add a Transform activity to the <case> section.

Note: The temporary variable is used for storing flight details from the Airline3 server, which would later be appended to the reply variable.

---

a. Click the **Variables...** icon represented by (x). The Variables dialog is displayed, as shown in Figure 5–109.

Figure 5–109 The Variables Dialog



b. Click the **Create...** icon. The Create Variable dialog is displayed.

c. Select **Message Type** as the variable type.

d. Click the **Browse Message Types** icon at the end of the Message Type field. The Type Chooser dialog is displayed.

e. Click **Message Types**, **Partner Links**, **FIDS**, **FIDS.wsdl**, **Message Types**, and then **FIDSProcessReply_msg_reply**, as shown in Figure 5–105.

f. Click **OK**. The Message type field in the Create Variable dialog is populated with the FIDSProcessReply_msg_reply partner link.

g. Click **OK**. A variable, Variable_2, of type Message Type is added to the Variables list in the Variable dialog.

h. Click **OK** to return to the BPEL_FIDS.bpel page.

i. Drag and drop a **Transform** activity in the <case> section, after the Invoke_3 activity.

j. Double-click the **Transform_3** activity.

k. Click the **Transformation** tab.

l. Click the **Create...** icon. The Source Variable dialog is displayed.

m. Select **Invoke_3_OutboundRequestReply_OutputVariable** from the Source Variable list and click **OK**.

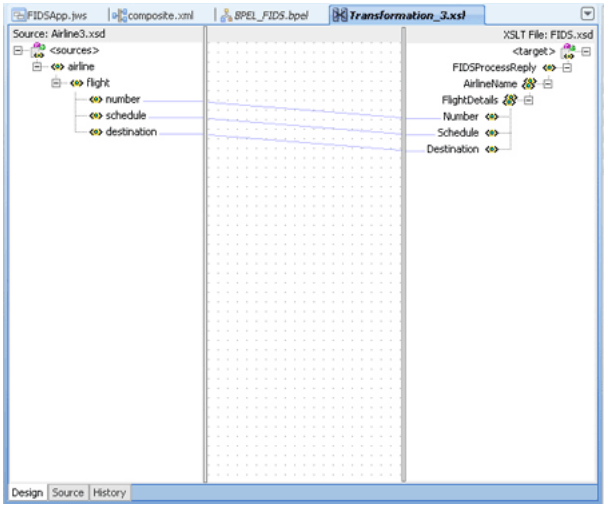
n. Select **Variable_2** from the Target Variable list.

o. Click **OK**. The XSL mapper tool is displayed.

p. Link the `tns:flight` node from the source, on the left pane to the target `FlightDetails` node on the right pane. The Auto Map Preferences dialog appears.

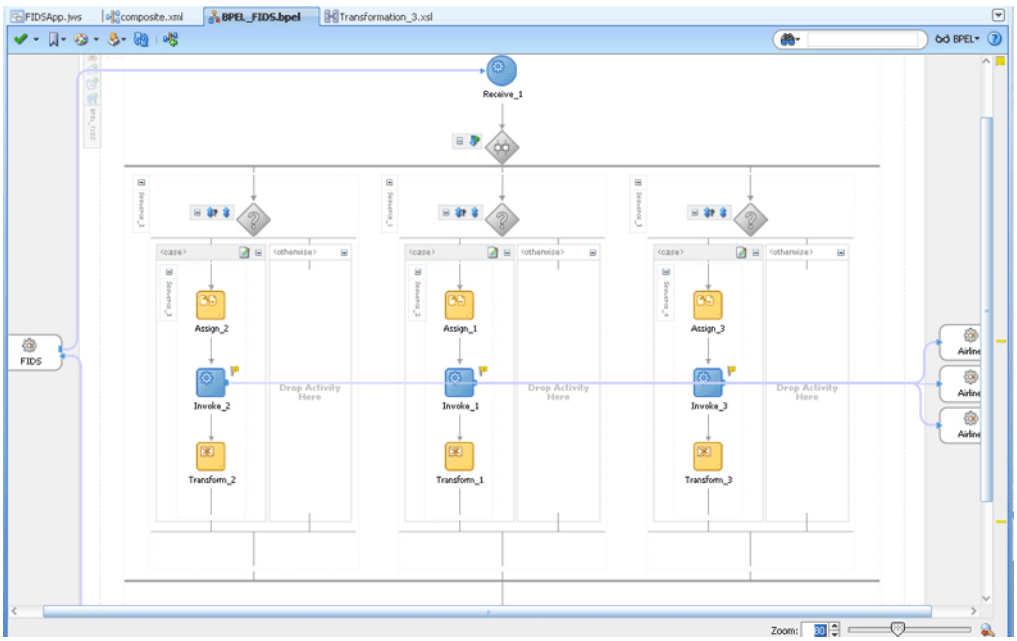
q. Click **OK**. The Transformation_3.xsl (XSL mapper tool) file with the XSL mapping is displayed, as shown in Figure 5–110.

Figure 5–110 The Transformation_3.XSL Page



10. Click **File**, **Save All**. The BPEL_FIDS.bpel page is displayed, as shown in Figure 5–111, with the flow defined for the Airline3 server.

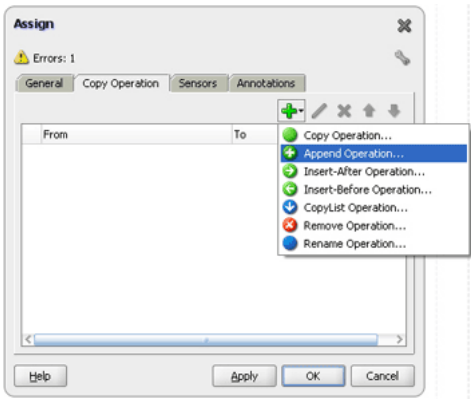
Figure 5–111 The JDeveloper - BPEL_FIDS.bpel



Add an Assign Activity

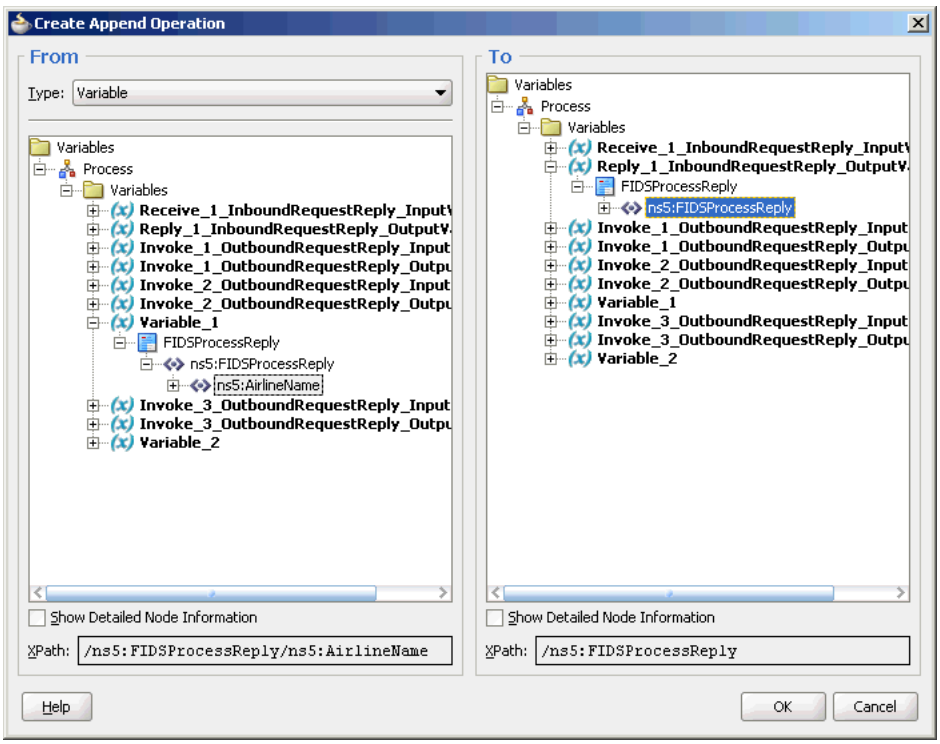
1. Drag and drop an **Assign** activity from the Component Palette in between the Reply and Receive activities in the design area.
2. Double-click the **Assign_4** activity. The Assign dialog is displayed.
3. Click the **Copy Operation** tab. The Assign dialog is displayed, as shown in Figure 5–112.

Figure 5–112 The Assign Dialog - Copy Operation Tab



4. Select **Append Operation**. The Create Append Operation dialog is displayed.
5. Create an append operation to append the information stored in the temporary variable, `Variable_1`, to the reply variable, `Reply_1_InboundRequestReply_OutputVariable`, as shown in Figure 5–113.

Figure 5–113 The Create Append Operation Dialog

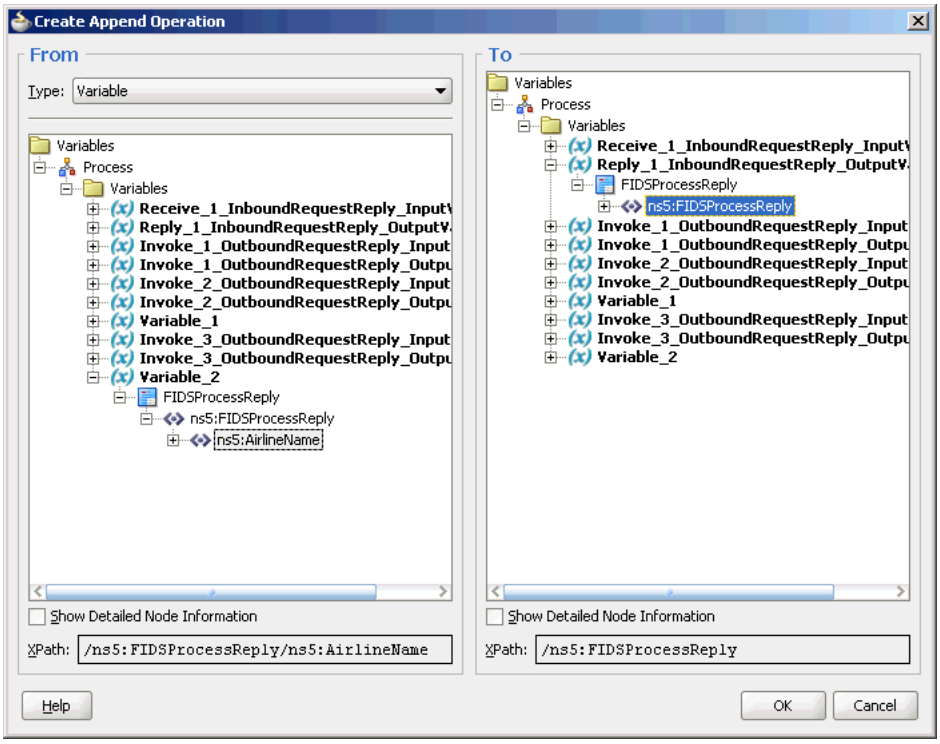


to append the information stored in the temporary variable to the reply variable.

****

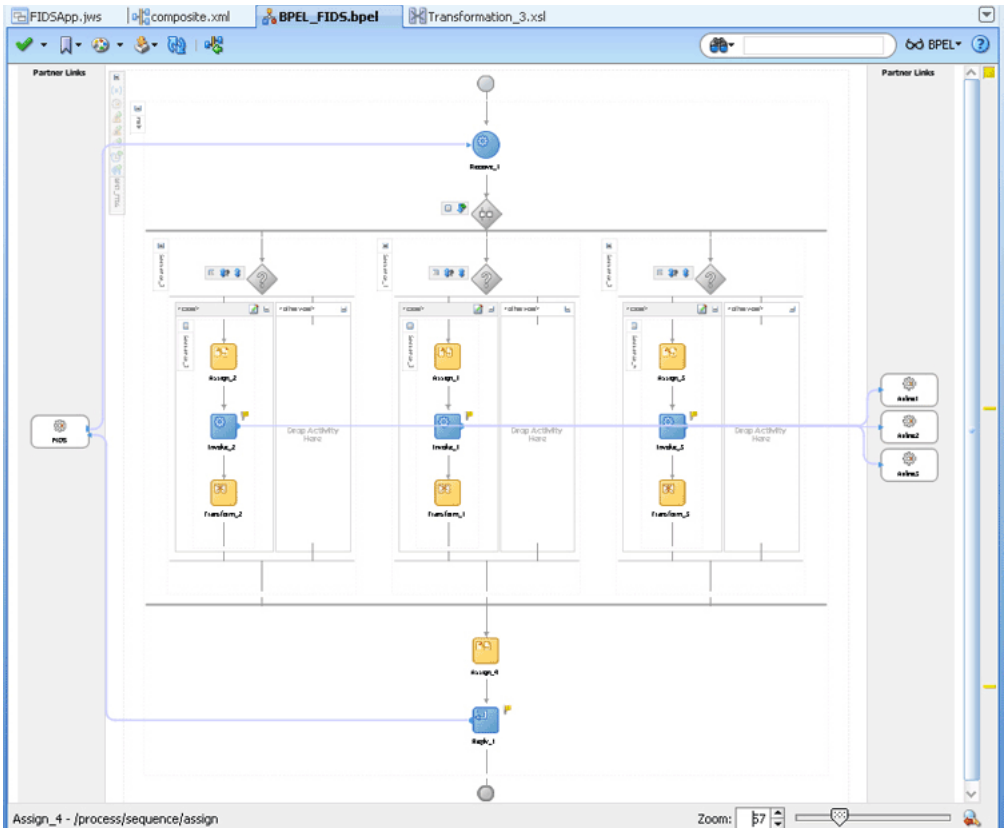
6. Click **OK**. The Assign dialog is displayed.
7. Select **Append Operation**. The Create Append Operation dialog is displayed.
8. Create another append operation to append the information stored in the temporary variable, `Variable_2`, to the reply variable, `Reply_1_InboundRequestReply_OutputVariable`, as shown in Figure 5–114.

Figure 5–114 The Create Append Operation Dialog



9. Click **OK**. The Assign dialog is displayed.
10. Click **OK**. The JDeveloper BPEL_FIDS.bpel page is displayed, as shown in Figure 5–115.

Figure 5–115 The JDeveloper - HelloWorldFlow.bpel



11. Click **File**, **Save All**.

5.5.2.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, perform the following steps:

1. Create an application server connection. For more information, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application. For more information, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

You must run the Server and Client java programs to test the application. For more information, see the associated README file.

5.5.2.7 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Click the SOA composite that you deployed. The Dashboard is displayed.

   Note your Instance ID in the Recent Instances area.
3. Click the **Instances** tab. The Instance IDs of the SOA composite are listed.
4. Click the Instance ID that you noted in Step 2. The Flow Trace page is displayed.

5. Click your BPEL process instance. The Audit Trail of the BPEL process instance is displayed.
6. Expand a payload node to view payload details.
7. Click the **Flow** tab to view the process flow. Additionally, click an activity (such as invoke, receive) to view the details of an activity.

6

Native Format Builder Wizard

This chapter describes the Native Format Builder wizard, which enables you to create native schemas used for translation. It includes use cases and constructs for the schema.

This chapter includes the following sections:

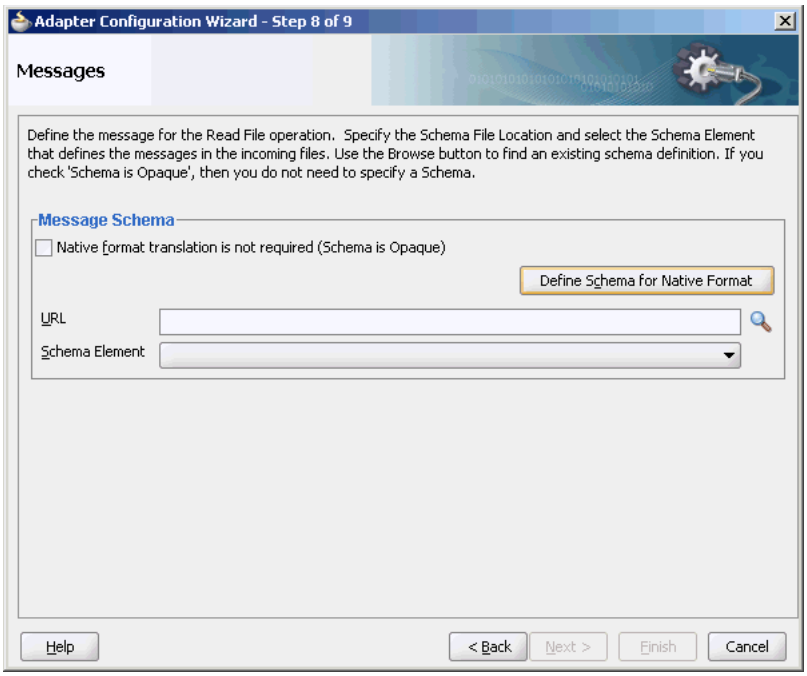
- Section 6.1, "Creating Native Schema Files with the Native Format Builder Wizard"
- Section 6.2, "Native Schema Constructs"
- Section 6.3, "Translator XPath Functions"
- Section 6.4, "Use Cases for the Native Format Builder"

6.1 Creating Native Schema Files with the Native Format Builder Wizard

Oracle JCA Adapters are software components that enable the integration between various enterprise information systems (EIS) and Oracle BPEL Process Manager (Oracle BPEL PM), or Oracle Mediator (Mediator). Adapters accept native messages in XML or non-XML format and publish them to Oracle BPEL PM or Mediator as XML messages. Adapters can also accept XML messages and convert them back to native EIS format. This translation from native data format to XML and back is performed using a definition file (non-XML schema definition), which itself is defined in XML schema format. The Native Format Builder wizard enables you to sample native data and create the native XSD (NXSD) grammar for translation of native data.

When you click the Define Schema for Native Format button in the Messages page of the Adapter Configuration Wizard shown in Figure 6–1, the Native Format Builder wizard is displayed. The Messages page is the last page that is displayed in the Adapter Configuration Wizard before the Finish page.

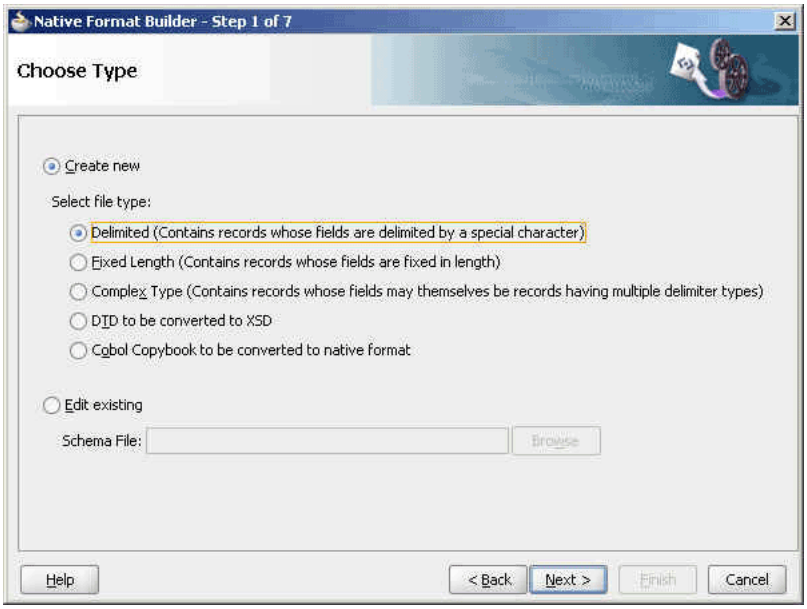
Figure 6–1 Starting the Native Format Builder Wizard



6.1.1 Supported File Formats

The Native Format Builder wizard guides you through the creation of a native schema file from the following file formats shown in Figure 6–2. You must have a sample data file format for the selected type to create a native schema. You can also select the option for editing an existing native schema created with this wizard, except for those generated from a Document Type Definition (DTD) or COBOL Copybook file types. For information on editing the native schema file, see Section 6.1.2, "Editing Native Schema Files."

Figure 6–2 Native Format Builder Wizard



6.1.1.1 Delimited

This option enables you to create native schemas for records, where the fields are separated by a value such as a comma or number sign (#).

6.1.1.2 Fixed Length (Positional)

This option enables you to create native schemas for records, where all fields are of fixed lengths.

6.1.1.3 Complex Type

This option enables you to create native schema for records, where the fields may themselves be records having multiple delimiter types.

6.1.1.4 DTD

This option enables you to generate native schema from the user-supplied DTD, which contains information about the structure of an XML document.

6.1.1.5 COBOL Copybook

This option enables you to generate native schema from the user-supplied COBOL Copybook definition.

A COBOL mainframe application typically uses a COBOL Copybook file to define its data layout. The converter creates a native schema from a COBOL Copybook so that the run-time translator can parse the associated data file.

A COBOL Copybook is typically a collection of group items (structures). These group items contain other items, which can be groups or elementary items. Elementary items are items that cannot be further subdivided. For example:

```
01 Purchase-Order
        05 Buyer
                10 BuyerName PIC X(5) USAGE DISPLAY.
        04 Seller
                08 SellerName PICTURE XXXXX.
```

`Purchase-order` is a group item with two child group items (`Buyer`, `Seller`). The numbers `01`, `05`, `04`, and so on indicate the level of the group (that is, the hierarchy of data within that group).

Groups can be defined that have different level-numbers for the same level in the hierarchy. For example, `Buyer` and `Seller` have different level numbers, but are at the same level in the hierarchy. A group item includes all group and elementary items that follow it until a level number less than or equal to the level number of that group is encountered.

Each of the group items (`Buyer` and `Seller`) has a child elementary item. The `PIC` or `PICTURE` clause defines the data layout. For example, `BuyerName` defines an alphanumeric type of size equal to five characters. `SellerName` has the same data layout as `BuyerName`.

Group items in COBOL can be mapped to elements in XML schema with the `complexType` type. Similarly, elementary items can be mapped to elements of type `simple type` with certain native format annotations to help the 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 new line character, or no delimiter, or any user-supplied string.
- Container tag name for generated native schema. By default, this is set to `Root-Element`.

COBOL Clauses

Table 6–1 describes COBOL clauses. The numeric types covered in Table 6–1 are stored as one character per digit. Support for clauses is defined as follows:

- `Y` indicates that the clause is supported.
- `N` indicates that the clause is not supported.
- `I` indicates that the clause is ignored.

Table 6–1 COBOL Clauses (Numeric Types Stored as One Character Per Digit)

COBOL Clause	Design-Time Support	Run-Time Support	Supported Synonyms	Comments
`PIC X(`*n*`)`	`Y`	`Y`	`XXX...`	Alphanumeric – An allowable character from the character set of the computer. Each X corresponds to one byte.
`PIC A(`*n*`)`	`Y`	`Y`	`AA...`	Alphabetic – Any letter of the alphabet or space. Each A corresponds to one byte.
`PIC 9(`*n*`) DISPLAY`	`Y`	`Y`	`9999...`	Any character position that contains a numeral. Each nine is counted in the size of the item.
`OCCURS` *n* `TIMES`	`Y`	`Y`		Fixed-length array
`JUSTIFIED`	`Y`	`Y`		For `A` and `X` types. Right justifies with the space pad. Data is aligned at the rightmost character position.
`REDEFINES`	`Y`	`Y`		Allows the same computer memory area to be described by different data items.

Table 6–1 (Cont.) COBOL Clauses (Numeric Types Stored as One Character Per Digit)

COBOL Clause	Design-Time Support	Run-Time Support	Supported Synonyms	Comments
PIC 9(*m*)V9(*n*) DISPLAY	Y	Y		Size = *n*+*m* bytes
OCCURS DEPENDING ON	Y	Y		NA
BLANK WHEN ZERO	I	I		Ignored
RENAMES	N	N		This is rarely seen in COBOL Copybooks
INDEX	N	N		Four-byte index
SYNCHRONIZED	I	I	SYNC	NA
POINTER	N	N		NA
PROCEDURE-POINTER				NA
FILLER	Y	Y		NA

The numeric types described in Table 6–1 are stored as one character per digit. Table 6–2 describes the numeric types that are stored in a more efficient manner.

Table 6–2 COBOL Clauses (Numeric Types Stored More Efficiently)

COBOL Clause	Design-Time Support	Run-Time Support	Supported Synonyms	Comments
USAGE [IS]	Y	Y		Both these keywords are optional.
PIC 9(*n*) COMP	Y	Y	COMPUTATIONAL, BINARY, COMP-4	Length varies with *n*: ■ *n* = 1-4 (2 bytes) ■ *n* = 5-9 (4 bytes) ■ *n* = 10-18 (8 bytes)
COMP-1	Y	Y	COMPUTATIONAL-1	Single precision, floating point number that is four bytes long.
COMP-2	Y	Y	COMPUTATIONAL-2	Double precision, floating point number that is eight bytes long.
PIC 9(*n*) COMP-3	Y	Y	PACKED-DECIMAL, COMPUTATIONAL-3	Two digits are stored in each byte. An additional half byte at the end is allocated for the sign, even if the value is unsigned.
PIC 9(*n*) COMP-4	Y	Y	COMPUTATIONAL-4	Treated the same as a COMP type and given its own data type for customizing requirements.
PIC 9(*n*) COMP-5	N	N		Capacity of the native binary representation.
PIC S9(*n*) DISPLAY	Y	Y	PIC S99…	Sign nibble in the rightmost zone by default. S is not counted in the size.

Table 6–2 (Cont.) COBOL Clauses (Numeric Types Stored More Efficiently)

COBOL Clause	Design-Time Support	Run-Time Support	Supported Synonyms	Comments
`PIC S9(n) COMP`	Y	Y		Same as `COMP`. Negative numbers are represented as two's complement.
`PIC S9(n) COMP-3`	Y	Y		NA
`PIC 9(m)V9(n) COMP`	Y	Y		Length is the same as `COMP`.
`PIC 9(m)V9(m) COMP-3`	Y	Y		Length = `Ceiling ((n+m+1)/2)`

The following clauses can be added to impact the sign position.

- `SIGN IS LEADING`

  Used with signed zoned numerics.

- `SIGN IS TRAILING`

  Used with signed zoned numerics.

- `SIGN IS LEADING SEPARATE`

  The character `S` is counted in the size.

- `SIGN IS TRAILING SEPARATE`

  The character `S` is counted in the size.

> **Note:** These assume that the numerics are stored using IBM COBOL format. If these are generated for other platforms with different data storage formats, then a custom data handler for that type must be written.

Table 6–3 describes picture editing types.

Table 6–3 Edited Pictures

Edited Pictures	Supported Editing Types	Unsupported Editing Types
Edited alphanumeric	Simple Insertion: B(blank) `0 / ,`	
Edited float numeric	Special insertion: `.` (period)	
Edited numeric	■ Simple Insertion: B(blank) `0 / ,` ■ Special insertion: `.` (period) ■ Fixed Insertion: `cs + - CR DB` (Inserts a symbol at the beginning or end)	■ Floating Insertion: `cs + -` ■ Zero suppression: `Z *` ■ Replacement insertion: `Z * + - c`

Edited pictures are more for presentation purposes and are rarely seen in data files. It is assumed that the editing symbols are also present in the data. For example, if you have:

```
05   AMOUNT     PIC 999.99
```

then, this field is six bytes wide and has a decimal point in the data.

Simple, special, and fixed insertions are handled by this method. Floating insertion, zero suppression, and replacement insertion are not supported.

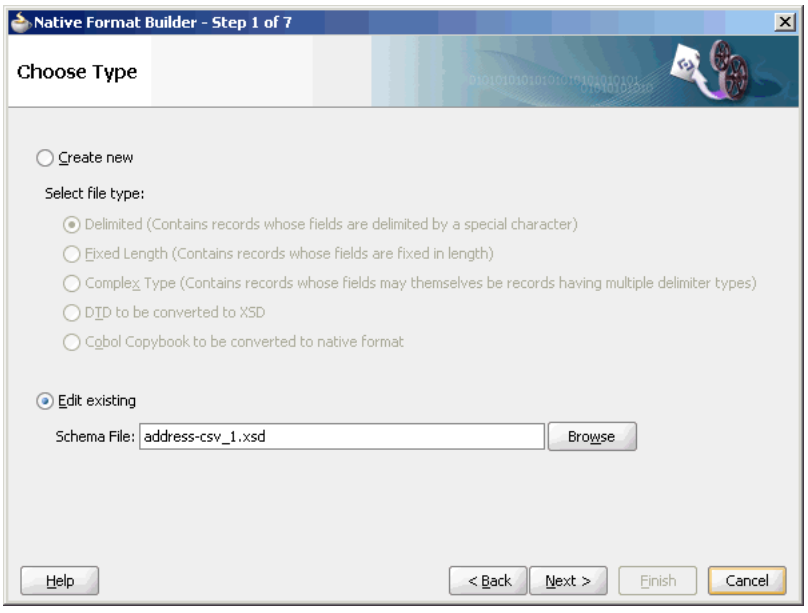
6.1.2 Editing Native Schema Files

You can edit an existing native schema generated using the Native Format Builder wizard by sampling a delimited, fixed length, or complex type file. To edit an existing native schema select the **Edit existing** option in the Choose Type page of the Native Format Builder wizard, and click **Browse** to navigate to the location of the existing schema file and then select the native schema file that must be edited. The Native Format Builder wizard guides you through the editing of the native schema file.

Note: You cannot edit native schemas generated from a Document Type Definition (DTD) or COBOL Copybook file types.

Figure 6–3 shows the Native Format Builder - Choose Type page with the Edit existing option selected.

Figure 6–3 The Native Format Builder Wizard - Choose Type Page



Before you edit a native schema file, you must ensure that the sample file specified in the annotation within the schema exists. This annotation is automatically added when the native schema is generated the first time from the sample file.

For example, if the specified sample file path in the annotation is `<!--NXSDWIZ:C:\Temp\Book1Out.csv:-->` and if the file is not located at the path specified, then the wizard displays an error.

6.2 Native Schema Constructs

This section provides an overview of the various constructs of native schema used to translate the native format data to XML and also explains the usage of these native schema constructs.

This section includes the following topics:

- Section 6.2.1, "Understanding Native Schema Constructs"
- Section 6.2.2, "Using Native Schema Constructs"

6.2.1 Understanding Native Schema Constructs

Table 6–4 shows the constructs applicable only on the `<schema>` tag.

Table 6–4 Constructs Applicable Only on the <schema> Tag

Construct	Description
`byteOrder`	The byte order of the native data as `bigEndian` or `littleEndian`.
`encoding`	The encoding in which the actual data is stored. UTF-8 is typically recommended for interoperability and Unicode support. You can specify any legal encoding supported by the Java run-time environment. For a complete listing of supported encodings, visit `http://download.oracle.com/javase/6/docs/technotes/guides/intl/encoding.doc.html`. You can specify the encoding in the (N)XSD associated with the adapter proxy meta data. For example, `nxsd:encoding="iso-8859-1`
`nxsd:alwaysQuote`	Set to true if quotes must be forced around native non-xml data in the outbound.
`headerLines`	A positive integer specifying the number of lines to be skipped, before translating the native data.
`headerLinesTerminatedBy`	Skip until the specified string, before translating the native data.
`standalone`	If declared, adds the standalone attribute in the XML declaration prolog of the translated XML, with the actual value as that specified in `nxsd:standalone`. Allowed values are `true` and `false`.
`stream`	Whether the data is stored as characters or bytes. Allowed values are `CHARS` and `BYTES`.
`uniqueMessageSeparator`	String specifying the unique message separator in the native data, in a batch of messages.
`version`	The type of native data. Possible values are `NXSD`, `DTD`, `XSD`, and `OPAQUE`.
`xmlversion`	If declared, adds the XML declaration prolog to the translated XML with the actual value as that specified in `nxsd:xmlversion`. Allowed values are `1.0` and `1.1`.
`outboundHeader`	String specifying the header value to be inserted in the outbound message.
`dataLines`	Integer specifying the number of lines to process in the native file.

Table 6–4 (Cont.) Constructs Applicable Only on the <schema> Tag

Construct	Description
fieldValidation	If set to true, then translator performs data type validation on the tokens read from the native. The fieldValidation construct is supported for built in simple types only.
validation	If set to true, then the translator performs result validation both on the inbound and outbound.
validateNxsd	If set to true, then a thorough native grammar validation is performed. This construct is switched off by default and must be switched off in production for better performance.
useArrayIdentifiers	If set to true, then it optimizes the native framework for handling array identifiers. This may result in a performance hit for very large payloads. By default, arrayIdentifiers are not supported.
parseBom	If set to true, then the byte order mark is looked for in the native stream and encoding is derived from this.
encodeLineTerminators	If set to true, then the semantic interpretation of ${eol} is \r\n instead of \n.

Table 6–5 shows the constructs applicable on all tags other than the <schema> tag.

Table 6–5 Constructs Applicable On All Tags Other Than the <schema> Tag

Construct	Description
arrayIdentifierLength	The length of the array being stored in the native data occupying the specified length
arrayLength	The value of this construct is used as the length of the array, which can also be a variable resolved to a valid number. This value overrides any minOccurs and maxOccurs attributes of the particle where it is specified. Use this feature as follows: nxsd:style="array" nxsd:arrayLength="10" This indicates that the array length is 10.
arrayTerminatedBy	The last item in the array being terminated by the specified string
assign	Assigns a value to the variable that is declared
cellSeparatedBy	The cells of the array in the native data being separated by the specified string
choiceCondition	Either fixedLength or terminated
conditionValue	Matches the string read from the native stream for the choiceCondition construct, against the specified string in the conditionValue construct
dataLines	The value specified in this construct is used to translate only a portion of the data and not the entire data.
dateFormat	A valid Java date format representing the date in the native data
identifierLength	The number of characters and bytes in which the actual length of the data is stored
itemSeparatedBy	The items in the list being separated by the specified string
leftSurroundedBy, rightSurroundedBy	The native data surrounded

Table 6–5 (Cont.) Constructs Applicable On All Tags Other Than the <schema> Tag

Construct	Description
`length`	The length of the native data to be read. Used with fixed-length style.
`listTerminatedBy`	The last item in the list being terminated by the specified string
`lookAhead`	Looks for a match ahead of the current position in the input stream. If a match is found, then the node on which this construct is specified is processed; otherwise, it is skipped. Use this feature as follows: `nxsd:lookAhead="20" nxsd:lookFor="abc"` This indicates to skip `20` characters and look for the string `abc` starting from that location. If this is found, then the node is processed; otherwise, it is skipped.
`paddedBy`	The string used for padding
`padStyle`	`head`, `tail`, or `none`
`quotedBy`	The native data being quoted by the specified string. By default, the specified string is `" (")`. If your data includes this character, you must override this default even if the field is not quoted. For more information, see "Native Data Format to Be Translated: Data Includes Default Quote Character" on page 6-16.
`skip`	Skips the specified number of bytes or characters
`skipLines`	Skips the number of lines specified
`skipUntil`	Skips until the string specified
`startsWith`	Looks for the specified string in the native data. If it exists, then proceeds with the element where it is specified; otherwise, skips and processes the next element.
`style`	The style used to read the native data from the input stream. Allowed values are `fixedLength`, `surrounded`, `terminated`, `list`, and `array`.
`surroundedBy`	The native data being surrounded by the specified string
`terminatedBy`	The native data being terminated by the string specified
`variable`	Declares a single variable
`variables`	Declares a set of variables or assigns the declared variables a valid value

6.2.2 Using Native Schema Constructs

This section includes the following topics:

- Section 6.2.2.1, "Defining Fixed-Length Data"
- Section 6.2.2.2, "Defining Terminated Data"
- Section 6.2.2.3, "Defining Surrounded Data"
- Section 6.2.2.4, "Defining Lists"
- Section 6.2.2.5, "Defining Arrays"
- Section 6.2.2.6, "Conditional Processing"
- Section 6.2.2.7, "Defining Dates"

- Section 6.2.2.8, "Using Variables"
- Section 6.2.2.9, "Defining Prefixes and Suffixes"
- Section 6.2.2.10, "Defining Skipping Data"
- Section 6.2.2.11, "Defining fixed and default Values"
- Section 6.2.2.12, "Defining write"
- Section 6.2.2.13, "Defining LookAhead"
- Section 6.2.2.14, "Defining outboundHeader"
- Section 6.2.2.15, "Defining Complex Condition in conditionValue"
- Section 6.2.2.16, "Defining Complex Condition in choiceCondition"
- Section 6.2.2.17, "Defining dataLines"
- Section 6.2.2.18, "Defining Date Formats with Time Zone"
- Section 6.2.2.19, "Implementing Validation During Translation"
- Section 6.2.2.20, "Processing Files with BOM"

6.2.2.1 Defining Fixed-Length Data

Fixed-length data in the native format can be defined in the native schema by using the fixed-length style. There are three types of fixed length:

- With padding
- Without padding
- With the actual length also being read from the native data

Native Data Format to Be Translated: With Padding

The actual data may be less than the length specified. In this case, you can specify `paddedBy` and `padStyle` as `head` or `tail`. When the data is read, the pads are trimmed accordingly. The following is a sample native data to be translated:

```
GBP*UK000012550.00
```

Native Schema: With Padding

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="fixedlength">
  <complexType>
    <sequence>
      <element name="currency_code" nxsd:style="fixedLength" nxsd:length="4"
         nxsd:padStyle="tail" nxsd:paddedBy="*">
        <simpleType>
          <restriction base="string">
            <maxLength value="4" />
          </restriction>
        </simpleType>
      </element>
```

```
      <element name="country_code" nxsd:style="fixedLength" nxsd:length="2"
         nxsd:padStyle="none">
        <simpleType>
          <restriction base="string">
            <length value="2" />
          </restriction>
        </simpleType>
      </element>
      <element name="to_usd_rate" nxsd:style="fixedLength" nxsd:length="12"
         nxsd:padStyle="head" nxsd:paddedBy="0">
        <simpleType>
          <restriction base="string">
            <maxLength value="12" />
          </restriction>
        </simpleType>
      </element>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema: With Padding

```
<fixedlength xmlns="http://www.oracle.com/ias/processconnect">
   <currency_code>GBP</currency_code>
   <country_code>UK</country_code>
   <to_usd_rate>12550.00</to_usd_rate>
</fixedlength>
```

Native Data Format to Be Translated: Without Padding

To define a fixed-length data in native schema, you can use the fixed-length style. In case the actual data is less than the length specified, the white spaces are not trimmed. The following is a sample native data to be translated:

```
GBP*UK000012550.00
```

Native Schema: Without Padding

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="fixedlength">
  <complexType>
    <sequence>
      <element name="currency_code" nxsd:style="fixedLength" nxsd:length="4">
        <simpleType>
          <restriction base="string">
            <maxLength value="4" />
          </restriction>
        </simpleType>
      </element>
```

```
      <element name="country_code" nxsd:style="fixedLength" nxsd:length="2">
        <simpleType>
          <restriction base="string">
            <length value="2" />
          </restriction>
        </simpleType>
      </element>
      <element name="to_usd_rate" nxsd:style="fixedLength" nxsd:length="12">
        <simpleType>
          <restriction base="string">
            <maxLength value="12" />
          </restriction>
        </simpleType>
      </element>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema: Without Padding

```
<fixedlength xmlns="http://www.oracle.com/ias/processconnect">
   <currency_code>GBP*</currency_code>
   <country_code>UK</country_code>
   <to_usd_rate>000012550.00</to_usd_rate>
</fixedlength>
```

Native Data Format to Be Translated: Actual Length Also Being Read from the Native Data

When the length of the data is also stored in the native stream, this style is used to first read the length, and subsequently read the data according to the length read. The following is a sample native data to be translated:

```
03joe13DUZac.1HKVmIY
```

Native Schema: Actual Length Also Being Read from the Native Data

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="fixedlength">
  <complexType>
    <sequence>
      <element name="user" type="string" nxsd:style="fixedLength"
         nxsd:identifierLength="2" />
      <element name="encr_user" type="string" nxsd:style="fixedLength"
         nxsd:identifierLength="2" />
    </sequence>
  </complexType>
</element>
```

```
</schema>
```

Translated XML Using the Native Schema: Actual Length Also Being Read from the Native Data

```
<fixedlength xmlns="http://www.oracle.com/ias/processconnect">
   <user>joe</user>
   <encr_user>DUZac.1HKVmIY</encr_user>
</fixedlength>
```

6.2.2.2 Defining Terminated Data

This format is used when the terminating mark itself is supposed to be treated as part of the actual data and not as a delimiter. When it is not clear whether the mark is part of actual data or not, you can use `nxsd:quotedBy` to be safe. Specifying `nxsd:quotedBy` means that the corresponding native data may or may not be quoted. If it is quoted, then the actual data is read from the begin quotation to the end quotation as specified in `nxsd:quotedBy`. Otherwise, it is read until the `terminatedBy` character is found.

By default, the terminating mark is `" (")`. If your data includes this character, you must override this default even if the field is not quoted. For more information, see "Native Data Format to Be Translated: Data Includes Default Quote Character" on page 6-16.

Examples for the Optionally quoted, Not quoted, and Includes default quote character scenarios are provided in the following sections:

Native Data Format to Be Translated: Optionally Quoted

The following is a sample native data to be translated:

```
Fred,"2 Old Street, Old Town,Manchester",20-08-1954,0161-499-1718
```

Native Schema: Optionally Quoted

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="terminated">
  <complexType>
    <sequence>
      <element name="PersonName" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," />
      <element name="Address" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," />
      <element name="Telephone" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
    </sequence>
  </complexType>
</element>
```

Translated XML Using the Native Schema: Optionally Quoted

```
<terminated xmlns="http://www.oracle.com/ias/processconnect">
   <PersonName>Fred</PersonName>
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>0161-499-1718</Telephone>
</terminated>
```

Native Data Format to Be Translated: Not Quoted

This is used when the data is terminated by a particular string or character. The following is a sample native data to be translated:

```
1020,16,18,,1580.00
```

Native Schema: Not Quoted

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="terminated">
  <complexType>
    <sequence>
      <element name="product" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," />
      <element name="ordered" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," />
      <element name="inventory" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="," />
      <element name="backlog" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="," />
      <element name="listprice" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema: Not Quoted

```
<terminated xmlns="http://www.oracle.com/ias/processconnect">
   <product>1020</product>
   <ordered>16</ordered>
   <inventory>18</inventory>
   <backlog></backlog>
   <listprice>1580.00</listprice>
</terminated>
```

Native Data Format to Be Translated: Data Includes Default Quote Character

The following is a sample native data to be translated:

```
aaa,"bbbbb,[cccc
```

In this case, fields are terminated by commas, the `"` character is part of the data in the second field, and the `[` character is part of the data in the third field.

Because the default `nxsd:quotedBy` terminating mark is `" (")`, the Oracle File Adapter fails to translate field two even if you specify that this field is terminated by a comma character. To successfully translate this data, you must override the default `nxsd:quotedBy` terminating mark to any character that is not be part of the data for this field. In this example, you override the default `nxsd:quotedBy` terminating mark to `< (<)` because this character never appears in field two:

```
<element name="FieldTwo" type="string" nxsd:style="terminated"
nxsd:terminatedBy="," nxsd:quotedBy="&lt;"/>
```

By contrast, for field three, you must only specify `nxsd:terminatedBy=","` because the `[` character does not conflict with the default `nxsd:quotedBy` terminating mark:

```
<element name="FieldThree" type="string" nxsd:style="terminated"
nxsd:terminatedBy="," />
```

Native Schema: Data Includes Default Quote Character

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="terminated">
  <complexType>
    <sequence>
      <element name="FieldOne" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," />
      <element name="FieldTwo" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," nxsd:quotedBy="&lt;"/>
      <element name="FieldThree" type="string" nxsd:style="terminated"
         nxsd:terminatedBy="," />
    </sequence>
  </complexType>
</element>
```

Translated XML Using the Native Schema: Data Includes Default Quote Character

```
<terminated xmlns="http://www.oracle.com/ias/processconnect">
   <FieldOne>aaa</FieldOne>
   <FieldTwo>"bbbbb</FieldTwo>
   <FieldThree>[cccc</FieldThree>
</terminated>
```

6.2.2.3 Defining Surrounded Data

This is used when the native data is surrounded by a mark.

The following are types of surrounded data:

- Left and right surrounding marks are different.
- Left and right surrounding marks are the same.

Native Data Format to Be Translated: Left and Right Surrounding Marks Are Different

The following is a sample native data to be translated for which the left and the right surrounding marks are different:

```
(Ernest Hemingway Museum){Whitehead St.}
```

Native Schema: Left and Right Surrounding Marks Are Different

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        xmlns:tns="http://www.oracle.com/ias/processconnect"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">
<element name="limstring">
  <complexType>
    <sequence>
      <element name="Landmark" type="string" nxsd:style="surrounded"
nxsd:leftSurroundedBy="(" nxsd:rightSurroundedBy=")" />
      <element name="Street" type="string" nxsd:style="surrounded"
nxsd:leftSurroundedBy="{" nxsd:rightSurroundedBy="}" />
    </sequence>
  </complexType>
</element>
</schema>
```

Translated XML Using the Native Schema: Left and Right Surrounding Marks Are Different

```
<limstring xmlns="http://www.oracle.com/ias/processconnect">
   <Landmark>Ernest Hemingway Museum</Landmark>
   <Street>Whitehead St.</Street>
</limstring>
```

Native Data Format to Be Translated: Left and Right Surrounding Marks Are the Same

The following is a sample native data to be translated for which the left and the right surrounding marks are the same:

```
.FL..Florida Keys.+Key West+
```

Native Schema: Left and Right Surrounding Marks Are the Same

```
<?xml version="1.0" encoding="US-ASCII"?>
```

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        xmlns:tns="http://www.oracle.com/ias/processconnect"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">
<element name="limstring">
  <complexType>
    <sequence>
      <element name="State" type="string" nxsd:style="surrounded"
nxsd:surroundedBy="."/>
      <element name="Region" type="string" nxsd:style="surrounded"
nxsd:surroundedBy="." />
      <element name="City" type="string" nxsd:style="surrounded"
nxsd:surroundedBy="+" />
    </sequence>
  </complexType>
</element>
</schema>
```

Translated XML Using the Native Schema: Left and Right Surrounding Marks Are the Same

```
<limstring xmlns="http://www.oracle.com/ias/processconnect">
   <State>FL</State>
   <Region>Florida Keys</Region>
   <City>Key West</City>
</limstring>
```

6.2.2.4 Defining Lists

This format applies to lists with the following characteristics:

- All Items Separated by the Same Mark, but the Last Item Terminated by a Different Mark (Bounded)
- All Items Separated by the Same Mark, Including the Last Item (Unbounded)

All Items Separated by the Same Mark, but the Last Item Terminated by a Different Mark (Bounded)

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to Be Translated:

```
125,200,255
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        xmlns:tns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
```

```
        nxsd:stream="chars"
        nxsd:version="NXSD">


<element name="list" type="tns:Colors" />


<complexType name="Colors" nxsd:style="list" nxsd:itemSeparatedBy=","
   nxsd:listTerminatedBy="${eol}">

  <sequence>
    <element name="Red" type="string" />
    <element name="Green" type="string" />
    <element name="Blue" type="string" />
  </sequence>
</complexType>

</schema>
```

Translated XML Using the Native Schema

```
<list xmlns="http://www.oracle.com/ias/processconnect">
   <Red>125</Red>
   <Green>200</Green>
   <Blue>255</Blue>
</list>
```

All Items Separated by the Same Mark, Including the Last Item (Unbounded)

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to Be Translated:

```
configure;startup;runtest;shutdown;
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        xmlns:tns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="list" type="tns:CommandSet" />

<complexType name="CommandSet" nxsd:style="list" nxsd:itemSeparatedBy=";">
  <sequence>
    <element name="Cmd1" type="string" />
    <element name="Cmd2" type="string" />
    <element name="Cmd3" type="string" />
    <element name="Cmd4" type="string" />
  </sequence>
</complexType>
```

```
</schema>
```

Translated XML Using the Native Schema:

```
<list xmlns="http://www.oracle.com/ias/processconnect">
   <Cmd1>configure</Cmd1>
   <Cmd2>startup</Cmd2>
   <Cmd3>runtest</Cmd3>
   <Cmd4>shutdown</Cmd4>
</list>
```

6.2.2.5 Defining Arrays

This is for an array of complex types where the individual cells are separated by a separating character and the last cell of the array is terminated by a terminating character.

The following are examples of array types:

- All Cells Separated by the Same Mark, but the Last Cell Terminated by a Different Mark (Bounded)
- All Cells Separated by the Same Mark, Including the Last Cell (Unbounded)
- Cells Not Separated by Any Mark, but the Last Cell Terminated by a Mark (Bounded)
- The Number of Cells Being Read from the Native Data
- Explicit Array Length

All Cells Separated by the Same Mark, but the Last Cell Terminated by a Different Mark (Bounded)

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to Be Translated:

```
"Smith, John","1 Old Street, Old Town, Manchester",,"0161-499-1717".
Fred,"2 Old Street, Old Town,Manchester","20-08-1954","0161-499-1718".
"Smith, Bob",,,0161-499-1719.#
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD"
>
<element name="array">
  <complexType>
    <sequence>
      <element name="Member" maxOccurs="unbounded"
               nxsd:style="array" nxsd:cellSeparatedBy="${eol}"
```

```
 nxsd:arrayTerminatedBy="#">
        <complexType>
          <sequence>
            <element name="Name" type="string" nxsd:style="terminated"
 nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Address" type="string" nxsd:style="terminated"
 nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="DOB" type="string" nxsd:style="terminated"
 nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Telephone" type="string" nxsd:style="terminated"
 nxsd:terminatedBy="." nxsd:quotedBy='"'/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema:

```
<array xmlns="http://www.oracle.com/ias/processconnect">
   <Member>
      <Name>Smith, John</Name>
      <Address>1 Old Street, Old Town, Manchester</Address>
      <DOB></DOB>
      <Telephone>0161-499-1717</Telephone>
   </Member>
   <Member>
      <Name>Fred</Name>
      <Address>2 Old Street, Old Town,Manchester</Address>
      <DOB>20-08-1954</DOB>
      <Telephone>0161-499-1718</Telephone>
   </Member>
   <Member>
      <Name>Smith, Bob</Name>
      <Address></Address>
      <DOB></DOB>
      <Telephone>0161-499-1719</Telephone>
   </Member>
</array>
```

All Cells Separated by the Same Mark, Including the Last Cell (Unbounded)

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to Be Translated:

```
"Smith, John","1 Old Street, Old Town, Manchester",,"0161-499-1717".
Fred,"2 Old Street, Old Town,Manchester","20-08-1954","0161-499-1718".
"Smith, Bob",,,0161-499-1719.
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
```

```
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="array">
  <complexType>
    <sequence>
      <element name="Member" maxOccurs="unbounded"
               nxsd:style="array" nxsd:cellSeparatedBy="\r\n">
        <complexType>
          <sequence>
            <element name="Name" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Address" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="DOB" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Telephone" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="." nxsd:quotedBy='"'/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>


</schema>
```

Translated XML Using the Native Schema:

```
<array xmlns="http://www.oracle.com/ias/processconnect">
   <Member>
      <Name>Smith, John</Name>
      <Address>1 Old Street, Old Town, Manchester</Address>
      <DOB></DOB>
      <Telephone>0161-499-1717</Telephone>
   </Member>
   <Member>
      <Name>Fred</Name>
      <Address>2 Old Street, Old Town,Manchester</Address>
      <DOB>20-08-1954</DOB>
      <Telephone>0161-499-1718</Telephone>
   </Member>
   <Member>
      <Name>Smith, Bob</Name>
      <Address></Address>
      <DOB></DOB>
      <Telephone>0161-499-1719</Telephone>
   </Member>
</array>
```

Cells Not Separated by Any Mark, but the Last Cell Terminated by a Mark (Bounded)

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to Be Translated:

```
"Smith, John","1 Old Street, Old Town, Manchester",,"0161-499-1717"
Fred,"2 Old Street, Old Town,Manchester","20-08-1954","0161-499-1718"
"Smith, Bob",,,0161-499-1719
#
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="array">
  <complexType>
    <sequence>
      <element name="Member" maxOccurs="unbounded"
               nxsd:style="array" nxsd:arrayTerminatedBy="#">
        <complexType>
          <sequence>
            <element name="Name" type="string" nxsd:style="terminated"
                nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Address" type="string" nxsd:style="terminated"
                nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="DOB" type="string" nxsd:style="terminated"
                nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Telephone" type="string" nxsd:style="terminated"
                nxsd:terminatedBy="\r\n" nxsd:quotedBy='"'/>          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema:

```
<array xmlns="http://www.oracle.com/ias/processconnect">
   <Member>
      <Name>Smith, John</Name>
      <Address>1 Old Street, Old Town, Manchester</Address>
      <DOB></DOB>
      <Telephone>0161-499-1717</Telephone>
   </Member>
   <Member>
      <Name>Fred</Name>
      <Address>2 Old Street, Old Town,Manchester</Address>
      <DOB>20-08-1954</DOB>
```

```
      <Telephone>0161-499-1718</Telephone>
   </Member>
   <Member>
      <Name>Smith, Bob</Name>
      <Address></Address>
      <DOB></DOB>
      <Telephone>0161-499-1719</Telephone>
   </Member>
</array>
```

The Number of Cells Being Read from the Native Data

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to Be Translated:

```
3"Smith, John","1 Old Street, Old Town, Manchester",,"0161-499-1717"
Fred,"2 Old Street, Old Town,Manchester","20-08-1954","0161-499-1718"
"Smith, Bob",,,0161-499-1719
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="arrayidentifierlength">
  <complexType>
    <sequence>
      <element name="Member" maxOccurs="unbounded" nxsd:style="array"
          nxsd:arrayIdentifierLength="1">
        <complexType>
          <sequence>
            <element name="Name" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Address" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="DOB" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy='"'/>
            <element name="Telephone" type="string" nxsd:style="terminated"
               nxsd:terminatedBy="\r\n" nxsd:quotedBy='"'/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema

```
<arrayidentifierlength xmlns="http://www.oracle.com/ias/processconnect">
   <Member>
      <Name>Smith, John</Name>
      <Address>1 Old Street, Old Town, Manchester</Address>
      <DOB></DOB>
      <Telephone>0161-499-1717</Telephone>
   </Member>
   <Member>
      <Name>Fred</Name>
      <Address>2 Old Street, Old Town,Manchester</Address>
      <DOB>20-08-1954</DOB>
      <Telephone>0161-499-1718</Telephone>
   </Member>
   <Member>
      <Name>Smith, Bob</Name>
      <Address></Address>
      <DOB></DOB>
      <Telephone>0161-499-1719</Telephone>
   </Member>
</arrayidentifierlength>
```

Explicit Array Length

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to Be Translated:

```
3;John;Steve;Paul;Todd;
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="array">
  <annotation>
    <appinfo>
      <nxsd:variables>
        <nxsd:variable name="len" />
      </nxsd:variables>
    </appinfo>
  </annotation>

  <complexType>
    <sequence>
      <element name="TotalMembers" type="string" nxsd:style="terminated"
nxsd:terminatedBy=";">
        <annotation>
          <appinfo>
            <nxsd:variables>
              <nxsd:assign name="len" value="${0}" />
            </nxsd:variables>
```

```
          </appinfo>
        </annotation>
      </element>
      <element name="Member" type="string" minOccurs="0" maxOccurs="unbounded"
             nxsd:style="array,terminated" nxsd:arrayLength="${len}"
nxsd:terminatedBy=";" />
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema:

```
<array xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <TotalMembers>3</TotalMembers>
   <Member>John</Member>
   <Member>Steve</Member>
   <Member>Paul</Member>
</array>
```

6.2.2.6 Conditional Processing

This section provides the following examples of conditional processing:

- Processing One Element Within a Choice Model Group Based on the Condition
- Processing Elements Within a Sequence Model Group Based on the Condition

Processing One Element Within a Choice Model Group Based on the Condition

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to Be Translated:

```
PO28/06/2004^|ABCD Inc.|Oracle
OracleApps025070,000.00
Database  021230,000.00
ProcessCon021040,000.00
PO01/07/2004^|EFGH Inc.|Oracle
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:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        xmlns:tns="http://www.oracle.com/ias/processconnect"
        targetNamespace="http://www.oracle.com/ias/processconnect"
```

```
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="container">

  <complexType>
    <choice maxOccurs="unbounded" nxsd:choiceCondition="fixedLength"
 nxsd:length="2">

      <element ref="tns:PurchaseOrder" nxsd:conditionValue="PO" />

      <element ref="tns:SalesOrder" nxsd:conditionValue="SO" />

      <element ref="tns:Invoice" nxsd:conditionValue="IN" />

    </choice>
  </complexType>
</element>

<!-- PO -->
<element name="PurchaseOrder" type="tns:POType"/>

<complexType name="POType">
  <sequence>

    <element name="Date" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="^" />
    <element name="Buyer" type="string" nxsd:style="surrounded"
       nxsd:surroundedBy="|" />
    <element name="Supplier" type="string" nxsd:style="terminated"
 nxsd:terminatedBy="${eol}" />
    <element name="Items">
      <complexType>
        <sequence>
          <element name="Line-Item" minOccurs="3" maxOccurs="3">
            <complexType>
              <group ref="tns:LineItems" />
            </complexType>
          </element>
        </sequence>
      </complexType>
    </element>
  </sequence>
</complexType>

<group name="LineItems">
  <sequence>
    <element name="Id" type="string" nxsd:style="fixedLength" nxsd:length="10"
        nxsd:padStyle="none"/>
    <element name="Quantity" type="string" nxsd:style="fixedLength"
        nxsd:identifierLength="2" />
    <element name="Price" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
  </sequence>
</group>

<!-- SO -->
<element name="SalesOrder" type="tns:SOType" />
```

```
<complexType name="SOType">
  <sequence>
    <element name="Date" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="|" />
    <element name="Item" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="|" />
    <element name="Quantity" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
    <element name="Buyer" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
  </sequence>
</complexType>

<!-- INV -->
<element name="Invoice" type="tns:INVType" />

<complexType name="INVType">
  <sequence>
    <element name="Date" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Purchaser" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Seller" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Item" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Price" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Quantity" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="TotalPrice" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />
  </sequence>
</complexType>

</schema>
```

Translated XML Using the Native Schema:

```
<container xmlns="http://www.oracle.com/ias/processconnect">
   <PurchaseOrder>
      <Date>28/06/2004</Date>
      <Buyer>ABCD Inc.</Buyer>
      <Supplier>Oracle</Supplier>
      <Items>
         <Line-Item>
            <Id>OracleApps</Id>
            <Quantity>50</Quantity>
            <Price>70,000.00</Price>
         </Line-Item>
         <Line-Item>
            <Id>Database  </Id>
            <Quantity>12</Quantity>
            <Price>30,000.00</Price>
         </Line-Item>
         <Line-Item>
            <Id>ProcessCon</Id>
            <Quantity>10</Quantity>
            <Price>40,000.00</Price>
```

```
      </Line-Item>
   </Items>
</PurchaseOrder>
<PurchaseOrder>
   <Date>01/07/2004</Date>
   <Buyer>EFGH Inc.</Buyer>
   <Supplier>Oracle</Supplier>
   <Items>
      <Line-Item>
         <Id>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>
```

Processing Elements Within a Sequence Model Group Based on the Condition

The following sections explain the format of the data to be translated, the native schema, and the translated XML.

Native Data Format to Be Translated:

```
PO28/06/2004^|ABCD Inc.|Oracle
OracleApps025070,000.00
Database  021230,000.00
ProcessCon021040,000.00
PO01/07/2004^|EFGH Inc.|Oracle
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:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        xmlns:tns="http://www.oracle.com/ias/processconnect"
        targetNamespace="http://www.oracle.com/ias/processconnect"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="container">

  <complexType>
    <sequence maxOccurs="unbounded">

      <element ref="tns:PurchaseOrder" minOccurs="0" nxsd:startsWith="PO" />

      <element ref="tns:SalesOrder" minOccurs="0" nxsd:startsWith="SO" />

      <element ref="tns:Invoice" minOccurs="0" nxsd:startsWith="IN" />

    </sequence>
  </complexType>
</element>

<!-- PO -->
<element name="PurchaseOrder" type="tns:POType"/>

<complexType name="POType">
  <sequence>
```

```
    <element name="Date" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="^" />    <element name="Buyer" type="string"
nxsd:style="surrounded"
       nxsd:surroundedBy="|" />
    <element name="Supplier" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />
    <element name="Items">
      <complexType>
        <sequence>
          <element name="Line-Item" minOccurs="3" maxOccurs="3">
            <complexType>
              <group ref="tns:LineItems" />
            </complexType>
          </element>
        </sequence>
      </complexType>
    </element>
  </sequence>
</complexType>

<group name="LineItems">
  <sequence>
    <element name="Id" type="string" nxsd:style="fixedLength" nxsd:length="10"
        nxsd:padStyle="none"/>
    <element name="Quantity" type="string" nxsd:style="fixedLength"
        nxsd:identifierLength="2" />
    <element name="Price" type="string" nxsd:style="terminated"
        nxsd:terminatedBy="${eol}" />
  </sequence>
</group>

<!-- SO -->
<element name="SalesOrder" type="tns:SOType" />

<complexType name="SOType">
  <sequence>
    <element name="Date" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="|" />
    <element name="Item" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="|" />
    <element name="Quantity" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />
    <element name="Buyer" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />  </sequence>
</complexType>

<!-- INV -->
<element name="Invoice" type="tns:INVType" />

<complexType name="INVType">
  <sequence>
    <element name="Date" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Purchaser" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Seller" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Item" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />    <element name="Price" type="string"
```

```
nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="Quantity" type="string" nxsd:style="terminated"
       nxsd:terminatedBy=";" />
    <element name="TotalPrice" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />
  </sequence>
</complexType>

</schema>
```

Translated XML Using the Native Schema:

```
<container xmlns="http://www.oracle.com/ias/processconnect">
   <PurchaseOrder>
      <Date>28/06/2004</Date>
      <Buyer>ABCD Inc.</Buyer>
      <Supplier>Oracle</Supplier>
      <Items>
         <Line-Item>
            <Id>OracleApps</Id>
            <Quantity>50</Quantity>
            <Price>70,000.00</Price>
         </Line-Item>
         <Line-Item>
            <Id>Database  </Id>
            <Quantity>12</Quantity>
            <Price>30,000.00</Price>
         </Line-Item>
         <Line-Item>
            <Id>ProcessCon</Id>
            <Quantity>10</Quantity>
            <Price>40,000.00</Price>
         </Line-Item>
      </Items>
   </PurchaseOrder>
   <PurchaseOrder>
      <Date>01/07/2004</Date>
      <Buyer>EFGH Inc.</Buyer>
      <Supplier>Oracle</Supplier>
      <Items>
         <Line-Item>
            <Id>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>
```

6.2.2.7 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>
```

Note: `nxsd:dateParsingMode="lax/strict"` and locale support have been added to the existing date format.

The following example depicts the use of `nxsd:dateParsingMode="lax/strict"` and locale support.

Native Data Format to Be Translated:

```
11/16/0224/11/02
11-20-2002
23*11*2002
01/02/2003 01:02
01/02/2003 03:04:05
Thu, 26 May 2005 15:50:11 India Standard Time
Do, 26 Mai 2005 15:43:10 Indische Normalzeit
20063202
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
```

```
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="dateformat">
  <complexType>
    <sequence>
      <element name="StartDate" type="date" nxsd:dateFormat="MM/dd/yy"
nxsd:localeLanguage="en" nxsd:style="fixedLength" nxsd:length="8" />
      <element name="EndDate" type="date"   nxsd:dateFormat="dd/MM/yy"
nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
      <element name="Milestone" type="dateTime" nxsd:dateFormat="MM-dd-yyyy"
nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
      <element name="DueDate" type="dateTime"   nxsd:dateFormat="dd*MM*yyyy"
nxsd:style="terminated" nxsd:terminatedBy="${eol}" />

      <element name="Date" type="dateTime"   nxsd:dateFormat="MM/dd/yyyy hh:mm"
nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
      <element name="Date" type="dateTime"   nxsd:dateFormat="MM/dd/yyyy hh:mm:ss"
nxsd:style="terminated" nxsd:terminatedBy="${eol}" />

      <element name="LongDateInEnglish" type="dateTime"   nxsd:dateFormat="EEE, d
MMM yyyy HH:mm:ss zzzz" nxsd:localeLanguage="en" nxsd:localeCountry="US"
nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
      <element name="LongDateInGerman" type="dateTime"   nxsd:dateFormat="EEE, d
MMM yyyy HH:mm:ss zzzz" nxsd:localeLanguage="de" nxsd:style="terminated"
nxsd:terminatedBy="${eol}" />

      <element name="InvalidDate" type="dateTime"   nxsd:dateParsingMode="lax"
nxsd:dateFormat="yyyyMMdd" nxsd:style="terminated" nxsd:terminatedBy="${eol}" />

    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML:

```
<dateformat xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <StartDate>2002-11-16</StartDate>
   <EndDate>2002-11-24</EndDate>
   <Milestone>2002-11-20T00:00:00</Milestone>
   <DueDate>2002-11-23T00:00:00</DueDate>
   <Date>2003-01-02T01:02:00</Date>
   <Date>2003-01-02T03:04:05</Date>
   <LongDateInEnglish>2005-05-26T15:50:11</LongDateInEnglish>
   <LongDateInGerman>2005-05-26T15:43:10</LongDateInGerman>
   <InvalidDate>2008-08-02T00:00:00</InvalidDate>
</dateformat>
```

6.2.2.8 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>
```

6.2.2.9 Defining Prefixes and Suffixes

In native format, when data is read, the specified data is prefixed, suffixed, or both, as shown in the following example.

Native Data to Be Translated:

```
     Fred,  "2 Old Street, Old Town,Manchester","20-08-1954",0161-499-1718
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD"
        >

<element name="terminated">
  <complexType>
    <sequence>
      <element name="PersonName" type="string" nxsd:prefixWith="Mr."
       nxsd:style="terminated" nxsd:terminatedBy="," nxsd:quotedBy="&quot;" />
      <element name="Address" type="string" nxsd:suffixWith="]]"
       nxsd:prefixWith="[[" nxsd:style="terminated" nxsd:terminatedBy=","
       nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using the Native Schema:

```
<terminated xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Mr.Fred</PersonName>
   <Address>[[2 Old Street, Old Town,Manchester]]</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>0161-499-1718</Telephone>
</terminated>
```

6.2.2.10 Defining Skipping Data

Translator skips, before or after the data is read, depending on the `skipMode` construct, as shown in the following example:

Native Data to Be Translated:

```
     Fred,   "2 Old Street, Old Town,Manchester","20-08-1954",0161-499-1718
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD"
        >

<element name="terminated">
  <complexType>
    <sequence>
      <element name="PersonName" type="string" nxsd:skip="5"
       nxsd:style="terminated" nxsd:terminatedBy="," nxsd:quotedBy="&quot;" />
      <element name="Address" type="string" nxsd:skipMode="before" nxsd:skip="3"
       nxsd:style="terminated" nxsd:terminatedBy="," nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:skipMode="after" nxsd:skip="6"
       nxsd:style="terminated" nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using Native Schema:

```
<terminated xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Fred</PersonName>
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>99-1718</Telephone>
</terminated>
```

6.2.2.11 Defining fixed and default Values

When an element is declared without `nxsd` annotations but the value specified is either `fixed` or `default`, the translator uses the value provided and does not throw any exceptions.

Native Data to Be Translated:

```
Fred,"2 Old Street, Old Town,Manchester","20-08-1954","0161-499-1718"
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="terminated">
  <annotation>
    <appinfo>
      <nxsd:variables>
        <nxsd:variable name="x" value="hello" />
      </nxsd:variables>
      <junkies/>
      <junkies/>
      <junkies/>
    </appinfo>
  </annotation>

  <complexType>
    <sequence>
      <element name="PersonName" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy="&quot;" />
      <element name="Age" type="string" fixed="16"  />
      <element name="Address" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="salutation" type="string" default="${x}"  />
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using Native Schema:

```
<terminated xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Fred</PersonName>
   <Age>16</Age>
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
   <salutation>hello</salutation>
   <Telephone>0161-499-1718</Telephone>
```

```
</terminated>
```

6.2.2.12 Defining write

The `write` construct writes the literal at the current position in the output stream, either before writing the actual data or after writing it.

Input XML:

```
<terminated xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Fred</PersonName>
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>0161-499-1718</Telephone>
</terminated>
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD"
        >

<element name="terminated">
  <complexType>
    <sequence>
      <element name="PersonName" type="string" nxsd:writeMode="before"
       nxsd:write="Mr." nxsd:style="terminated" nxsd:terminatedBy=","
       nxsd:quotedBy="&quot;" />
      <element name="Address" type="string" nxsd:writeMode="after"
       nxsd:write="Over." nxsd:style="terminated" nxsd:terminatedBy=","
       nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated Data Using Native Schema:

```
Mr.Fred,"2 Old Street, Old Town,Manchester",Over.20-08-1954,0161-499-1718
```

6.2.2.13 Defining LookAhead

The `LookAhead` construct is of the following types:

- Type 1: `LookAhead` X chars, read the value from a position using a style, and match against the specified literal.

- Type 2: `LookAhead` X chars, read the value from a position using a style, and store that value in a variable to be used later.

LookAhead: Type 1

`LookAhead` X chars, read the value from a position using a style, and match against the specified literal.

Native Data Format to Be Translated:

```
Fred,"2 Old Street, Old Town,Manchester","20-08-1954","0161-499-1718",YES
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="LookAhead">
  <complexType>
    <sequence minOccurs="0" nxsd:lookAhead="70" nxsd:lookFor="YES">
      <element name="PersonName" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy="&quot;" />
      <element name="Address" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using Native Schema:

```
<LookAhead xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Fred</PersonName>
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>0161-499-1718</Telephone>
</LookAhead>
```

LookAhead: Type 2

In native schema, `LookAhead` X chars, read the value from a position using a style, and store that value in a variable to be used later.

Native Data Format to Be Translated:

```
Name1,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", YES
Name2,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", NO
Name3,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", NO
```

```
Name4,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", YES
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<!--
nxsd:lookAhead="70" nxsd:scan="3"
-->

<element name="LookAhead">
  <complexType>
    <choice maxOccurs="unbounded" nxsd:choiceCondition="${x}" nxsd:lookAhead="70"
       nxsd:scanLength="3" nxsd:assignTo="${x}">
      <element name="Record1" type="string" nxsd:conditionValue="YES"
       nxsd:style="terminated" nxsd:terminatedBy="," nxsd:skipMode="after"
       nxsd:skipUntil="${eol}" />
      <element name="Record2" type="string" nxsd:conditionValue="NO "
       nxsd:style="terminated" nxsd:terminatedBy="," nxsd:skipMode="after"
       nxsd:skipUntil="${eol}" />
    </choice>
  </complexType>
</element>

</schema>
```

Translated XML Using Native Schema:

```
<LookAhead xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <Record1>Name1</Record1>
   <Record2>Name2</Record2>
   <Record2>Name3</Record2>
   <Record1>Name4</Record1>
</LookAhead>
```

6.2.2.14 Defining outboundHeader

The actual content of `outboundHeader` can use variables, specifically `${eol}`. When `headerLines` and `outboundHeader` both are available, `outboundHeader` takes precedence in the outbound.

> **Note:** In the inbound direction, the Skipping Headers feature is supported. Only predefined variables can be used in a header because other variables might either not be accessible or would have only literals.

Input XML:

```
<terminated xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Fred</PersonName>
```

```
   <Address>2 Old Street, Old Town,Manchester</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>0161-499-1718</Telephone>
</terminated>
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD"
        nxsd:hasHeader="true"
        nxsd:outboundHeader="This is a header ${eol}">

<element name="terminated">
  <complexType>
    <sequence>
      <element name="PersonName" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy="&quot;" />
      <element name="Address" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy="&quot;"/>
      <element name="DOB" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," nxsd:quotedBy='"'/>
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" nxsd:quotedBy='"'/>
    </sequence>
  </complexType>
</element>

</schema>
```

Translated Data:

```
This is a header
Fred,"2 Old Street, Old Town,Manchester",20-08-1954,0161-499-1718
```

6.2.2.15 Defining Complex Condition in conditionValue

When you use the `conditionValue` construct along with the `choiceCondition` construct, you can specify match criteria such as equals (==) and not equals (!=), along with the Boolean operators AND and OR, for comparison between the value read and the value specified in the `conditionValue` construct.

Native Data Format to Be Translated:

```
Order, ID41678, 20May2000
Item1, GigaWidget, 60, $75
Item2, MegaBucket, 48, $125
Cust1, Hopkins Associates, ID26490
Order, ID41680, 20May2000
Item3, Rt.Clopper, 40, $100
Item4, Lt.Clopper, 50, $100
Cust2, Jersey WebInovaters, ID46786
```

Native Schema:

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/extensions/SampleNS"
            xmlns:tns="http://xmlns.oracle.com/pcbpel/nxsd/extensions/SampleNS"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified" nxsd:encoding="US-ASCII"
nxsd:stream="chars" nxsd:version="NXSD">

<xsd:element name="Container">
    <xsd:complexType>
      <xsd:choice minOccurs="1" maxOccurs="unbounded"
       nxsd:choiceCondition="terminated" nxsd:terminatedBy=",">
        <xsd:element name="Customer" nxsd:conditionValue="(== Cust1) or (== Cust2)
         and (!= emp)">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="C1" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="C2" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="${eol}" nxsd:quotedBy="&quot;">
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Item" nxsd:conditionValue="(== Item1) or (== Item2) or
         (==Item3) or (== Item4)">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="C1" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="C2" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="C3" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="${eol}" nxsd:quotedBy="&quot;">
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Order" nxsd:conditionValue="Order">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="C1" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="C2" type="xsd:string" nxsd:style="terminated"
               nxsd:terminatedBy="${eol}" nxsd:quotedBy="&quot;">
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Translated XML Using Native Schema:

```
<Container xmlns="http://xmlns.oracle.com/pcbpel/nxsd/extensions/SampleNS">
   <Order>
      <C1> ID41678</C1>
      <C2> 20May2000</C2>
   </Order>
   <Item>
      <C1> GigaWidget</C1>
      <C2> 60</C2>
      <C3> $75</C3>
   </Item>
   <Item>
      <C1> MegaBucket</C1>
      <C2> 48</C2>
      <C3> $125</C3>
   </Item>
   <Customer>
      <C1> Hopkins Associates</C1>
      <C2> ID26490</C2>
   </Customer>
   <Order>
      <C1> ID41680</C1>
      <C2> 20May2000</C2>
   </Order>
   <Item>
      <C1> Rt.Clopper</C1>
      <C2> 40</C2>
      <C3> $100</C3>
   </Item>
   <Item>
      <C1> Lt.Clopper</C1>
      <C2> 50</C2>
      <C3> $100</C3>
   </Item>
   <Customer>
      <C1> Jersey WebInovaters</C1>
      <C2> ID46786</C2>
   </Customer>
</Container>
```

6.2.2.16 Defining Complex Condition in choiceCondition

The `choiceCondition` construct is used along with the conditionValue construct for records that are complex and may have fields delimited by multiple delimiter types. The other choiceCondition types available are `FixedLength`, `Variable`, and `Ad hoc`. The following example is for the variable choiceCondition type.

Native Data Format to Be Translated:

```
Name1,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", YES
Name2,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", NO
```

Native Schema:

```
<element name="LookAhead">
  <complexType>
    <choice maxOccurs="unbounded" nxsd:choiceCondition="${x}" nxsd:lookAhead="70"
nxsd:scanLength="3" nxsd:assignTo="${x}">
      <element name="Record1" type="string" nxsd:conditionValue="YES"
```

```
nxsd:style="terminated" nxsd:terminatedBy="," nxsd:skipMode="after"
nxsd:skipUntil="${eol}" />
      <element name="Record2" type="string" nxsd:conditionValue="NO "
nxsd:style="terminated" nxsd:terminatedBy="," nxsd:skipMode="after"
nxsd:skipUntil="${eol}" />
    </choice>
  </complexType>
</element>
```

Translated XML Using Native Schema:

```
<LookAhead xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <Record1>Name1</Record1>
   <Record2>Name2</Record2>
</LookAhead>
```

6.2.2.17 Defining dataLines

If the requirement is to translate only a portion of the data and not the entire data, then you can specify the number of lines to be ignored from the beginning of the file and the number of lines to be translated from that point onwards by using the `dataLines` construct.

Native Data Format to Be Translated:

```
Fred,addr,20-08-1954,0161-499-1718
Tam,addr,20-08-1954,0161-499-1718
Albert,addr,20-08-1954,0161-499-1718
Bill,addr,20-08-1954,0161-499-1718
Phil,addr,20-08-1954,0161-499-1718
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD"

        nxsd:headerLines="1"
        nxsd:dataLines="1">

<element name="terminated">
  <complexType>
    <sequence maxOccurs="unbounded">
      <element name="PersonName" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," />
      <element name="Address" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," />
      <element name="DOB" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="," />
      <element name="Telephone" type="string" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />
    </sequence>
  </complexType>
```

```
</element>

</schema>
```

Translated XML Using Native Schema:

```
<terminated xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <PersonName>Tam</PersonName>
   <Address>addr</Address>
   <DOB>20-08-1954</DOB>
   <Telephone>0161-499-1718</Telephone>
</terminated>
```

6.2.2.18 Defining Date Formats with Time Zone

In the translator, the date or time must be associated with a time zone. The translator supports the date formats with time zone for both, the date in native data and for the date in XML.

There are two parts when translating a date/time string. First, the format of the date in the native data (`dateformat`), second is the time zone to use when parsing that date (`timeZone` or `useTimeZone`). The translator uses these details while parsing the date/time string.

After the parsing, by default, the date string is converted to the ISO-8601 format in an XML. You can override the defaults by using `XMLDateFormat` and `XMLTimeZone`, or `useTimeZone`.

Native Data Format to Be Translated:

```
11/16/0224/11/02
11-20-2002
23*11*2002
01/02/2003 01:02
01/02/2003 03:04:05
Thu, 26 May 2005 15:50:11 India Standard Time
Do, 26 Mai 2005 15:43:10 Indische Normalzeit
20063202
11/16/02
```

Native Schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        nxsd:stream="chars"
        nxsd:version="NXSD">

<element name="dateformat">
  <complexType>
    <sequence>
      <element name="StartDate" type="date" nxsd:dateFormat="MM/dd/yy"
       nxsd:localeLanguage="en" nxsd:style="fixedLength" nxsd:length="8" />

      <element name="EndDate" type="date"   nxsd:dateFormat="dd/MM/yy"
       nxsd:style="terminated" nxsd:terminatedBy="${eol}" />
```

```
      <element name="Milestone" type="dateTime" nxsd:useTimeZone="UTC"
       nxsd:dateFormat="MM-dd-yyyy" nxsd:style="terminated"
nxsd:terminatedBy="${eol}" />

      <element name="DueDate" type="dateTime" nxsd:useTimeZone="UTC"
       nxsd:dateFormat="dd*MM*yyyy" nxsd:style="terminated"
nxsd:terminatedBy="${eol}" />

      <element name="Date" type="dateTime" nxsd:useTimeZone="UTC"
       nxsd:dateFormat="MM/dd/yyyy hh:mm" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />

      <element name="Date" type="dateTime" nxsd:useTimeZone="UTC"
       nxsd:dateFormat="MM/dd/yyyy hh:mm:ss" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />

      <element name="LongDateInEnglish" type="dateTime"
       nxsd:displayTimeZone="true" nxsd:useTimeZone="IST" nxsd:dateFormat="EEE, d
MMM
       yyyy HH:mm:ss zzzz" nxsd:localeLanguage="en" nxsd:localeCountry="US"
       nxsd:style="terminated" nxsd:terminatedBy="${eol}" />

      <element name="LongDateInGerman"  type="dateTime"
       nxsd:displayTimeZone="true" nxsd:useTimeZone="IST" nxsd:dateFormat="EEE, d
MMM
       yyyy HH:mm:ss zzzz" nxsd:localeLanguage="de" nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />

      <element name="InvalidDate" type="dateTime" nxsd:useTimeZone="UTC"
       nxsd:dateParsingMode="lax" nxsd:dateFormat="yyyyMMdd"
nxsd:style="terminated"
       nxsd:terminatedBy="${eol}" />

      <element name="MyFormatDate" type="string" nxsd:dateFormat="MM/dd/yy"
       nxsd:xmlDateFormat="dd-MM-yyyy" nxsd:localeLanguage="en"
nxsd:style="fixedLength"
       nxsd:length="8" />

    </sequence>
  </complexType>
</element>

</schema>
```

Translated XML Using Native Schema:

```
<dateformat xmlns="http://xmlns.oracle.com/pcbpel/nxsd/smoketest">
   <StartDate>2002-11-16</StartDate>
   <EndDate>2002-11-24</EndDate>
   <Milestone>2002-11-20T00:00:00</Milestone>
   <DueDate>2002-11-23T00:00:00</DueDate>
   <Date>2003-01-02T01:02:00</Date>
   <Date>2003-01-02T03:04:05</Date>
   <LongDateInEnglish>2005-05-26T15:50:11+05:30</LongDateInEnglish>
   <LongDateInGerman>2005-05-26T15:43:10+05:30</LongDateInGerman>
   <InvalidDate>2008-08-02T00:00:00</InvalidDate>
   <MyFormatDate>16-11-2002</ MyFormatDate >
</dateformat>
```

6.2.2.19 Implementing Validation During Translation

You must configure Oracle JCA Adapters to implement validation during translation. Validation helps ensure that Oracle JCA Adapters do not publish invalid messages during translation.

You can implement either one or both of the following types of validation:

- Payload Validation
- Schema Validation

6.2.2.19.1 Payload Validation Payload validation involves validating the input and output XML messages that are processed by Oracle JCA Adapters. You can set payload validation at one of the these levels:

- Top-Level Validation
- Field-Level Validation

Top-Level Validation

In top-level validation, the DOMResult (result in the form of a Document Object Model) is validated against the XML schema. This form of validation is implemented on both inbound and outbound payloads. This form of validation can control the publishing of invalid records and provide information about XML validation errors. However, it does not provide translation context. For example, information about the line and column in the native stream where the error was encountered is not provided by top-level validation.

To implement top-level validation of XML messages:

- The nxsd namespace in the message must be set to the following:

  ```
  xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
  ```

- The validation flag must be set to `true` as follows:

  ```
  nxsd:validation="true"
  ```

For example:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
     xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
         attributeFormDefault="unqualified"
        nxsd:stream="chars"
         nxsd:version="NXSD"
        nxsd:validation="true"
        >
```

Field-Level Validation

In field-level validation, the individual fields are validated against the XML schema. This form of validation is implemented only on inbound payloads, not on outbound payloads.

If the XML message does not conform to the XML schema, then information about the exact line and character where the error was encountered is displayed.

To implement field-level validation of XML messages:

- The nxsd namespace in the message must be set to the following:

```
xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
```

- The validation flag must be set to `true` as follows:

```
nxsd:fieldValidation="true"
```

For example:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
elementFormDefault="qualified"
attributeFormDefault="unqualified"
 nxsd:stream="chars"
  nxsd:version="NXSD"
   nxsd:fieldValidation="true"
   >
```

6.2.2.19.2 Schema Validation Schema validation involves validating the schema (native schemas or XML schemas) that you define for the native or XML data formats to be translated by the Oracle JCA Adapters.

To enable schema validation:

- The nxsd namespace in the message must be set to the following:

```
xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
```

- The validate nxsd flag must be set to `true` as follows:

```
nxsd:validateNxsd="true"
```

For example:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
     xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/smoketest"
        elementFormDefault="qualified"
         attributeFormDefault="unqualified"
        nxsd:stream="chars"
         nxsd:version="NXSD"
        nxsd: validateNxsd ="true"
        >
```

Note: The `nxsd:validateNxsd="true"` validation flag *does not* affect payload level validations.

6.2.2.20 Processing Files with BOM

The byte order mark (BOM) is a special U+FEFF Unicode character that describes the encoding of a byte sequence. The Native Format Translator can be configured to use BOM for determining the character encoding of the native input data. By default, BOM is not used. If your input data uses BOM, then set the `nxsd:parseBom` attribute to `true` in the native schema. Otherwise, the translator throws a parsing error.

The following is a sample nxsd file:

```
<?xml version= '1.0' encoding= 'UTF-8' ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
```

```
            xmlns:tns="http://TargetNamespace.com/InboundService"
            targetNamespace="http://TargetNamespace.com/InboundService"
            elementFormDefault="qualified" attributeFormDefault="unqualified"
            nxsd:parseBom="true" nxsd:version="NXSD" nxsd:stream="chars"
nxsd:encoding="UTF8">
    <xsd:element name="Root-Element">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="mydata" minOccurs="1" maxOccurs="unbounded"
                    nxsd:style="array"
                    nxsd:cellSeparatedBy="${eol}">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="C1" type="xsd:string"
                               nxsd:style="fixedLength"
                               nxsd:length="3"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

6.3 Translator XPath Functions

The translator XPath functions can translate data from a native format (such as CSV, fixed-length, tab-delimited, and COBOL Copybook formats) to an XML format and from an XML format to a native format. The translator XPath functions are of two types, streaming and non-streaming. The difference is that the streaming translator XPath functions implement the batching transformation approach while the non-streaming XPath functions do not implement the batching transformation approach. With the batching transformation approach, files that are of the order of a few gigabytes (GB) can be processed without running into memory issues.

This section includes the following topics:

- Section 6.3.1, "Terminologies"
- Section 6.3.2, "Translator XPath Functions"

6.3.1 Terminologies

This section describes the terminologies that you must understand for using translation XPath functions.

Attachment Element

An attachment element unusually refers to the actual content elsewhere by using an `"href"` attribute. The actual content may be present in a file system or in a database table. An attachment is usually represented by using the following schema construct:

```
<element name="hrefelement">
    <complexType>
        <attribute name="href" type="string"/>
    </complexType>
</element>
```

The `"href"` attribute contains the actual location of the data being referred to. It can contain the path to a file in the file system or a pointer (primary key) to a database entity.

Scalable DOM

Scalable DOM (SDOM), from Oracle XML Developer Kit (Oracle XDK), provides scalable and pluggable support for DOM. This removes problems of memory inefficiency, limited scalability, and lack of control over the DOM configuration. Using the lazy materialization mechanism, Oracle XDK only creates nodes that are accessed and frees unused nodes from memory. Applications can process very large XML documents with improved scalability.

6.3.2 Translator XPath Functions

A translator may be required while reading and writing files. This section discusses the following translator XPath functions:

- doTranslateFromNative Function
- doTranslateToNative Function
- doStreamingTranslate Function

6.3.2.1 doTranslateFromNative Function

The doTranslateFromNative XPath function translates input data into XML. The input data can be a string, an attachment element, or a base64Binary element.

```
ora:dotranslateFromNative('input','nxsdTemplate','nxsdRoot','targetType','attachment element?')
```

The following table describes the parameters used in the syntax for using this function:

Parameter	Description
`input`	Input data for the XPath function; the data can either be a string data that must be translated, an Oracle File or FTP Adapter attachment, an attachment referring to an external file path, or a base64Binary element.
nxsdTemplate	NXSD schema to use to translate the input data into XML format.
nxsdRoot	Root element in the NXSD schema.
targetType	This parameter decides how the XPath function translates the native data into XML. Must be set to either `'DOM'`, or `'ATTACHMENT'` or `'SDOM'`. If the `targetType` parameter is: ■ `'DOM'`, then the translated data is returned as a DOM. ■ `'ATTACHMENT'`, then the translated data is returned as an attachment. If the optional parameter (`attachmentElement`) is available to the XPath function, then the XPath function uses the corresponding `href` attribute to write the translated XML. However, if the parameter is absent, then the XPath function creates a new database-backed attachment and returns that. See Example 6–4 for more details. ■ `'SDOM'`, then the translated data is returned as SDOM. You must use this if the returned XML file is huge.
attachmentElement	This parameter is optional. This is the attachment for the returned XML file.

Example 6–1 Configuring the XPath Function When the Input Data Is of String Type and Must Be Converted to an XML DOM

```
<variables>
     <variable…/>
    <variable name="csv_data" type="xsd:string"/>
  </variables>
<assign name="assignCSVData">
      <copy>
        <from expression="'this, is, csv, data…'"/>
        <to variable="csv_data"/>
      </copy>
</assign>

<assign name="doTranslateFromNativeCall">
      <copy>
        <from expression="ora: doTranslateFromNative (bpws:getVariableData('csv_
data'),'xsd/address-csv.xsd','Root-Element','DOM')"/>
        <to variable="returnVariable" query="/ns1:Root-Element"/>
      </copy>
    </assign>
```

In this example:

- `csv_data` is a BPEL variable containing CSV data to be translated into XML.
- `xsd/address-csv.xsd` is relative path to the NXSD schema in the project.
- `Root-Element` is a root element in the NXSD schema (This is optional.)
- `returnVariable` is the returned XML data as DOM.

Example 6–2 When the Input data is an Attachment, Which Must Be Translated to a DOM

1. Define `attachmentElement` in the schema of the BPEL process, as follows:

```
<schema targerNamespace="…">
        <element name="attachmentElement">
            <complexType>
                <attribute name="href" type="string"/>
            </complexType>
        </element>
</schema>
```

2. Create a variable of type attachment element in the schema of the BPEL process, as follows:

```
  <variables>
     <variable…/>
    <variable name="attachmentVariable" type="client:attachmentElement"/>
  </variables>
```

3. Assign the source file path that you must translate, as follows:

```
<assign name="AssignAttachmentReference">
  <copy>
    <from expression="'/tmp/xpath/in/address.csv'"/>
    <to variable="attachmentVariable"
            query="/client:attachmentElement/@href"/>
  </copy>
</assign>
```

4. Call the XPath function, as follows:

```
<assign name="xlateFromNative">
  <copy>
    <from
expression="ora:doTranslateFromNative(bpws:getVariableData('attachmentVariable'
),'xsd/address-csv.xsd', 'Root-Element', 'DOM')"/>
    <to variable="returnVariable" query="/ns1:Root-Element"/>
  </copy>
</assign>
```

In this example:

- `attachmentVariable` is an attachment variable in BPEL referring to the source file path.
- `xsd/address-csv.xsd` is the relative path to the NXSD schema in the project.
- `Root-Element` is a root element in the NXSD schema.
- `returnVariable` is the XML data returned as DOM.

Example 6–3 Configuring XPath Function When the Input Data Is Base64-encoded and Must Be Translated to DOM

1. Define the base64-encoded element in the schema of the BPEL process, as follows:

```
<schema targerNamespace="…">
  <element name="mtomElement" type="base64Binary"/>
</schema>
```

2. Create a variable of type mtom element in the schema of the BPEL process, as follows:

```
<variables>
     <variable…/>
    <variable name="encodedData" type="client:mtomElement"/>
</variables>
```

3. Assign the source file path that you must translate, as follows

```
<assign name="assignBase64EncodedData">
      <copy>
        <from expression="'b3JhY2xl'"/>
        <to variable="encodedData" query="/client:mtomElement"/>
      </copy>
</assign>

<assign name="doTranslateFromNativeCall">
      <copy>
        <from expression="ora: doTranslateFromNative
(bpws:getVariableData('encodedData'),'xsd/address-csv.xsd','Root-Element','DOM'
)"/>
        <to variable="returnVariable" query="/ns1:Root-Element"/>
      </copy>
    </assign>
```

In this example:

- `mtomElement` is a BPEL variable containing base64-encoded data to be translated into XML.
- `xsd/address-csv.xsd` is the relative path to the NXSD schema in the project.
- `Root-Element` is a root element in the NXSD schema.

- `returnVariable` is the XML data returned as DOM.

Example 6–4 Configuring XPath Function When the Input Data Is of String Type, Which Must Be Translated to an Attachment Referred to by a File-Path

1. Define `attachmentElement` in the schema of the BPEL process, as follows:

```
<schema targerNamespace="…">
        <element name="attachmentElement">
            <complexType>
                <attribute name="href" type="string"/>
            </complexType>
        </element>
</schema>
```

2. Create an input variable of type string and an output variable of type attachment in the schema of the BPEL process, as follows:

```
<variables>
    <variable…/>
    <variable name="csv_data" type="xsd:string"/>
   <variable name="returnAttachmentVariable" type="client:attachmentElement"/>
</variables>
```

3. Assign the CSV data that you must translate, as follows:

```
<assign name="assignCSVData">
    <copy>
      <from expression="'this, is, csv, data…'"/>
      <to variable="csv_data"/>
    </copy>
</assign>
```

4. Populate the attachment with the path of the file where you want the translated data to be stored, as follows:

```
<assign name="AssignAttachmentReferenceForOutput">
    <copy>
      <from expression="'/tmp/xpath/output/address.xml'"/>
      <to variable=" returnAttachmentVariable "
            query="/client:attachmentElement/@href"/>
    </copy>
</assign>
```

5. Call the XPath function as follows:

```
<assign name="doTranslateFromNativeCall">
    <copy>
      <from expression="ora: doTranslateFromNative (bpws:getVariableData('csv_
data'),'xsd/address-csv.xsd','Root-Element','ATTACHMENT',
bpws:getVariableData('returnAttachmentVariable'))"/>
      <to variable="returnAttachmentVariable"/>
    </copy>
</assign>
```

In this example:

- `csv_data` is a BPEL string variable containing CSV data to be translated into XML.
- `xsd/address-csv.xsd` is the relative path to the NXSD schema in the project.
- `Root-Element` is a root element in the NXSD schema.

- `returnAttachmentVariable` is the returned attachment.

Note: In this example, `targetType` is set to `ATTACHMENT`, and `returnAttachmentVariable` points to the file path where the translated XML is to be written.

However, the fifth parameter (`bpws:getVariableData('returnAttachmentVariable')`) is optional. If this parameter is missing, then the XPath function creates a database-backed attachment and returns it. In such a case, the XPath function is configured, as follows:

1. Define `attachmentElement` in the schema of the BPEL process, as follows:

```
<schema targerNamespace="…">
        <element name="attachmentElement">
            <complexType>
                <attribute name="href" type="string"/>
            </complexType>
        </element>
</schema>
```

2. Create an input variable of type string and an output variable of type attachment in the schema of the BPEL process, as follows:

```
<variables>
    <variable…/>
    <variable name="csv_data" type="xsd:string"/>
   <variable name="returnAttachmentVariable" type="client:attachmentElement"/>
</variables>
```

3. Assign the CSV data that you must translate, as follows:

```
<assign name="assignCSVData">
    <copy>
      <from expression="'this, is, csv, data…'"/>
      <to variable="csv_data"/>
    </copy>
</assign>

<assign name="doTranslateFromNativeCall">
      <copy>
        <from expression="ora: doTranslateFromNative
(bpws:getVariableData('csv_
data'),'xsd/address-csv.xsd','Root-Element','ATTACHMENT')"/>
        <to variable="returnAttachmentVariable"/>
      </copy>
    </assign>
```

After the XPath call returns, the `returnAttachmentVariable` variable is populated with the `href` attribute pointing to the GUID representing the database-backed attachment.

Note: If the data being translated is huge, then you must use either `ATTACHMENT` or `SDOM` as the targetType parameter for the XPath function.

6.3.2.2 doTranslateToNative Function

The `doTranslateToNative` XPath function translates an input DOM into string data or an attachment.

Syntax:

`ora:dotranslateToNative('input','nxsdTemplate','nxsdRoot','targetType','attachmentElement?')`

The following table describes the parameters used in the syntax for using this function:

Parameter	Description
`input`	Input data for the XPath function; the data can either be DOM or SDOM data that must be translated to a native format such as CSV.
`nxsdTemplate`	NXSD schema to be used to translate the input data into XML format.
`nxsdRoot`	Name of the root element in the NXSD schema.
`targetType`	This parameter decides how the XPath function translates the XML data into native formats. Must be set to either `'STRING'`, or `'ATTACHMENT'`. If the `targetType` parameter is: ■ `STRING`, then the translated data is returned as a string. ■ `ATTACHMENT`, then the translated data is returned as an attachment. If the optional parameter (`attachmentElement`) is available to the XPath function, then the XPath function uses the corresponding `href` attribute to write the translated native data. However, if the parameter is absent, then the XPath function creates a new database-backed attachment and returns that. See Example 6–6 for more details.
`attachmentElement`	This parameter is optional. This is the attachment to which the translated data is written.

Example 6–5 Configuring the XPath Function When the Input Data Is of XML Format and Must Be Translated Into CSV String Format

```
<variables>
     <variable…/>
   <variable name="inputDOM" type="ns1:Root-Element"/> <!- - data that must be
translated into native - ->
    <variable name="returned_csv_data" type="xsd:string"/>
  </variables>

<assign name="doTranslateToNativeCall">
      <copy>
        <from expression="ora: doTranslateToNative
(bpws:getVariableData('inputDOM'),'xsd/address-csv.xsd','Root-Element','STRING')"/
>
        <to variable="returned_csv_data"/>
      </copy>
    </assign>
```

In this example:

- `inputDOM` is a BPEL DOM variable containing XML data to be translated into string data representing the translated CSV.
- `xsd/address-csv.xsd` is the relative path to the NXSD schema in the project.
- `Root-Element` is a root element in the NXSD schema.
- `return_csv_data` is the string variable that contains the translated CSV data.

Example 6–6 Configuring XPath Function to Translate an Incoming XML DOM into an Attachment Representing the Target File-Path for the Translated CSV

1. Define `attachmentElement` in the schema of the BPEL process, as follows:

```
<schema targerNamespace="…">
        <element name="attachmentElement">
            <complexType>
                <attribute name="href" type="string"/>
            </complexType>
        </element>
</schema>
```

2. Create an input variable of type `attachmentElement` in the schema of the BPEL process, as follows:

```
<variables>
     <variable…/>
    <variable name="inputDOM" type="ns1:Root-Element"/>
    <variable name="attachmentVariable" type="client:attachmentElement"/>
  </variables>
```

3. Assign the target file path where you want the translated CSV to be written, as follows:

```
<assign name="AssignAttachmentReference">
  <copy>
     <from expression="'/tmp/xpath/out/address.csv'"/>
     <to variable="attachmentVariable"
            query="/client:attachmentElement/@href"/>
  </copy>
</assign>
```

4. Call the XPath function, as follows:

```
<assign name="xlateToNative">
      <copy>
        <from
expression="ora:doTranslateToNative(bpws:getVariableData('inputDOM'),'xsd/addre
ss-csv.xsd', 'Root-Element', 'ATTACHMENT', bpws:getVariableData('
attachmentVariable'))"/>
        <to variable="attachmentVariable"/>
      </copy>
    </assign>
```

In this example:

- `inputDOM` is a BPEL DOM variable containing XML data to be translated into a CSV output file represented by `/tmp/xpath/out/address.csv`.
- `xsd/address-csv.xsd` is the relative path to the NXSD schema in the project.
- `Root-Element` is a root element in the NXSD schema.
- `AttachmentElement` points to the target output file path represented by `/tmp/xpath/out/address.csv`.

Note: In this example, `targetType` is set to `ATTACHMENT`, and `AttachmentVariable` points to the file path where the translated CSV file is to be written.

However, the fifth parameter (`bpws:getVariableData('attachmentVariable')`) is optional. If this

parameter is missing, then the XPath function creates a database-backed attachment and returns it. In such a case, the XPath function is configured as follows:

1. Define `attachmentElement` in the schema of the BPEL process, as follows:

```
<schema targerNamespace="…">
        <element name="attachmentElement">
            <complexType>
                <attribute name="href" type="string"/>
            </complexType>
        </element>
</schema>
```

2. Create an input variable of type `attachmentElement` in the schema of the BPEL process, as follows:

```
<variables>
     <variable…/>
    <variable name="inputDOM" type="ns1:Root-Element"/>
    <variable name="attachmentVariable" type="client:attachmentElement"/>
  </variables>
```

3. Call the XPath function, as follows:

```
<assign name="xlateToNative">
      <copy>
        <from
expression="ora:doTranslateToNative(bpws:getVariableData('inputDOM'),'xsd/addre
ss-csv.xsd', 'Root-Element', 'ATTACHMENT')"/>
        <to variable="attachmentVariable"/>
      </copy>
    </assign>
```

After the XPath call returns, `attachmentVariable` is populated with the `href` attribute pointing to the GUID representing the database-backed attachment.

6.3.2.3 doStreamingTranslate Function

XPath functions implement the batching transformation approach. With this approach, files that are of the order of a few gigabytes (GB) can be processed without running into memory issues. Arbitrarily large payloads can be handled because the transformation engine does not store the result of the transformation in its memory. The transformation engine flushes its memory after a batch of elements of the large file is processed. The default batch size is 10000, which is the number of elements after which the transformation engine flushes its memory. This parameter is used internally and is optional.

> **Note:** Batching transformation approach is supported for XML documents that have repeating structures only.

Syntax:

`ora:doStreamingTranslate('`*`input`*`','`*`streamingXpathContext`*`','`*`targetType`*`','`*`attachmentElement`*`?')`

The following table describes the parameters used in the syntax for using this function:

Parameter	Description
input	Input data for the XPath function; the data can either be SDOM or an Attachment element.
streamingXpathContext	DOM representing the XPath context.
targetType	This parameter decides how the XPath function translates the input data into an attachment. This must be set to either SDOM or ATTACHMENT.
attachmentElement	This parameter is optional. This is the attachment to which the data is streamed.

The streamingXPathContext parameter specifies the context for the streaming transformation and, it must conform to the following schema element:

```
<schema  targetNamespace="…">
    <element name="streamingcontext">
        complexType>
            <sequence>
                <element name="sourceSchema" type="string"/>
                <element name="sourceRootElement" type="string"/>
                <element name="sourceType" type="string"/>
                <element name="xsl" type="string"/>
                <element name="targetSchema" type="string"/>
                <element name="targetRootElement" type="string"/>
                <element name="targetType" type="string"/>
                <element name="batchSize" type="string"/>
            </sequence>
        </complexType>
    </element>
</schema>
```

In context:

Schema Element	Description
sourceSchema	Source NXSD schema used to translate a native data to XML.
sourceRootElement	Name of root element in source NXSD schema.
sourceType	Set this to either xml or native depending on the input data.
xsl	Relative path of the XSL file.
targetSchema	Target NXSD schema used to translate an XML into native data.
targetRootElement	Name of root element in target NXSD schema.
targetType	Set this to either xml or native depending on the output data.
batchSize	The number of elements after which the transformation engine flushes its memory.

6.3.2.4 Batching Transformation Features

This section discusses the following features of batching transformation:

Applicability

Batching transformation is applicable to:

- Documents with repeating structure
- XSLTs not requiring aggregation across entire document

Batched Invocation of XSLT Engine

The following procedure highlights the batched invocation of the XSLT engine:

1. Splitting the source document into multiple batches of one or more records
2. Performing the XSLT transformation one batch at a time
3. Combining the result of the XSLT invocation to a single target document

Splitting or Combining Performed on the Fly

The source documents are split and the results are combined into a target document:

- Without any intermediate memory or disk storage
- Through pipelining or intercepting SAX events

Low In-Memory Footprint

Batching transformation method uses low memory for the following tasks:

- Transforming arbitrarily large XML documents, which are constrained by the target system
- For standalone tests, 540 MB is transformed in less than 3 minutes

Example 6–7 implements the FlatStructure FileAdapter sample using streaming transformation XPath functions. This sample use case translates the inbound native attachment from a CSV format to an XML format, and then applies the user-supplied XSL file to the resulting XML file. The transformed XML file is then translated into a fixed-length content represented by an attachment.

Example 6–7 Using Streaming Transformation XPath Function

1. Define attachmentElement, as shown:

```
<schema targerNamespace="…">
    <element name="attachmentElement">
        <complexType>
           <attribute name="href" type="string"/>
        </complexType>
    </element>
</schema>
```

2. Create a variable for the input attachment referring to the inbound csv file and the output attachment referring to the output fixed-length file. Create the variable corresponding to the streaming context. You must populate this variable before making a call to the XPath function.

```
  <variables>
    <variable name="xlationContext" element="client:streamingcontext"/>
    <variable name="inputAttachment" element="client:attachmentElement"/>
    <variable name="returnAttachment" element="client:attachmentElement"/>
  </variables>

 <!- - Assign the input and output attachments - ->
    <assign name="assignValuesForAttachments">
      <copy>
        <from expression="'/tmp/xpath/in/address.csv' "/>
        <to variable="inputAttachment"
```

```
query="/client:attachmentElement/@href"/>
      </copy>
      <copy>
        <from expression="'/tmp/xpath/out/address_fixedLength.txt' "/>
        <to variable="returnAttachment"
            query="/client:attachmentElement/@href"/>
      </copy>
    </assign>
<!- - Assign the streaming context - ->
    <assign name="AssignStreamingContext">
      <copy>
        <from expression="'xsd/address-csv.xsd'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/client:sourceSchema"/>
      </copy>
      <copy>
        <from expression="'Root-Element'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/client:sourceRootElement"/>
      </copy>
      <copy>
        <from expression="'native'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/client:sourceType"/>
      </copy>
      <copy>
        <from expression="'xsd/address-fixedLength.xsd'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/client:targetSchema"/>
      </copy>
      <copy>
        <from expression="'Root-Element'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/client:targetRootElement"/>
      </copy>
      <copy>
        <from expression="'native'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/client:targetType"/>
      </copy>
      <copy>
        <from expression="'xsl/addr1Toaddr2.xsl'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/client:xsl"/>
      </copy>
      <copy>
        <from expression="'10000'"/>
        <to variable="xlationContext"
            query="/client:streamingcontext/client:batchSize"/>
      </copy>
    </assign>
<!- - call the XPath function - ->
    <assign name="executeStreamingXPath">
      <copy>
        <from
expression="ora:doStreamingTranslate(bpws:getVariableData('inputAttachment','/c
lient:attachmentElement'),
bpws:getVariableData('xlationContext'), 'ATTACHMENT',
bpws:getVariableData('returnAttachment'))"/>
        <to variable="returnAttachment" query="/client:attachmentElement"/>
```

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

6.4 Use Cases for the Native Format Builder

This section describes the following use cases:

- Section 6.4.1, "Defining the Schema for a Delimited File Structure"
- Section 6.4.2, "Defining the Schema for a Fixed Length File Structure"
- Section 6.4.3, "Defining the Schema for a Complex File Structure"
- Section 6.4.4, "Removing or Adding Namespaces to XML with No Namespace"
- Section 6.4.5, "Defining the Choice Condition Schema for a Complex File Structure"
- Section 6.4.6, "Defining Choice Condition With LookAhead for a Complex File Structure"
- Section 6.4.7, "Defining Array Type Schema for a Complex File Structure"
- Section 6.4.8, "Defining the Schema for a DTD File Structure"
- Section 6.4.9, "Defining the Schema for a COBOL Copybook File Structure"

Note: Sampling the data with multi-character delimiter in Native Format Builder is not supported currently. The same can be achieved through hand coding the NXSD with the appropriate Delimited By string.

6.4.1 Defining the Schema for a Delimited File Structure

A comma-separated value (CSV) file is a common non-XML file structure.

Use the **Delimited** option in the Native Format Builder wizard, when creating the XML schema for this native file.

The `nxsd:headerLines="1"` schema attribute signifies that the first line must be treated as a header row and skipped in the native data before actually translating the rest of the data. The `nxsd:stream="chars"` schema attribute signifies that the data should be read as characters. If `nxsd:stream` is set as `bytes`, `nxsd:stream="bytes"`, then this schema attribute signifies that the native data should be read as bytes. For each of the element declarations, `Name`, `Street`, `City`, `State`, and `Country`, which have a corresponding scalar data, the `nxsd:style="terminated"` attribute defines that the corresponding data is stored in terminated style. The actual terminator is then defined by the `nxsd:terminatedBy=","` attribute specified at that construct. See Section 6.2.2.2, "Defining Terminated Data" for details on the terminated style.

In this use case, the Native Format Builder uses a delimited sample file type that contains the address details, such as name, street, city, state, and country. Every element in this sample native file is delimited by a comma (,). You can generate the corresponding NXSD and also test it. Perform the following steps to run the use case:

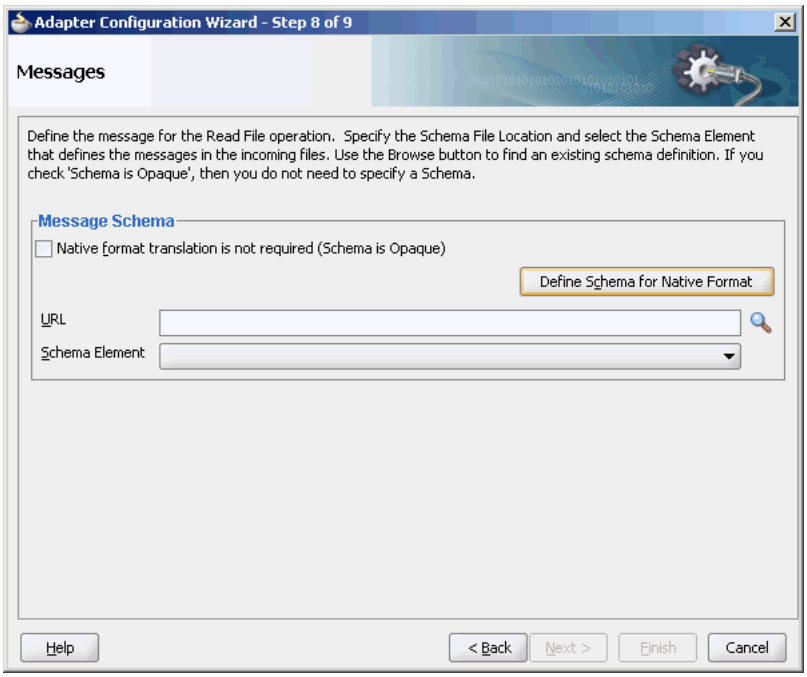
1. The data in a sample text file, `address-csv.txt`, appears as below:

```
Name,Street1,Street2,City,State,Country
Oracle India Private Limited, Lexington Towers Prestige St. John's Woods, 2nd
Cross Road Chikka Audugodi, Bangalore, Karnataka, India
```

```
Intel Technology India Private Limited, Survey #23-56 P Devarabeesanahalli
Village, Outer Ring Road Varthur Hobli, Bangalore, Karnataka, India
```

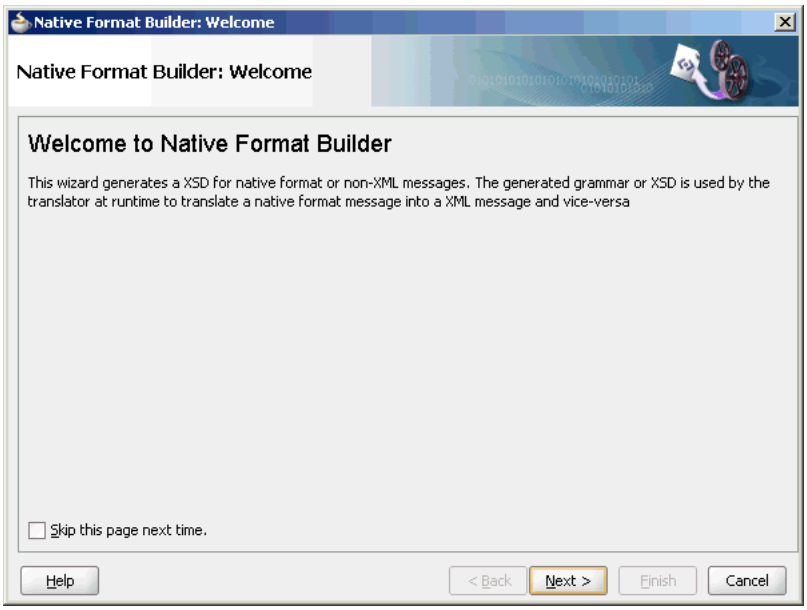
2. Navigate to the Adapter Configuration Wizard Messages page, as displayed in Figure 6–4, and click the **Define Schema for Native Format** button.

Figure 6–4 Starting the Native Format Builder Wizard



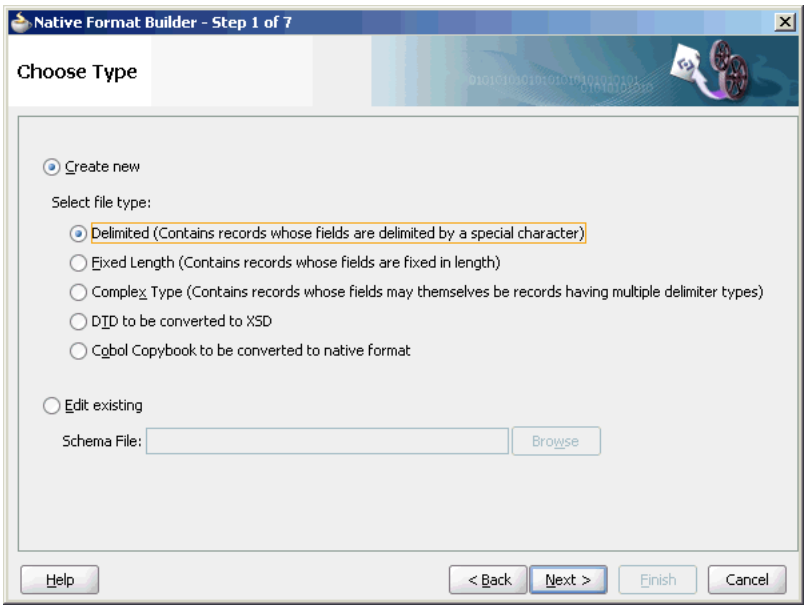
The Native Format Builder Welcome page is displayed, as shown in Figure 6–5.

Figure 6–5 Native Format Builder Wizard Welcome Page



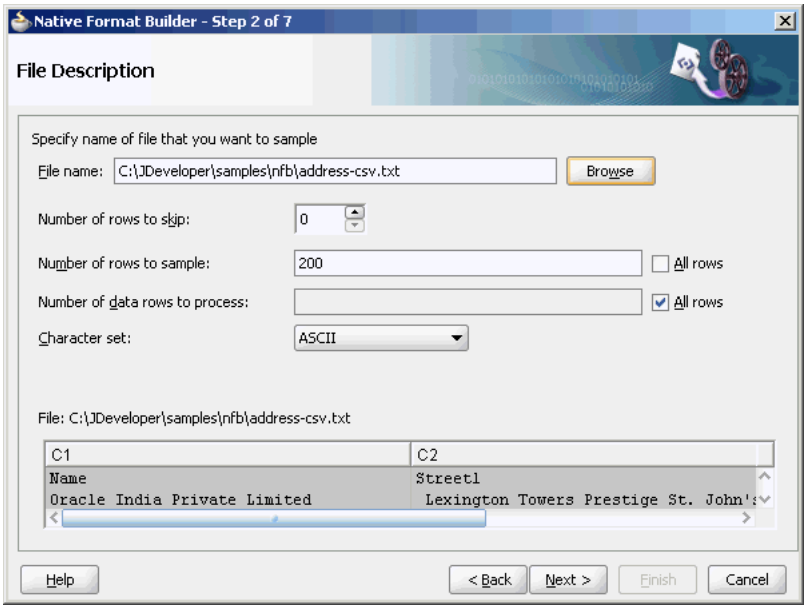
3. Click **Next**. The Choose Type page is displayed, as shown in Figure 6–6.

Figure 6–6 Native Format Builder Wizard Choose Type Page



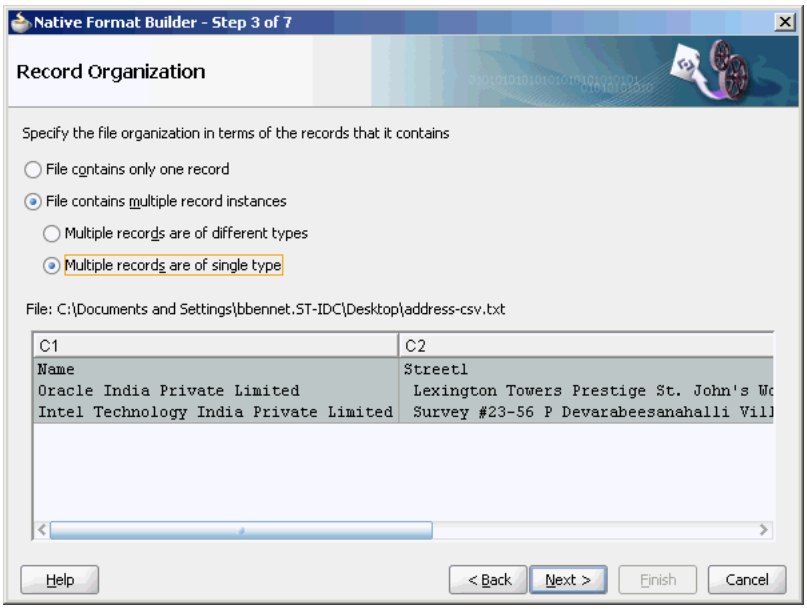
4. Click **Next**. The Native Format Builder File Description page is displayed.
5. Click **Browse** and select the `address-csv.txt` file, as shown in Figure 6–7.

Figure 6–7 Native Format Builder Wizard File Description Page



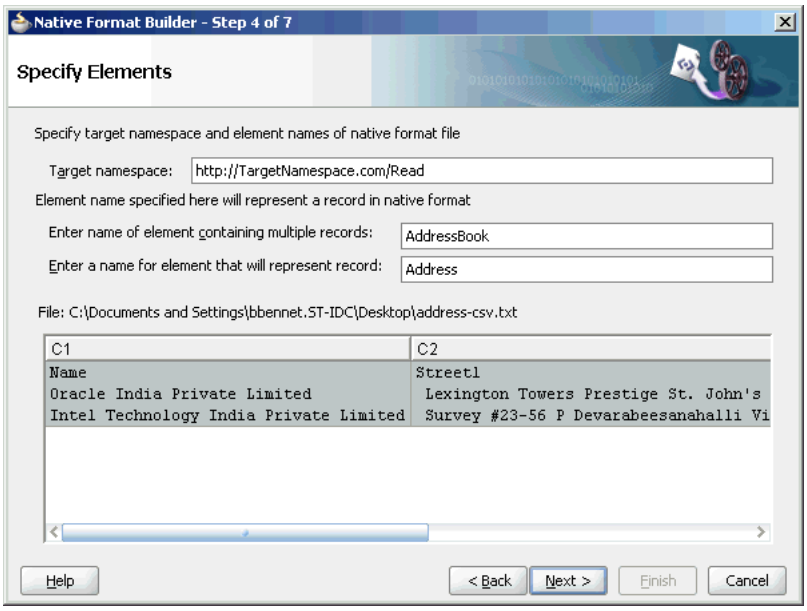
6. Click **Next**. The Record Organization page is displayed, as shown in Figure 6–8.

Figure 6–8 Native Format Builder Wizard Record Organization Page



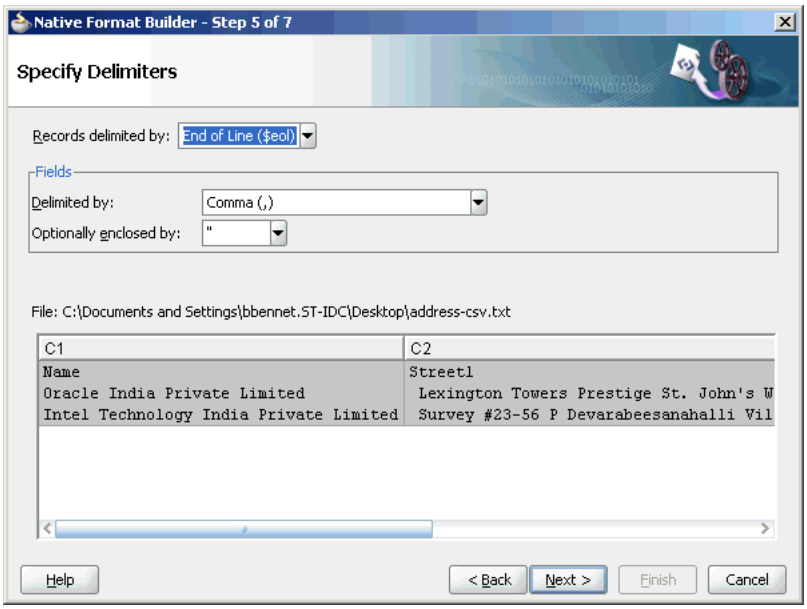
7. Select **File contains multiple record instances**, then select **Multiple records are of single type**, and then click **Next**. The Specify Elements page is displayed.
8. Enter `AddressBook` in the **Enter name of element containing multiple records** field and enter `Address` in the **Enter a name for element that will represent record** field, as shown in Figure 6–9.

Figure 6–9 Native Format Builder Wizard Specify Elements Page



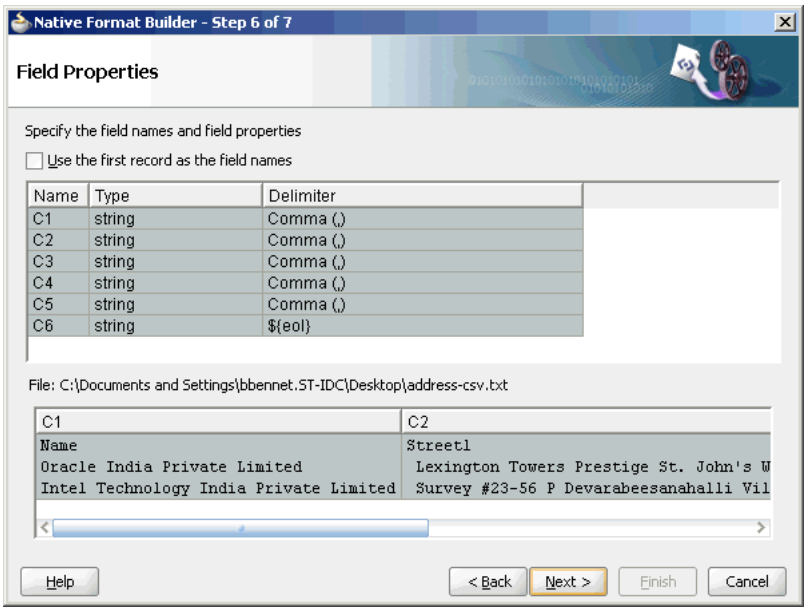
9. Click **Next**. The Specify Delimiters page is displayed, as shown in Figure 6–10.

Figure 6–10 Native Format Builder Wizard Specify Delimiters Page



10. Ensure that the **Comma(,)** option is selected in the Delimited By field, and click **Next**. The Field Properties page is displayed, as shown in Figure 6–11.

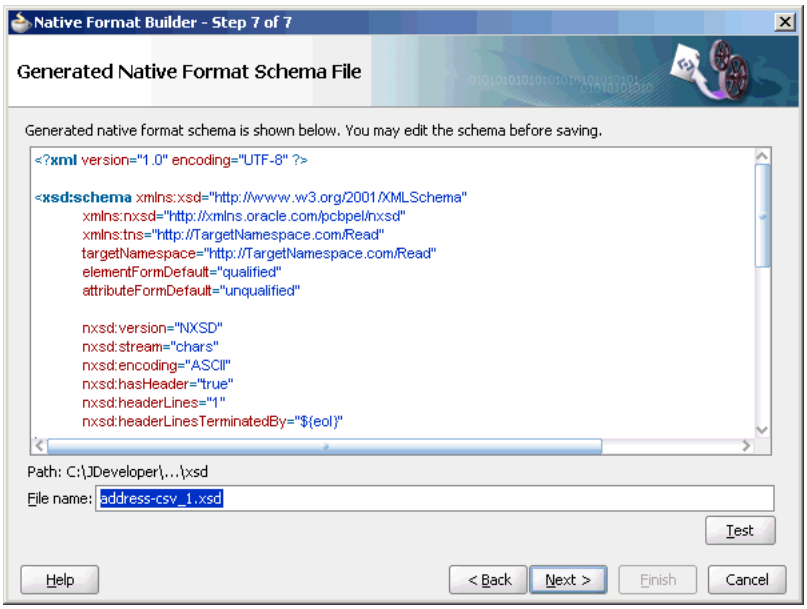
Figure 6–11 Native Format Builder Wizard Field Properties Page



11. Select **Use the first record as the field names**, then click **Next**. The Generated Native Format File page is displayed, as shown in Figure 6–12.

Note: The first record is used as the field name, is also treated as a header record, and is skipped during translation.

Figure 6–12 Native Format Builder Wizard Generated Native Format File Page



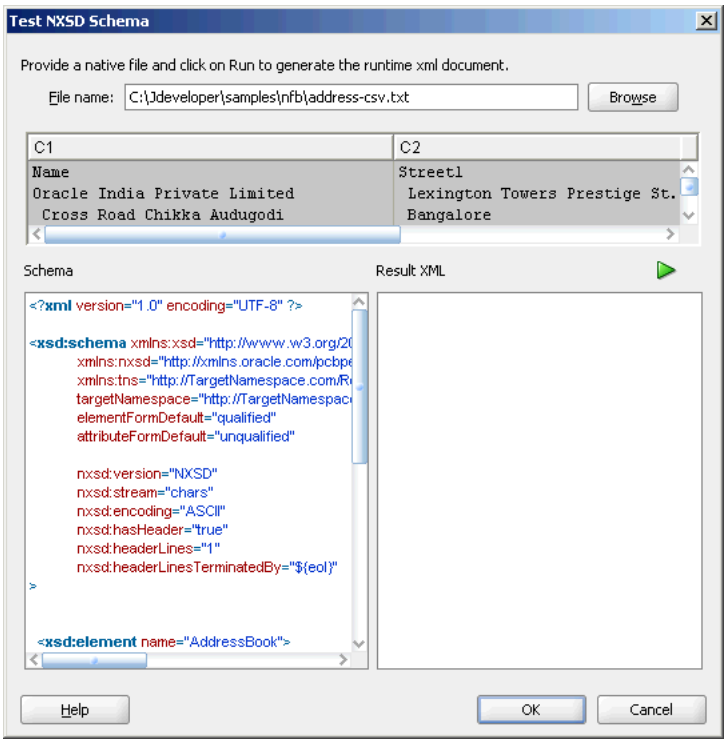
The corresponding native schema definition is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            xmlns:tns="http://TargetNamespace.com/Read"
            targetNamespace="http://TargetNamespace.com/Read"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            nxsd:version="NXSD"
            nxsd:stream="chars"
            nxsd:encoding="ASCII"
            nxsd:hasHeader="true"
            nxsd:headerLines="1"
            nxsd:headerLinesTerminatedBy="${eol}"
>
  <xsd:element name="AddressBook">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Address" minOccurs="1" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Name" type="xsd:string"
               nxsd:style="terminated" nxsd:terminatedBy=","
nxsd:quotedBy="&quot;" />
              <xsd:element name="Street1" type="xsd:string"
               nxsd:style="terminated" nxsd:terminatedBy=","
nxsd:quotedBy="&quot;" />
              <xsd:element name="Street2" type="xsd:string"
               nxsd:style="terminated" nxsd:terminatedBy=","
nxsd:quotedBy="&quot;" />
              <xsd:element name="City" type="xsd:string"
               nxsd:style="terminated" nxsd:terminatedBy=","
nxsd:quotedBy="&quot;" />
              <xsd:element name="State" type="xsd:string"
               nxsd:style="terminated" nxsd:terminatedBy=","
nxsd:quotedBy="&quot;" />
```

```
                <xsd:element name="Country" type="xsd:string"
                 nxsd:style="terminated" nxsd:terminatedBy="${eol}"
nxsd:quotedBy="&quot;" />
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

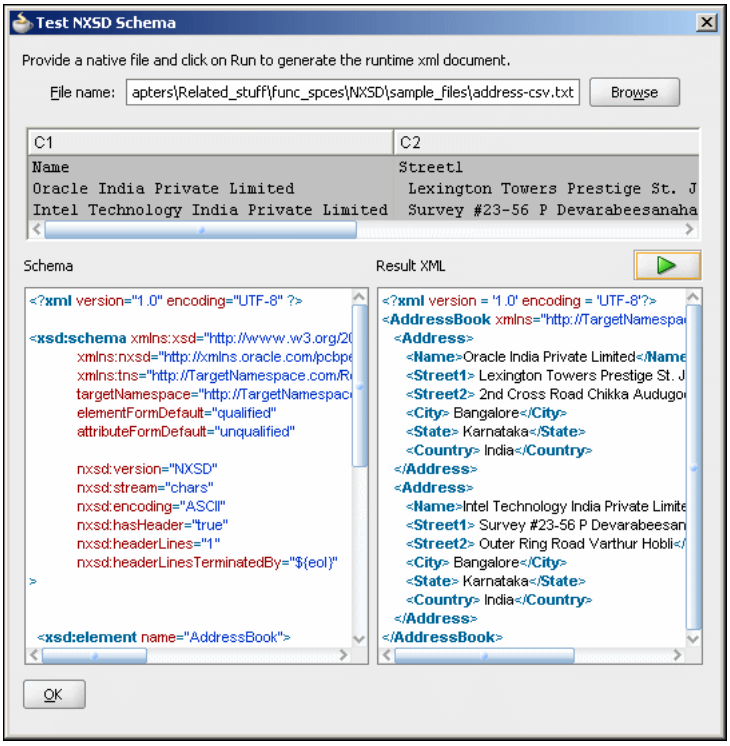
12. Click **Test**. The Test NXSD Schema dialog is displayed, as shown in Figure 6–13.

Figure 6–13 Test NXSD Schema Dialog



13. Click the **Generate XML** icon. The resultant XML is displayed on the Result XML pane of the Test NXSD Schema dialog, as shown in Figure 6–14.

Figure 6–14 Test NXSD Schema Dialog

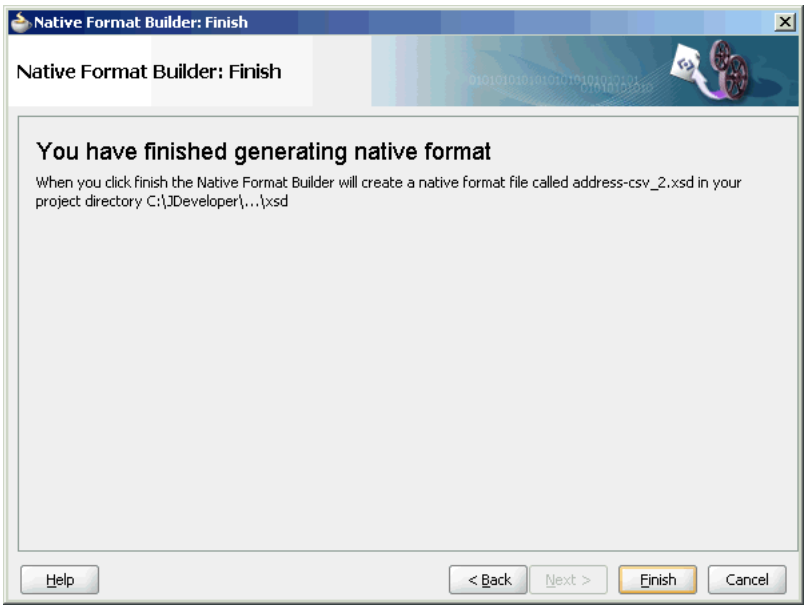


The native data using the corresponding native schema format is translated into the following XML:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<AddressBook xmlns="http://TargetNamespace.com/ReadFile">
   <Address>
      <Name>Oracle India Private Limited</Name>
      <Street1> Lexington Towers Prestige St. John's Woods</Street1>
      <Street2> 2nd Cross Road Chikka Audugodi</Street2>
      <City> Bangalore</City>
      <State> Karnataka</State>
      <Country> India</Country>
   </Address>
   <Address>
      <Name>Intel Technology India Private Limited</Name>
      <Street1> Survey #23-56 P Devarabeesanahalli Village</Street1>
      <Street2> Outer Ring Road Varthur Hobli</Street2>
      <City> Bangalore</City>
      <State> Karnataka</State>
      <Country> India</Country>
   </Address>
</AddressBook>
```

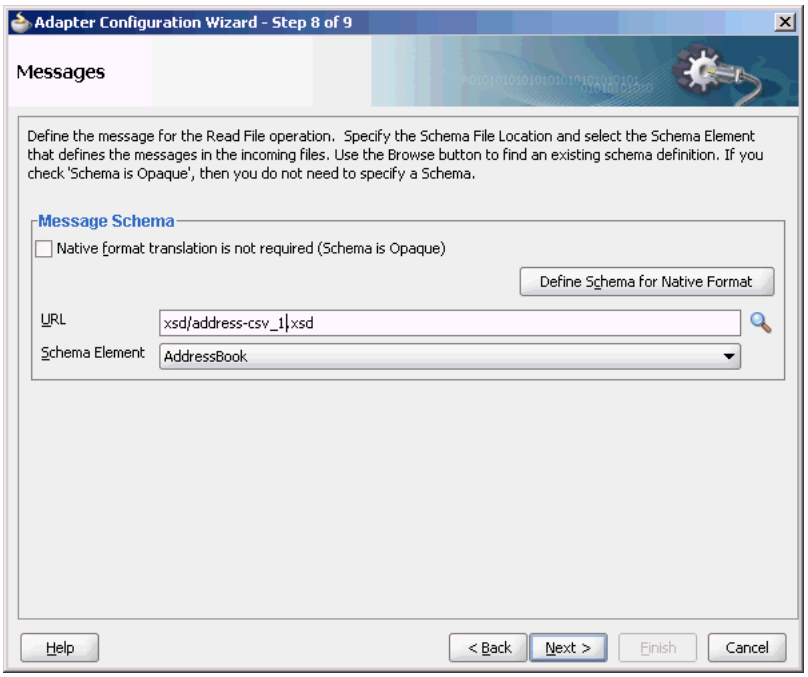
14. Click **OK**. The Generated Native Format Schema File page is displayed, as shown in Figure 6–12.
15. Click **Next**. The Native Format Builder Finish page is displayed, as shown in Figure 6–15.

Figure 6–15 Native Format Builder Wizard Finish Page



16. Click **Finish**. The Adapter Configuration Wizard Messages page is displayed, as shown in Figure 6–16, containing the generated NXSD.

Figure 6–16 Adapter Configuration Wizard Messages Page



6.4.1.1 Defining a Asterisk (*) Separated Value File Structure

The use case defined in the previous example is just one specific case of the *SV class, where the wildcard can be substituted by any character or string. For example, for the native data containing a plus (+) separated value, substitute the wildcard with the plus (+) character.

Use the **Delimited** type option in the Native Format Builder wizard when creating the XML schema for this native file.

Native Data Format to Be Translated

The following native data format is provided:

```
a+b+c+d+e
f+g+h+i+j
```

Native Schema

The corresponding native schema definition is similar to the one in the previous use case except that instead of `nxsd:terminatedBy=","` you now define the terminated by format as `nxsd:terminatedBy="+"`. See Section 6.2.2.2, "Defining Terminated Data" for details about the terminated style.

6.4.2 Defining the Schema for a Fixed Length File Structure

In this example, the native data used is the same as in the CSV case, but the data used is of type fixed length and not CSV.

Use the Fixed Length option in the Native Format Builder wizard, to create the XML schema for this native file.

In this use case, the Native Format Builder uses a fixed-length file type called `address` that contains the address details such as name, street, city, state, and country. Every element in this `address` native file has a fixed length. You can generate the corresponding NXSD and also test it. Perform the following steps to run the use case:

1. The data in a sample text file, `address.txt`, appears as below:

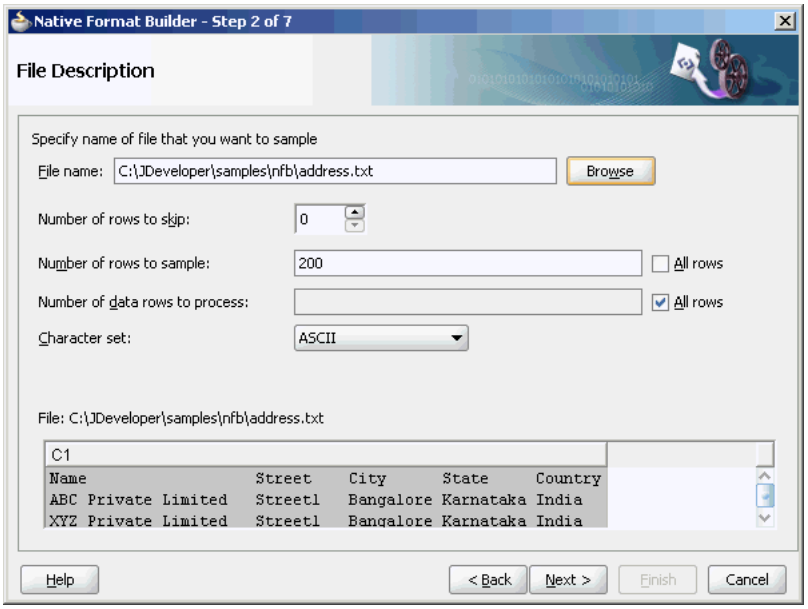
```
Name                 Street    City      State     Country
ABC Private Limited  Street1   Bangalore Karnataka India
XYZ Private Limited  Street1   Bangalore Karnataka India
```

2. Launch the Adapter Configuration Wizard and navigate to the Messages page, as displayed in Figure 6–4, and click **Define Schema For Native Format**. The Native Format Builder Welcome page is displayed, as shown in Figure 6–5.
3. Click **Next**. The Choose Type page is displayed.
4. Select **Fixed Length** as the file type, as shown in Figure 6–17.

Figure 6–17 Native Format Builder Wizard Choose Type Page

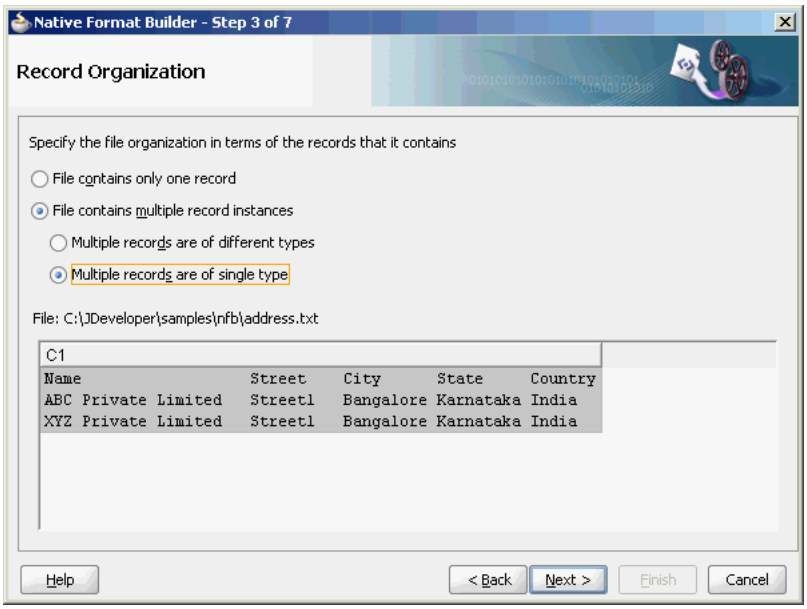
5. Click **Next**. The Native Format Builder File Description page is displayed.
6. Click **Browse** and select the `address.txt` file, as displayed in Figure 6–18.

Figure 6–18 Native Format Builder Wizard File Description Page



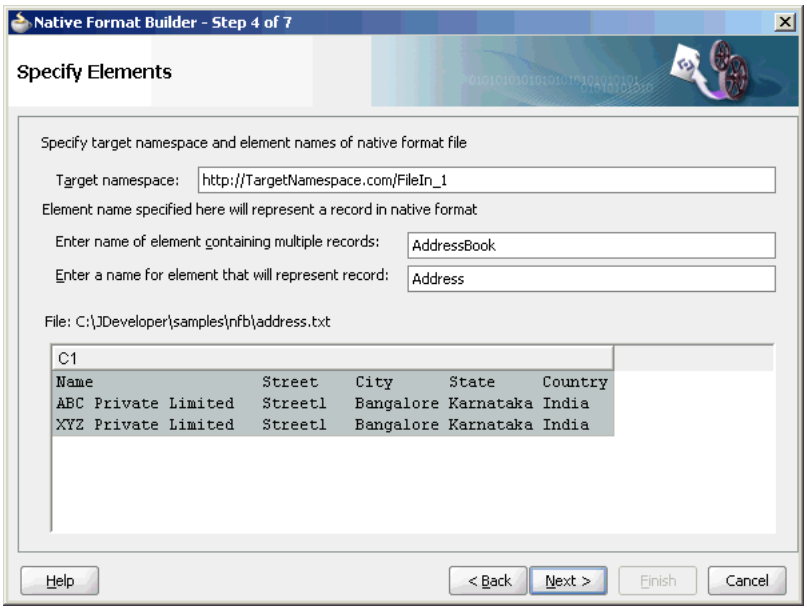
7. Click **Next**. The Record Organization page is displayed, as shown in Figure 6–19.

Figure 6–19 Native Format Builder Wizard Record Organization Page



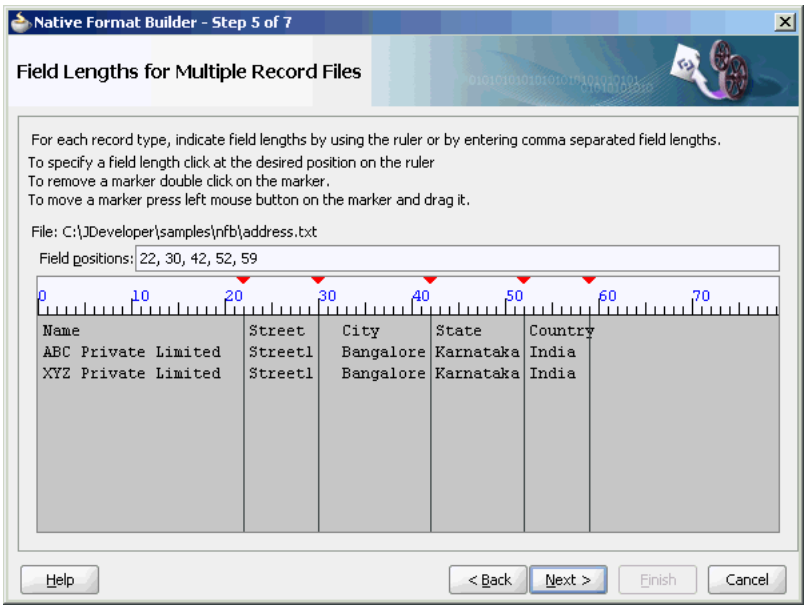
8. Select **Multiple records are of single type**, and click **Next**. The Specify Elements page is displayed.
9. Enter `AddressBook` in the **Enter name of element containing multiple records** field, and enter `Address` in the **Enter a name for element that will represent record** field, as shown in Figure 6–20.

Figure 6–20 Native Format Builder Wizard Specify Elements Page



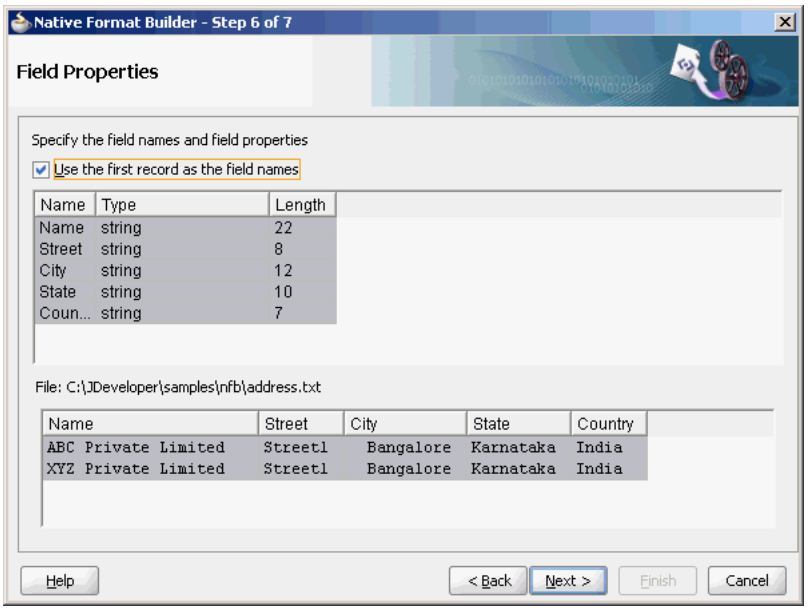
10. Click **Next**. The Field Lengths for Multiple Record Files page is displayed.
11. Click the ruler at the desired position to mark fields on the sample text area, as shown in Figure 6–21 and click **Next**. The Field Properties page is displayed.

Figure 6–21 Native Format Builder Wizard Field Lengths for Multiple Record Files Page



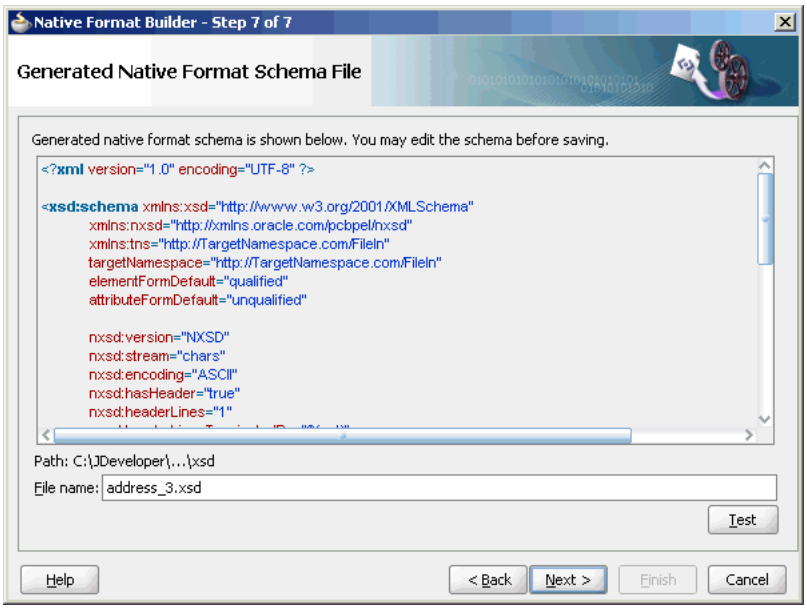
12. Check **Use the first record as the field names**, as shown in Figure 6–22.

Figure 6–22 Native Format Builder Wizard Field Properties Page



13. Click **Next**. The Generated Native Format Schema File page is displayed, as shown in Figure 6–23.

Figure 6–23 Native Format Builder Wizard Native Format Schema File Page



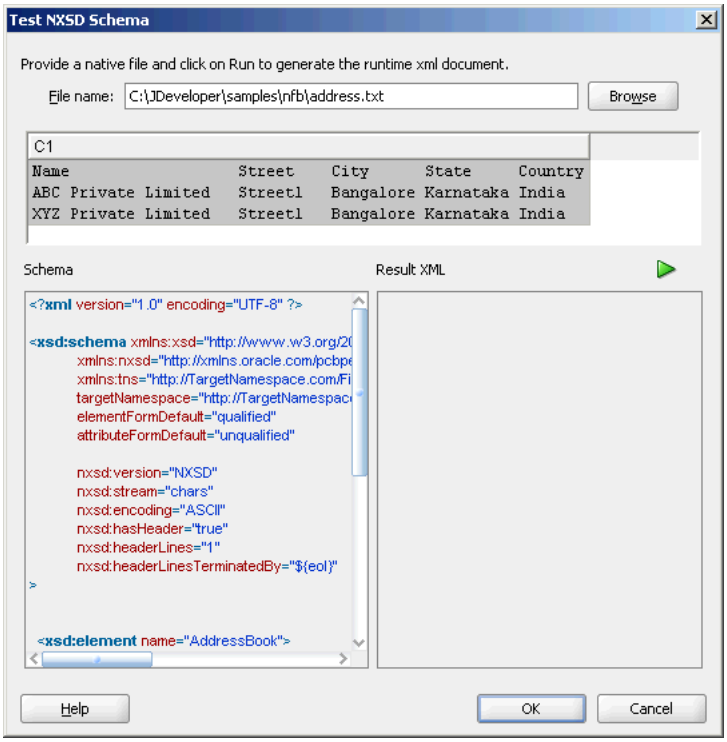
The corresponding native schema definition is similar to the definition of the CSV, file but `style` changes from `nxsd:style="terminated"` to `nxsd:style="fixedLength"` along with the relevant attributes for the fixed-length style. For the fixed-length style, the one mandatory attribute is the length: `nxsd:length`. The value of `nxsd:length` is the actual length of the data to be read.

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            xmlns:tns="http://TargetNamespace.com/FileIn_1"
            targetNamespace="http://TargetNamespace.com/FileIn_1"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"

            nxsd:version="NXSD"
            nxsd:stream="chars"
            nxsd:encoding="ASCII"
            nxsd:hasHeader="true"
            nxsd:headerLines="1"
            nxsd:headerLinesTerminatedBy="${eol}"

  <xsd:element name="AddressBook">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Address" minOccurs="1" maxOccurs="unbounded"
nxsd:style="array" nxsd:cellSeparatedBy="${eol}">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Name" type="xsd:string"
nxsd:style="fixedLength" nxsd:length="22" />
              <xsd:element name="Street" type="xsd:string"
nxsd:style="fixedLength" nxsd:length="8" />
              <xsd:element name="City" type="xsd:string"
nxsd:style="fixedLength" nxsd:length="12" />
              <xsd:element name="State" type="xsd:string"
```

```
nxsd:style="fixedLength" nxsd:length="10" />
              <xsd:element name="Country" type="xsd:string"
nxsd:style="fixedLength" nxsd:length="7" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

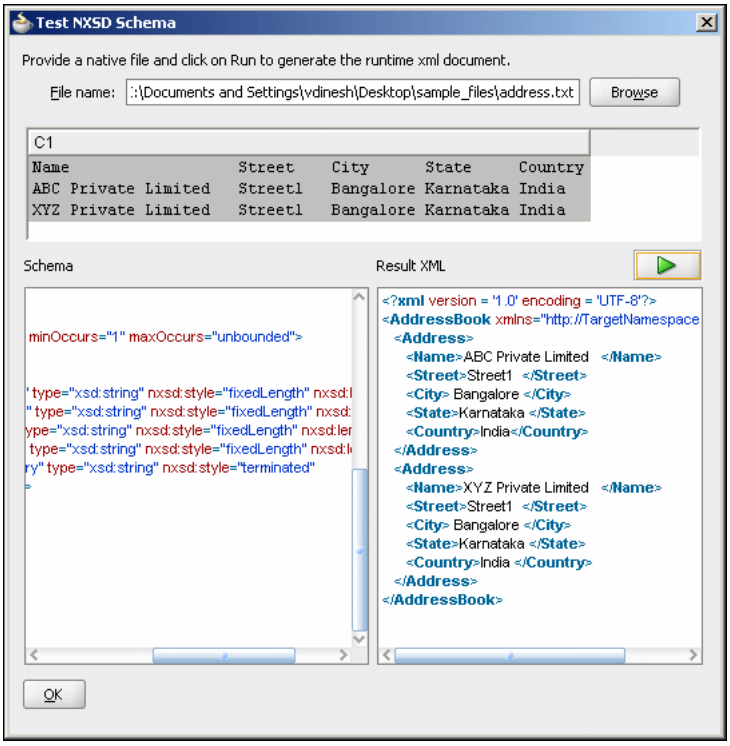
14. Click **Test**. The Test NXSD Schema dialog is displayed, as shown in Figure 6–24.

Figure 6–24 Test NXSD Schema Dialog



15. Click the **Generate XML** icon. The resultant XML is displayed on the Result XML pane of the Test NXSD Schema dialog, as shown in Figure 6–25.

Figure 6–25 Test NXSD Schema Dialog



The native data using the corresponding native schema format is translated into the following XML:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<AddressBook xmlns="http://TargetNamespace.com/Read">
   <Address>
      <Name>ABC Private Limited   </Name>
      <Street>Street1  </Street>
      <City> Bangalore </City>
      <State>Karnataka </State>
      <Country>India</Country>
   </Address>
   <Address>
      <Name>XYZ Private Limited   </Name>
      <Street>Street1  </Street>
      <City> Bangalore </City>
      <State>Karnataka </State>
      <Country>India </Country>
   </Address>
</AddressBook>
```

16. Click **OK**. The Generated Native Format File page is displayed, as shown in Figure 6–23.
17. Click **Next**. The Native Format Builder Finish page is displayed, as shown in Figure 6–15.
18. Click **Finish**. The Adapter Configuration Wizard Messages page is displayed, as shown in Figure 6–16, that contains the generated NXSD.

6.4.3 Defining the Schema for a Complex File Structure

The file structure of an invoice is more complex than the structure of CSV, *SV, and fixed-length files discussed in the preceding use cases. An invoice usually contains buyer information, seller information, and line items. Each of these elements, in turn, can be of complex type. For example, the buyer element can be defined as a partner-type, where partner-type consists of three elements - id, name, and address.

Use the Complex Type option in the Native Format Builder wizard when creating the XML schema for this native file.

In this use case, the Native Format Builder uses `invoice.txt`, a complex file type called invoice, which contains multiple records such as buyer, seller, and items. Also, using this use case, you can generate the NXSD and test it. Perform the following steps to run this use case:

1. The data in a sample text file, `invoice.txt`, appears as below:

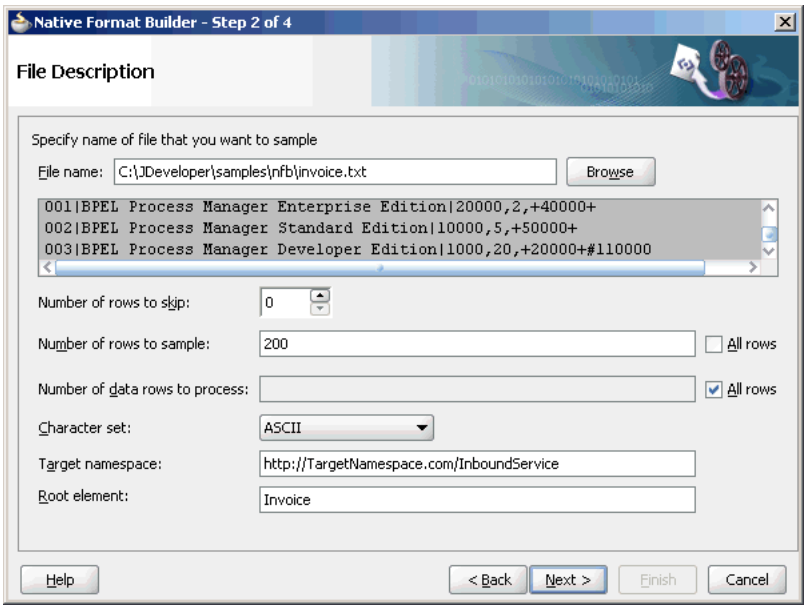
```
6335722^Company One^First Street   999       San Jose
95129USCA650-801-6250
       ^Oracle^Bridge Parkway 1600      Redwood Shores 94065USCA650-506-7000
001|BPEL Process Manager Enterprise Edition|20000,2,+40000+
002|BPEL Process Manager Standard Edition|10000,5,+50000+
003|BPEL Process Manager Developer Edition|1000,20,+20000+#110000
```

2. Launch the Adapter Configuration Wizard and navigate to the Messages page, as displayed in Figure 6–4, and click **Define Schema For Native Format**. The Native Format Builder Welcome page is displayed, as shown in Figure 6–5.
3. Click **Next**. The Choose Type page is displayed, as shown in Figure 6–26.

Figure 6–26 Native Format Builder Wizard Choose Type Page

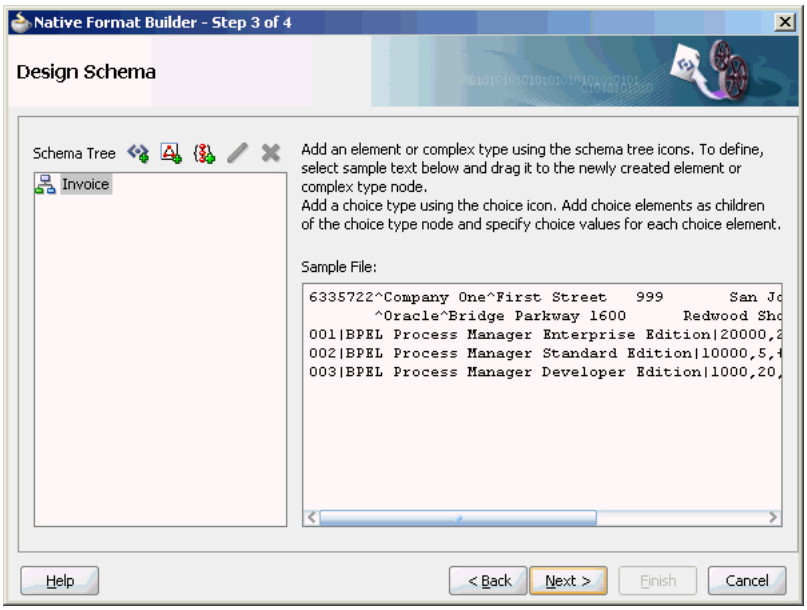
4. Select **Complex Type (Contains records whose fields may themselves be records having multiple delimiter types)**.
5. Click **Next**. The Native Format Builder File Description page is displayed.
6. Click **Browse** and select the `invoice.txt` file, and enter `Invoice` in the Root Element field, as displayed in Figure 6–27.

Figure 6–27 Native Format Builder Wizard File Description Page



7. Click **Next**. The Native Format Builder Design Schema is displayed, as shown in Figure 6–28.

Figure 6–28 Native Format Builder Wizard Design Schema Page



Create the partner-type Complex Type

The schema structure that you can build using the `invoice.txt` sample is as follows:

Invoice

Buyer => partner-type

Seller => partner-type

Items => item-type

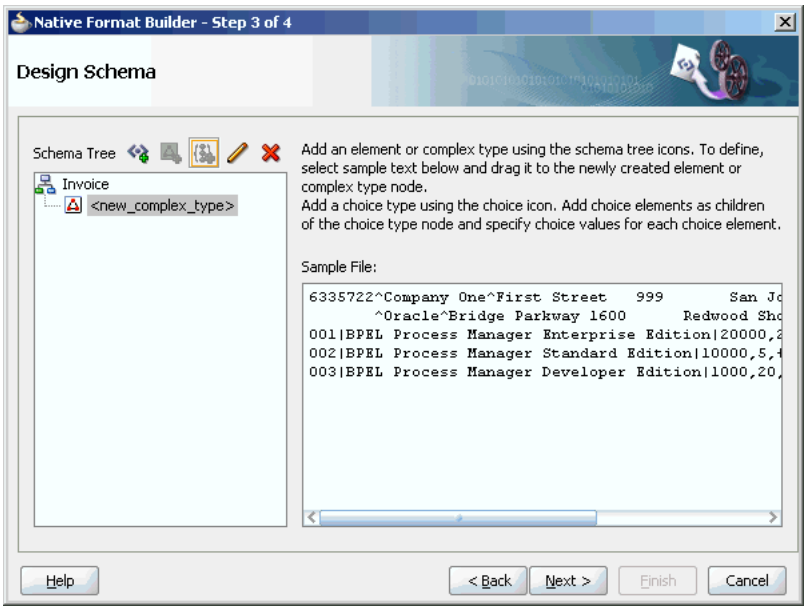
Invoice-total => double

The first line in the native data consists of buyer details, followed by seller details, followed by line items, and finally the total for the line items. Both buyer and seller elements have the same complex structure, as follows:

- The first seven characters are the UID
- This is followed by the buyer/seller name surrounded by "^".
- This is followed by the address until the end of the line.

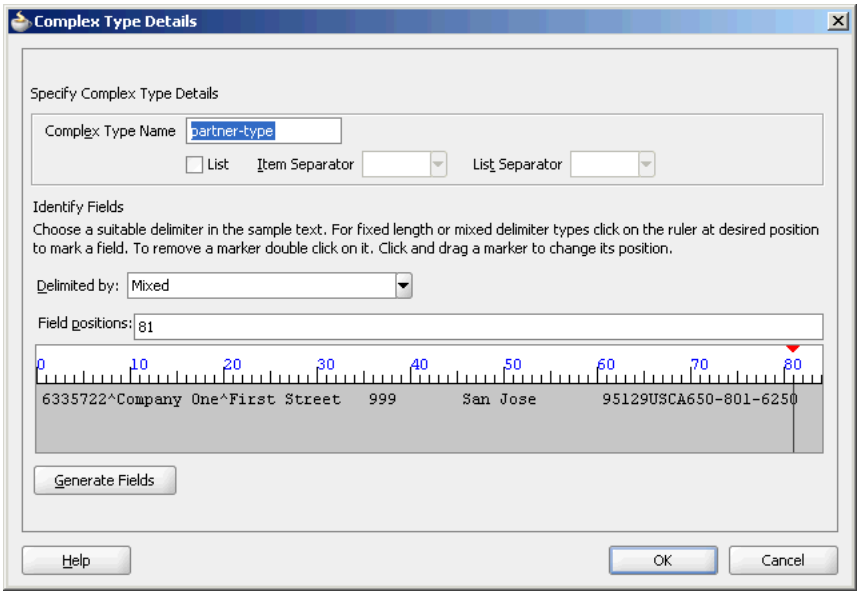
1. Click the **Add Complex Type** icon. A Complex Type, `<new_complex_type>` is created in the Schema Tree under `Invoice`, as shown in Figure 6–29.

Figure 6–29 Native Format Builder Wizard Design Schema Page



2. Select the first row of the sample text from the right-hand pane of the Sample File section, and drag and drop it on the **<new_complex_type>** node. The Complex Type Details dialog is displayed.
3. Enter `partner-type` in the Complex Type Name field, as shown in Figure 6–30.

Figure 6–30 Native Format Builder Wizard Design Schema Page - Complex Type Details Dialog

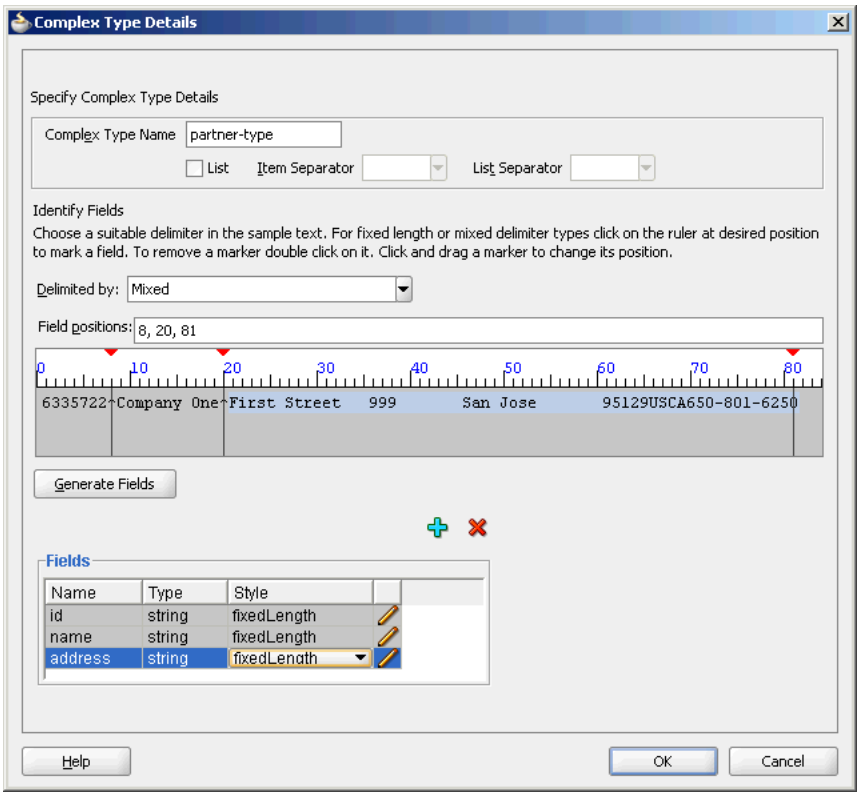


4. Click the ruler at the desired position to mark fields on the sample text area, and then click the **Generate Fields** button. The system tries to interpret the style of data for the defined fields.

> **Note:** For the Fixed Length or Mixed Delimiter type options, a ruler-based text area is displayed. You have to use the rulers to identify fields within the sample text. In case of delimited data, select or enter the appropriate delimiter in the Delimited By field.

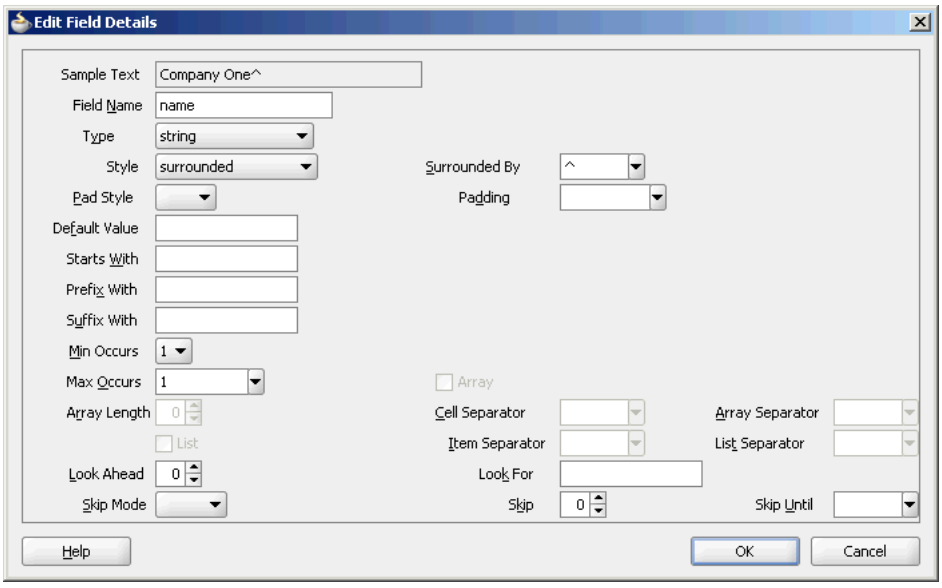
5. Enter **id**, **name**, and **address** in the Name field, as shown in Figure 6–31.

Figure 6–31 Complex Type Details Dialog



6. Click the pencil icon adjacent to each field to display the corresponding Edit Field Details dialog that enables you to edit the field properties. For example, click the pencil icon adjacent to the Name field. The Edit Field Details dialog is displayed, as shown in Figure 6–32.

Figure 6–32 Edit Field Details Dialog



7. Edit the following field properties, as shown in Figure 6–32.

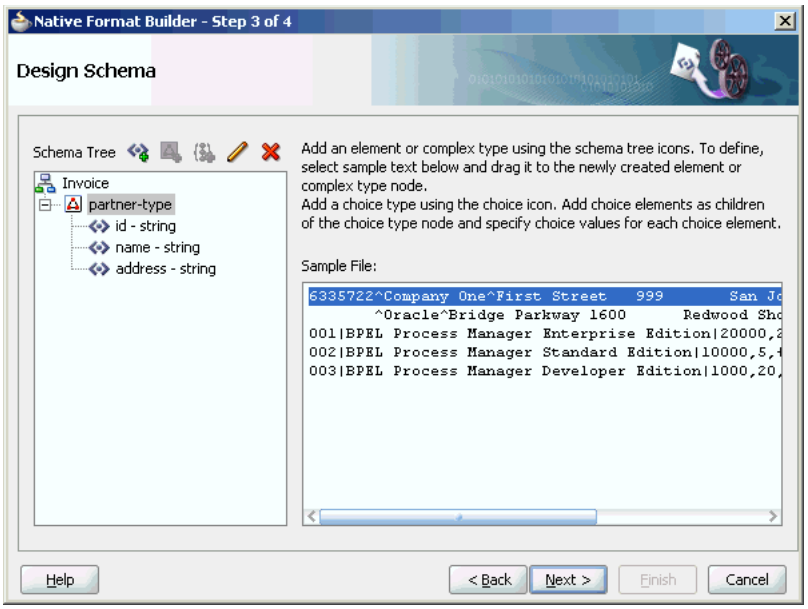
- **Type**: The data type of the sample text. Select **String** from the Type list.
- **Style**: Represents the style of the complex type element. You can select any of the following four options:
  - fixed length
  - surrounded
  - terminated
  - left/right surrounded

  In this example, select **surrounded**.
- **Surrounded By**: This option is displayed when you select surrounded in the Style option. In this example, enter caret (^) in the Surrounded By field.

The field properties displayed on this panel correspond to the NXSD attributes used in the schema.

8. Click **OK**. The Complex Type Details dialog is displayed with the field properties that you selected.
9. Verify or edit the field properties for `id` and `address` Name fields.
10. Click **OK** in the Complex Type Details dialog. The Native Format Builder Design Schema page is displayed, as shown in Figure 6–33.

Figure 6–33 Native Format Builder Wizard Design Schema Page - partner-type Complex Type



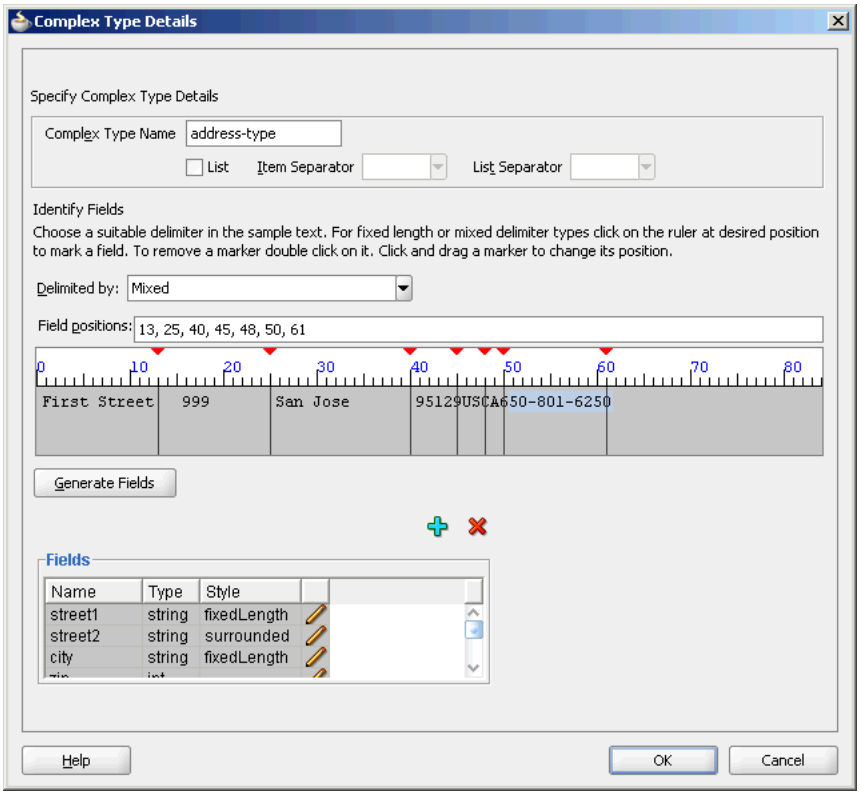
Create an address-type Complex Type

The address element can be further defined as another complex-type that contains a fixed-length street, city, and so on.

1. Create another <new_complex_type> node in the Schema Tree. See Step 1 in Create the partner-type Complex Type.
2. Drag and drop the address part in the first row of the sample text to the Complex Type, `<new_complex_type>`. The Complex Type Details dialog is displayed.

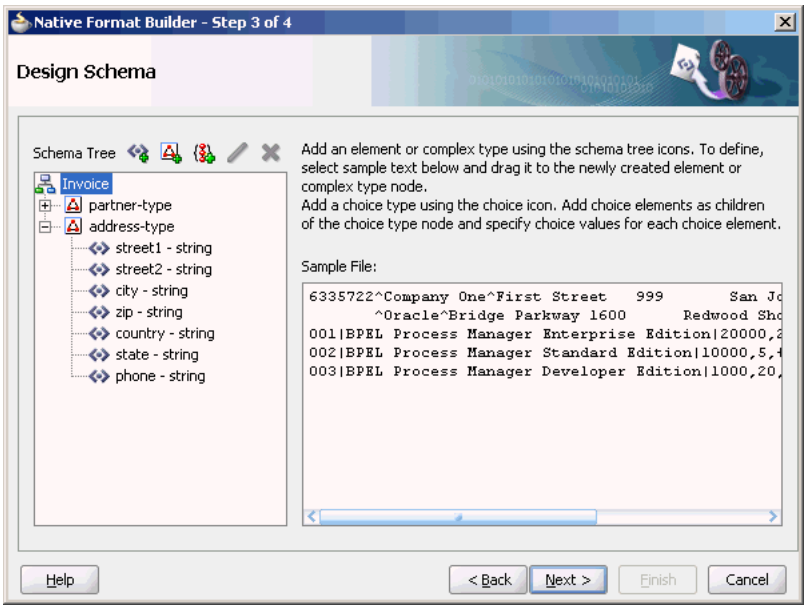
3. Enter **address-type** in the Complex Type Name field.
4. Click the ruler to mark fields on the sample text area, and then click the **Generate Fields** button. Now, enter **street1**, **street2**, **city**, **zip**, **country**, **state**, and **phone** in the Name field, as shown in Figure 6–34.

Figure 6–34 Native Format Builder Wizard Design Schema Page - Complex Type Details Dialog



5. Click **OK**. The Native Format Builder Design Schema page is displayed, as shown in Figure 6–35.

Figure 6–35 Native Format Builder Wizard Design Schema Page



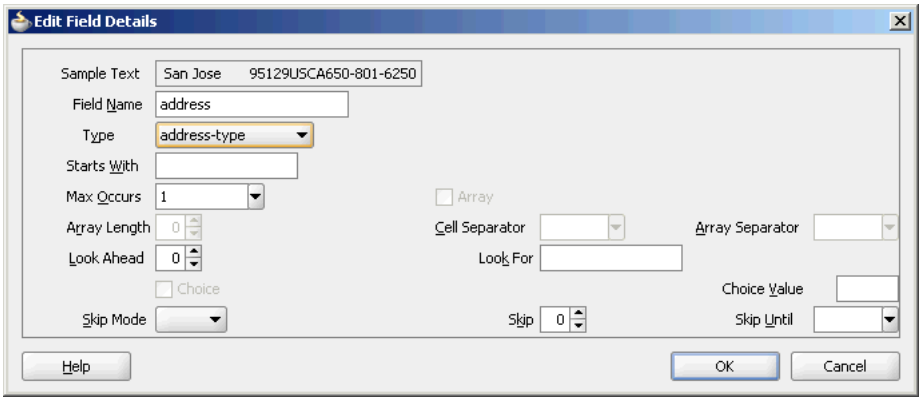
Assign the address-type Complex Type to address field of partner-type Complex Type

You must assign the address-type complex type to the address field of the partner-type complex type. You can assign a complex type to an element by using one of the following methods:

- Drag and drop the address-type node on the address field node of the partner-type complex type. This instantly assigns address-type to the address field element.
- Select the address field node of the partner-type complex type and then click the pencil icon.

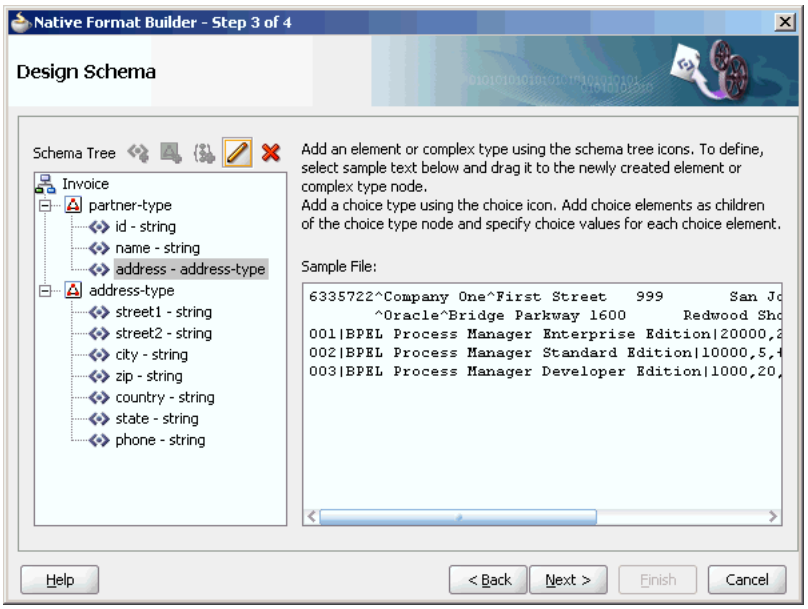
  The Edit Field Details dialog is displayed, as shown in Figure 6–36.

Figure 6–36 Edit Field Details Dialog



6. Select the **address-type** option in the Type list, and click **OK**. The address-type option is assigned to the address field element in the Native Format Builder Design Schema page, as shown in Figure 6–37.

Figure 6–37 Native Format Builder Wizard Design Schema Page

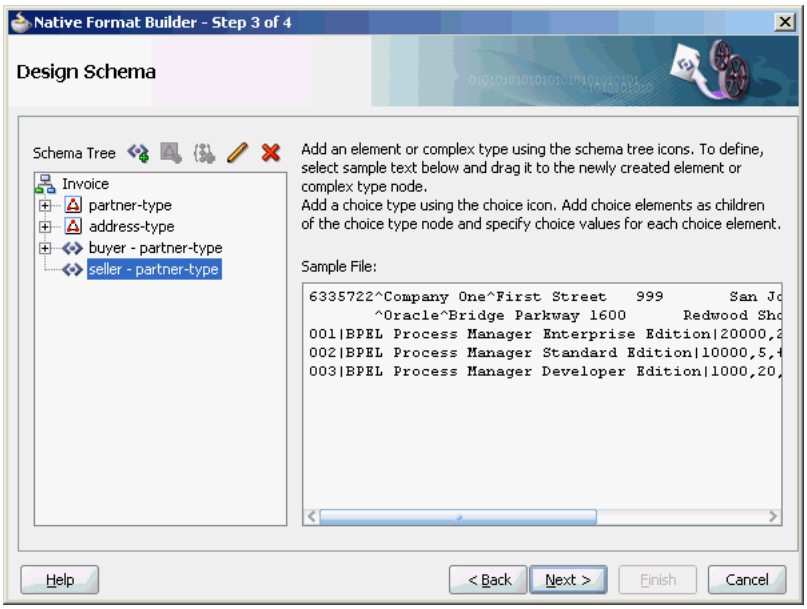


Create 'buyer' and 'seller' Global Elements

1. Select **Invoice**, and click the **Add Element** icon. An element, `<new_element>`, is created in the Schema Tree under the root element, **Invoice**.
2. Rename it to **buyer**.
3. Again, select **Invoice**, and click the **Add Element** icon. An element, `<new_element>`, is created in the Schema Tree under **Invoice**.
4. Rename it to **seller**.

   Now, drag and drop the partner-type node on each of the buyer and seller nodes, to assign the partner-type complex type to these nodes. The Schema Tree appears, as shown in Figure 6–38.

Figure 6–38 Native Format Builder Wizard Design Schema Page

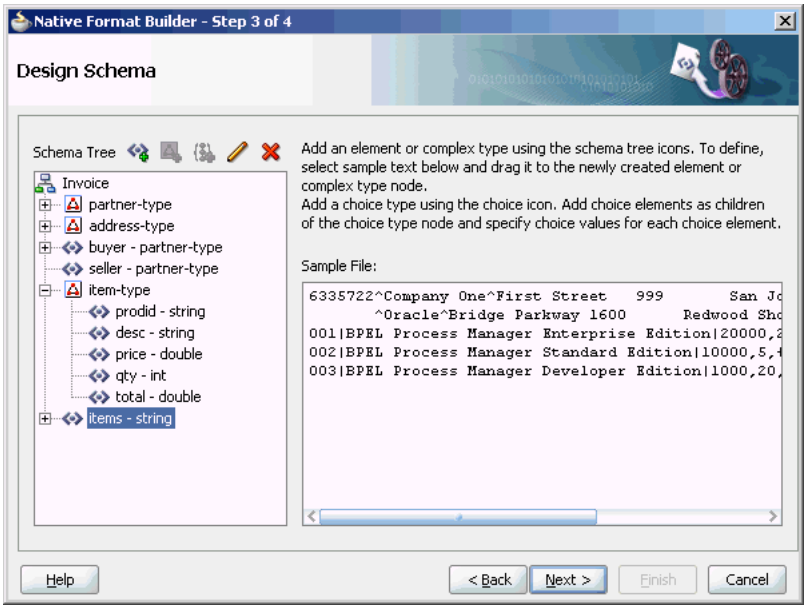


Create item-type Complex Type, and items and invoice-total Element Nodes

The items element can be considered an array of item-types. The last line item in the native file ends with the number sign (#), followed by the line-item total.

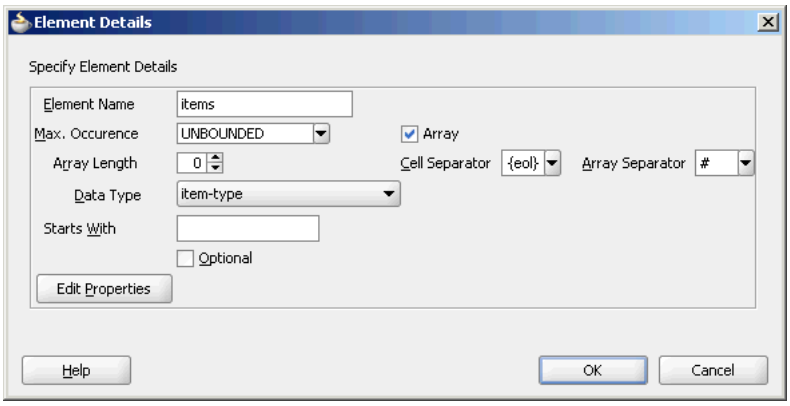
1. Select **Invoice**, and click the **Add Element** icon. An element, `<new_element>`, is created in the Schema Tree under **Invoice**.
2. Rename it to **items**.
3. Create the item-type complex type and define the field properties, as shown in Figure 6–39.

Figure 6–39 Native Format Builder Wizard Design Schema Page



4. Drag and drop **item-type** complex type to the **items** element to assign item-type to this element.
5. Select **items - item-type** and click the pencil icon. The Element Details dialog is displayed.

Figure 6–40 Element Details Dialog

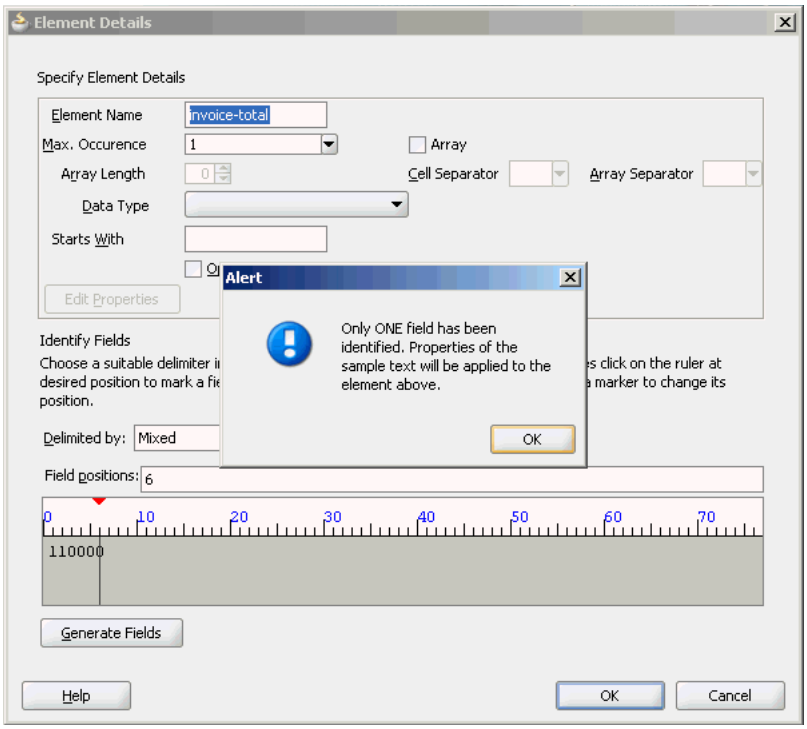


6. Set the following properties in the Element Details dialog, as shown in Figure 6–40:
   1. Set Max. Occurence - UNBOUNDED
   2. Select **Array**. The Cell Separator and Array Separator are enabled.
   3. Set Cell Separator - ${eol}
   4. Set Array Separator - #

   Note: The element `items` is defined as an array of `item-type`.

7. Click **OK**.
8. Create the invoice-total element, and drag and drop the sample text (110000) on the <new_element> node. The Element Details dialog is displayed.
9. Enter invoice-total in the Element Name field, and click **Generate Fields**. The Alert message is displayed, as shown in Figure 6–41.

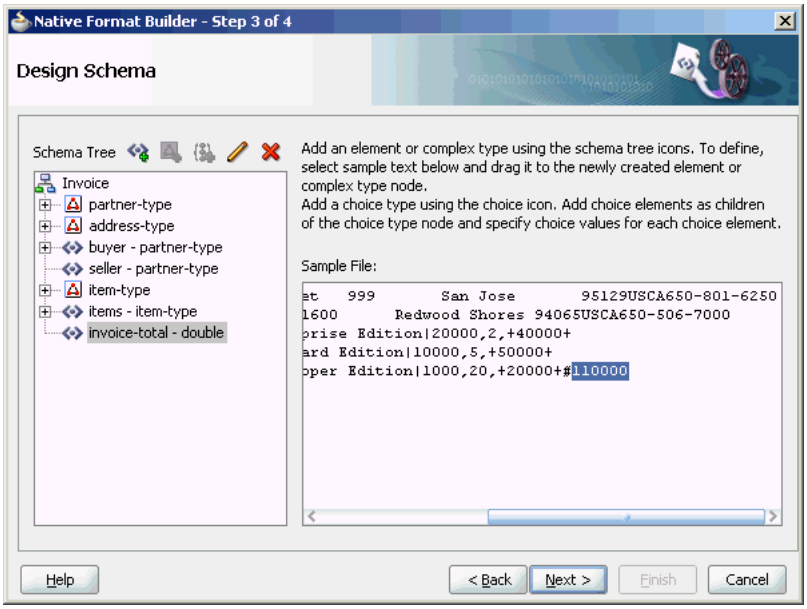
Figure 6–41 Element Details Dialog - Alert Message



If a single field is identified in the sampled data for a global element, then the properties of this data are applied to the global element itself.

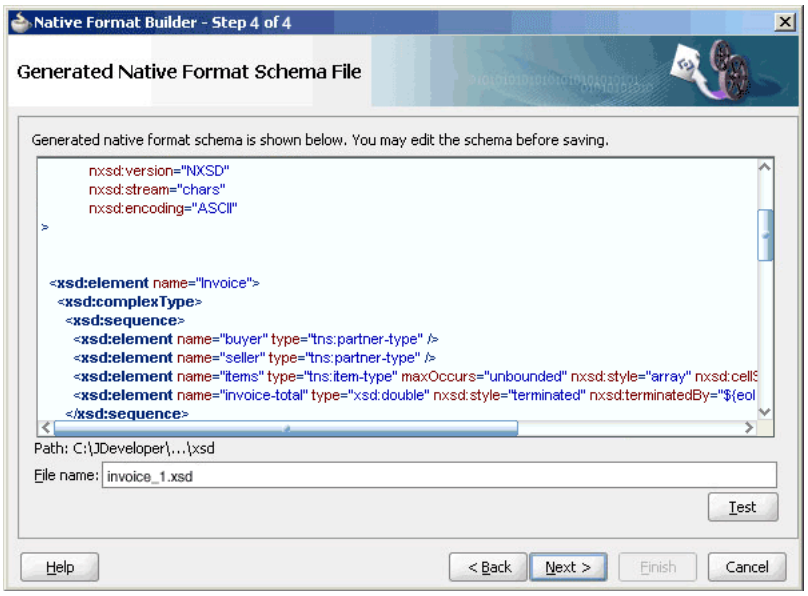
10. Click **OK** in the Alert message. The Element Details dialog is displayed.
11. Select **double** in the Data Type list, and click **OK**. The Native Format Builder Design Schema page is displayed, as shown in Figure 6–42.

Figure 6–42 Native Format Builder Wizard Design Schema Page - Complete Schema Tree



12. Click **Next**. The Generated Native Format Schema File page is displayed, as shown in Figure 6–43, which displays the native format file.

Figure 6–43 Native Format Builder Wizard Generated Native Format File Page



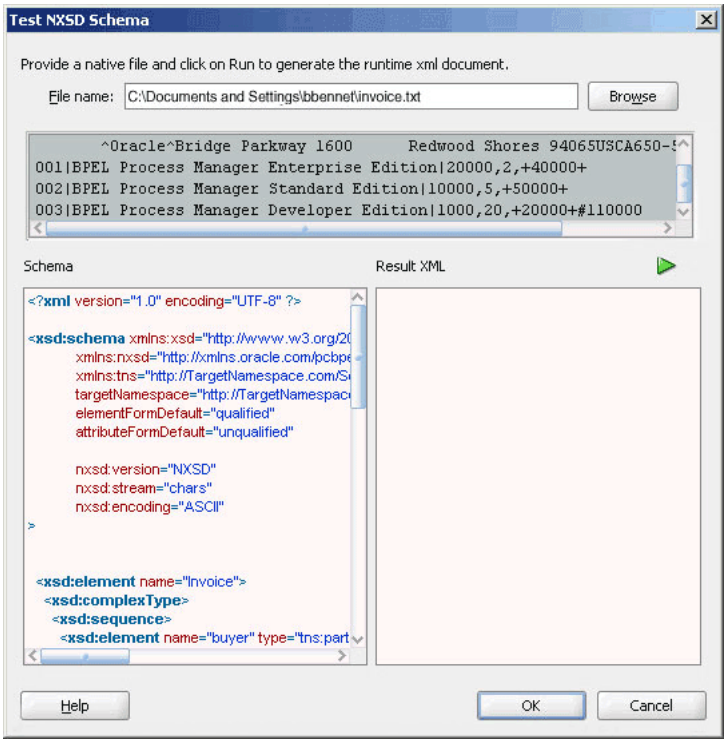
The native schema definition corresponding to the preceding native data can be defined as follows:

```
<schema attributeFormDefault="qualified" elementFormDefault="qualified"

targetNamespace="http://xmlns.oracle.com/ias/pcbpel/fatransschema/demo"

xmlns:tns="http://xmlns.oracle.com/ias/pcbpel/fatransschema/demo"
                xmlns="http://www.w3.org/2001/XMLSchema"
                xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
              nxsd:version="NXSD" nxsd:stream="chars">

    <element name="invoice" type="tns:invoiceType" />

    <complexType name="invoiceType">
       <sequence>
          <element name="purchaser" type="tns:partnerType" />
          <element name="seller" type="tns:partnerType" />
          <element name="line-item" type="tns:line-itemType"
              maxOccurs="unbounded" nxsd:style="array"
                nxsd:cellSeparatedBy="${eol}" nxsd:arrayTerminatedBy="#"/>
          <element name="total" type="double" nxsd:style="terminated"
                nxsd:terminatedBy="${eol}"/>
       </sequence>
    </complexType>

    <complexType name="partnerType">
       <sequence>
          <element name="uid" type="string" nxsd:style="fixedLength"
             nxsd:length="7" nxsd:padStyle="tail" nxsd:paddedBy=" "/>
          <element name="name" type="string" nxsd:style="surrounded"
             nxsd:surroundedBy="^"/>
          <element name="address" type="tns:addressType" />
       </sequence>
```

```
    </complexType>


    <complexType name="addressType">
       <sequence>
          <element name="street1" type="string" nxsd:style="fixedLength"
             nxsd:length="15" nxsd:padStyle="tail" nxsd:paddedBy=" "/>
          <element name="street2" type="string" nxsd:style="fixedLength"
             nxsd:length="10" nxsd:padStyle="tail" nxsd:paddedBy=" "/>
          <element name="city" type="string" nxsd:style="fixedLength"
             nxsd:length="15" nxsd:padStyle="tail" nxsd:paddedBy=" "/>
          <element name="postal-code" type="string" nxsd:style="fixedLength"
             nxsd:length="5" nxsd:padStyle="none"/>
          <element name="country" type="string" nxsd:style="fixedLength"
             nxsd:length="2" nxsd:padStyle="none"/>
          <element name="state" type="string" nxsd:style="fixedLength"
             nxsd:length="2" nxsd:padStyle="none"/>
          <element name="phone" type="string" nxsd:style="terminated"
             nxsd:terminatedBy="${eol}"/>
       </sequence>
    </complexType>

    <complexType name="line-itemType">
       <sequence>
          <element name="uid" type="string" nxsd:style="fixedLength"
             nxsd:length="3" nxsd:padStyle="none"/>
          <element name="description" type="string" nxsd:style="surrounded"
             nxsd:surroundedBy="|"/>
          <element name="price" type="double" nxsd:style="terminated"
               nxsd:terminatedBy=","/>
          <element name="quantity" type="integer" nxsd:style="terminated"
               nxsd:terminatedBy=","/>
          <element name="line-total" type="double" nxsd:style="surrounded"
               nxsd:surroundedBy="+"/>
       </sequence>
    </complexType>

</schema>
```

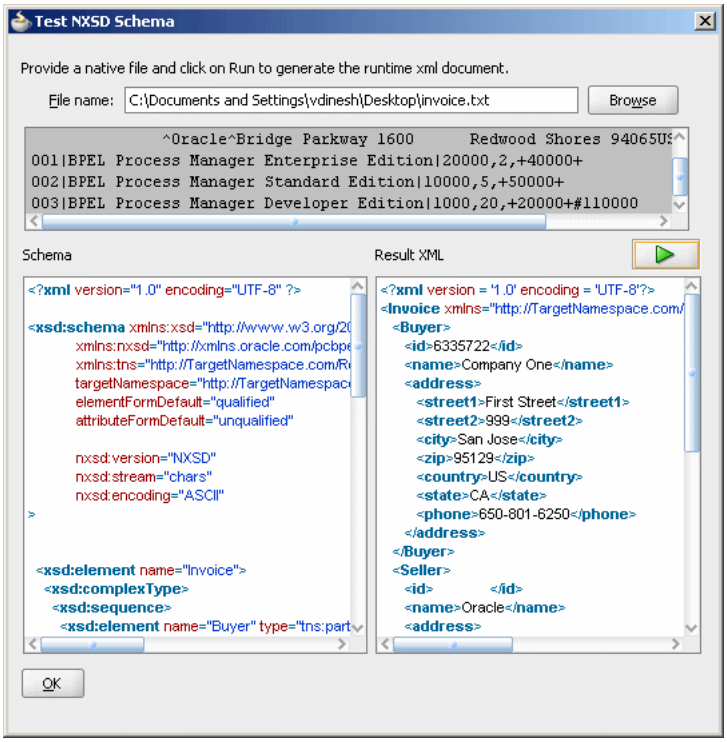
13. Click **Test**. The Test NXSD Schema dialog is displayed, as shown in Figure 6–44.

Figure 6–44 Test NXSD Schema Dialog



14. Click the **Generate XML** icon. The Result XML is displayed on the right pane of the Test NXSD Schema dialog, as shown in Figure 6–45.

Figure 6–45 Test NXSD Schema Dialog - Result XML



The translated XML 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>
```

15. Click **OK**. The Generated Native Format File page is displayed, as shown in Figure 6–43.
16. Click **Next**. The Native Format Builder Finish page is displayed, as shown in Figure 6–15.
17. Click **Finish**. The Adapter Configuration Wizard Messages page is displayed, containing the generated NXSD, as shown in Figure 6–16.

6.4.4 Removing or Adding Namespaces to XML with No Namespace

When the native data is XML and that XML has no namespace, you can use the Native Format Translator to add a namespace to an inbound XML document and remove the namespace from an outbound XML document.

The XML has no namespace when either of the following is true:

- The XML has a corresponding XML schema, and there is no target namespace specified in that XML schema.
- The XML has a corresponding DTD, which was converted to the XML schema.

In both cases, you must create a wrapper schema with `targetNamespace` specified, and the wrapper schema must include the actual schema. In addition, the wrapper schema must also have the `nxsd:version` attribute set to DTD. For example:

```
--wrapper.xsd
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="myNamespace"
   xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
   nxsd:version="DTD">
   <include schemaLocation="actual.xsd"/>
</schema>
```

> **Note:** Ensure that `elementFormDefault="qualified"` is specified in the actual schema.

Using this `wrapper.xsd` file for the original `.xsd` file would add the `myNamespace` namespace to the inbound XML and would remove the `myNamespace` namespace from the outbound XML.

6.4.5 Defining the Choice Condition Schema for a Complex File Structure

In this use case, the Native Format Builder uses order.txt, a complex type file, which contains multiple record types such as order, customer, and items. Also, using this use case you can generate the NXSD and test it. Perform the following steps to run this use case:

1. The data in a sample text file, order.txt, appears as below:

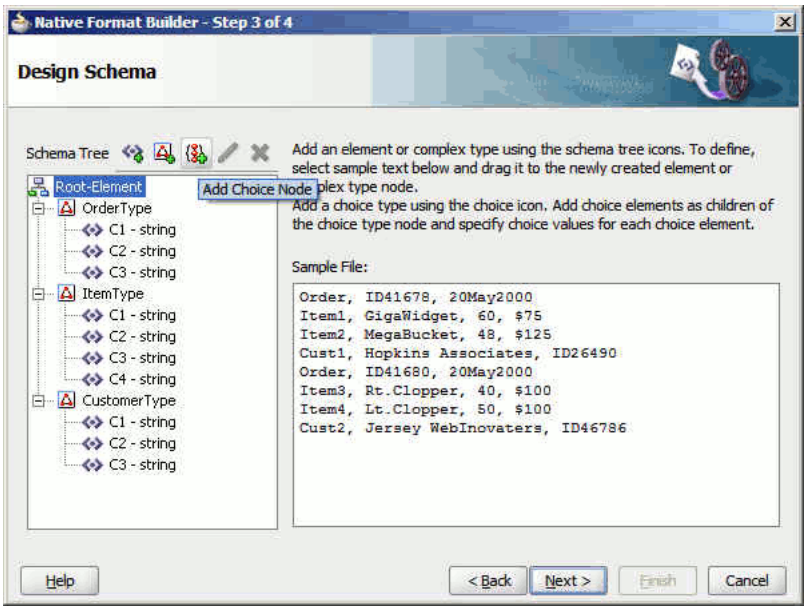
   ```
   Order, ID41678, 20May2000
   Item1, GigaWidget, 60, $75
   Item2, MegaBucket, 48, $125
   Cust1, Hopkins Associates, ID26490
   Order, ID41680, 20May2000
   Item3, Rt.Clopper, 40, $100
   Item4, Lt.Clopper, 50, $100
   Cust2, Jersey WebInovaters, ID46786
   ```

2. Create the following complex types by dragging one row each of order, customer, and item native data:
   - OrderType
   - ItemType
   - CustomerType

For more information about creating a complex type, see Section 6.4.3, "Defining the Schema for a Complex File Structure".

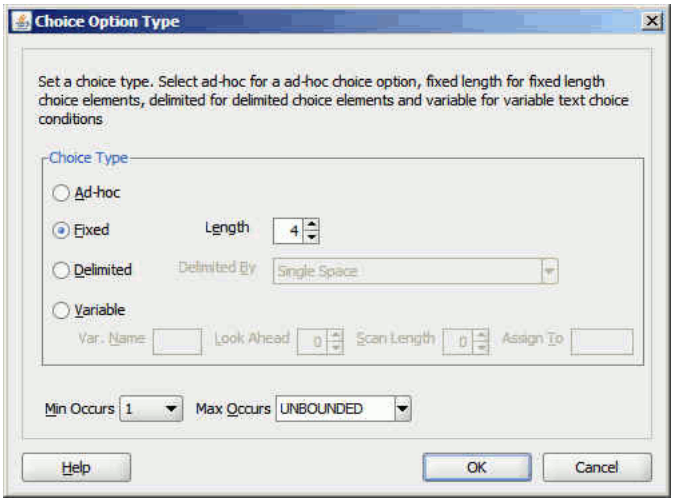
The Native Format Builder Design Schema page is displayed, as shown in Figure 6–46.

Figure 6–46 Native Format Builder Design Schema Page



3. Click **Add Choice Node**. The Choice Option Type dialog is displayed.
4. Set the options in the Choice Option Type dialog, as shown in Figure 6–47, and then click **OK**.

Figure 6–47 The Choice Option Type Dialog

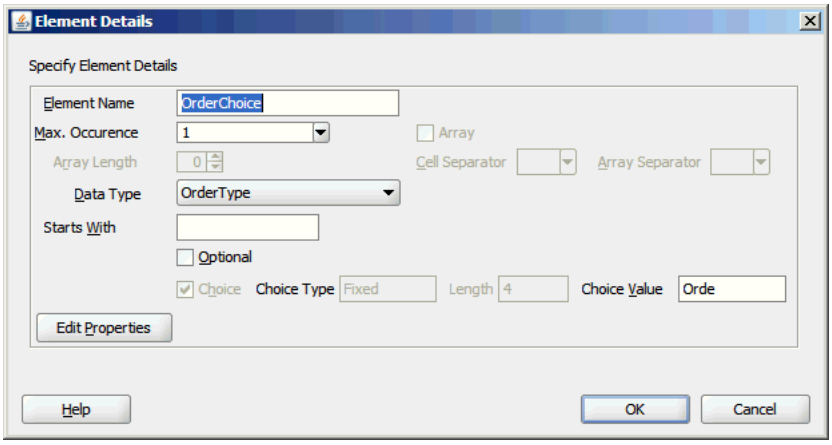


5. Select **choice** and click the **Add Element** icon. A <new_element> is added to the choice node.
6. Rename the newly added element to OrderChoice, and then drag and drop the OrderType complex type element to OrderChoice.

7. Select **OrderChoice - string** and click the **Edit Node** icon. The Element Details dialog is displayed.
8. Enter `Order` in the Choice Value field, as shown in Figure 6–48, and then click **OK**.

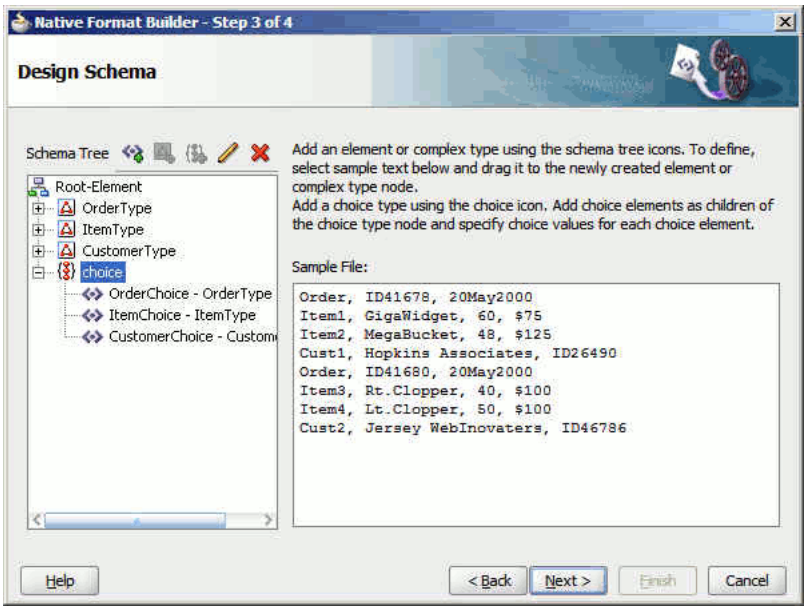
> **Note:** You should specify four characters in Choice Value field as the Length field has the value 4 in it.

Figure 6–48 The Element Details Dialog



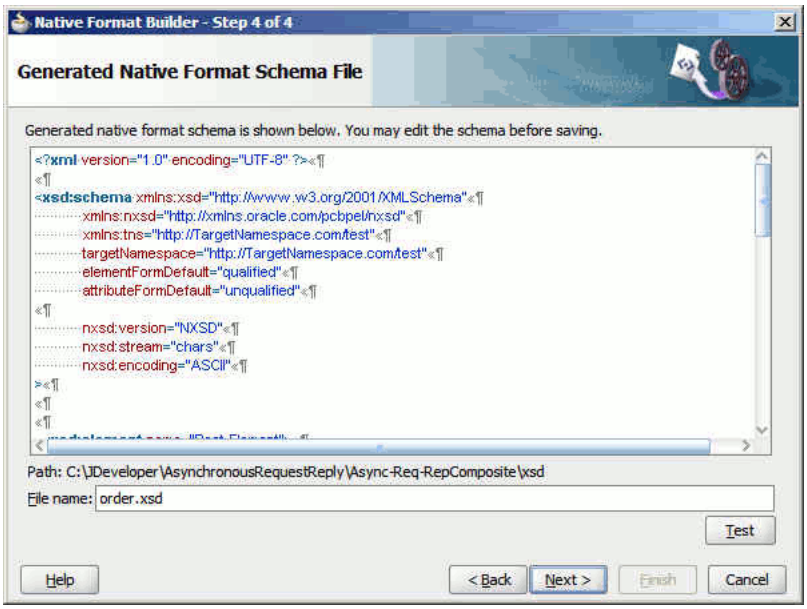
9. Follow Step 5 to 8 to create the ItemChoice choice complex type with ItemType data type and CustomerChoice choice complex type with CustomerType data type. The Native Format Builder Design Schema dialog is displayed, as shown in Figure 6–49.

Figure 6–49 Native Format Builder Design Schema Page



10. Click **Next**. The Generated Native Format Schema File page is displayed, as shown in Figure 6–50, which displays the native format file.

Figure 6–50 Generated Native Format Schema File Page



Native Schema

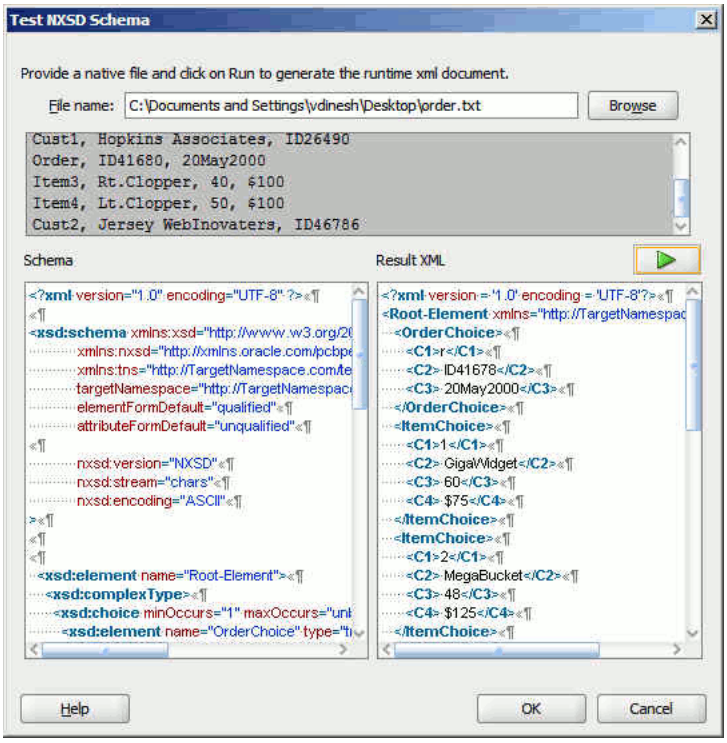
The native schema definition corresponding to the preceding native data can be defined as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            xmlns:tns="http://TargetNamespace.com/test"
            targetNamespace="http://TargetNamespace.com/test"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            nxsd:version="NXSD"
            nxsd:stream="chars"
            nxsd:encoding="ASCII"
>
  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:choice minOccurs="1" maxOccurs="unbounded"
 nxsd:choiceCondition="fixedLength" nxsd:length="4">
        <xsd:element name="OrderChoice" type="tns:OrderType"
 nxsd:conditionValue="Orde" />
        <xsd:element name="ItemChoice" type="tns:ItemType"
 nxsd:conditionValue="Item" />
        <xsd:element name="CustomerChoice" type="tns:customerType"
 nxsd:conditionValue="Cust" />
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="customerType">
    <xsd:sequence>
      <xsd:element name="C1" type="xsd:string" nxsd:style="terminated"
 nxsd:terminatedBy="," />
      <xsd:element name="C2" type="xsd:string" nxsd:style="terminated"
 nxsd:terminatedBy="," />
      <xsd:element name="C3" type="xsd:string" nxsd:style="terminated"
 nxsd:terminatedBy="${eol}" />
```

```
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ItemType">
    <xsd:sequence>
      <xsd:element name="C1" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="," />
      <xsd:element name="C2" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="," />
      <xsd:element name="C3" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="," />
      <xsd:element name="C4" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="${eol}" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="OrderType">
    <xsd:sequence>
      <xsd:element name="C1" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="," />
      <xsd:element name="C2" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="," />
      <xsd:element name="C3" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="${eol}" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

11. Click **Test**. The Test NXSD Schema dialog is displayed.
12. Click the **Generate XML** icon. The Result XML is displayed on the right pane of the Test NXSD Schema dialog, as shown in Figure 6–51.

Figure 6–51 Test NXSD Schema Dialog



Translated XML Using the Native Schema

The translated XML looks as follows:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<Root-Element xmlns="http://TargetNamespace.com/test">
   <OrderChoice>
      <C1>r</C1>
      <C2> ID41678</C2>
      <C3> 20May2000</C3>
   </OrderChoice>
   <ItemChoice>
      <C1>1</C1>
      <C2> GigaWidget</C2>
      <C3> 60</C3>
      <C4> $75</C4>
   </ItemChoice>
   <ItemChoice>
      <C1>2</C1>
      <C2> MegaBucket</C2>
      <C3> 48</C3>
      <C4> $125</C4>
   </ItemChoice>
   <CustomerChoice>
      <C1>1</C1>
      <C2> Hopkins Associates</C2>
      <C3> ID26490</C3>
   </CustomerChoice>
   <OrderChoice>
      <C1>r</C1>
      <C2> ID41680</C2>
      <C3> 20May2000</C3>
   </OrderChoice>
   <ItemChoice>
      <C1>3</C1>
      <C2> Rt.Clopper</C2>
      <C3> 40</C3>
      <C4> $100</C4>
   </ItemChoice>
   <ItemChoice>
      <C1>4</C1>
      <C2> Lt.Clopper</C2>
      <C3> 50</C3>
      <C4> $100</C4>
   </ItemChoice>
   <CustomerChoice>
      <C1>2</C1>
      <C2> Jersey WebInovaters</C2>
      <C3> ID46786</C3>
   </CustomerChoice>
</Root-Element>
```

13. Click **OK**. The Generated Native Format File page is displayed.
14. Click **Next**. The Native Format Builder Finish page is displayed.

6.4.6 Defining Choice Condition With LookAhead for a Complex File Structure

In this use case, the Native Format Builder uses `address.txt`, a complex type file, which contains multiple records with different addresses. In this use case, you would build a schema which has 2 record types. The RecOne record takes data for records

ending with text "YES" and the RecTwo record takes data for records ending with text "NO ".

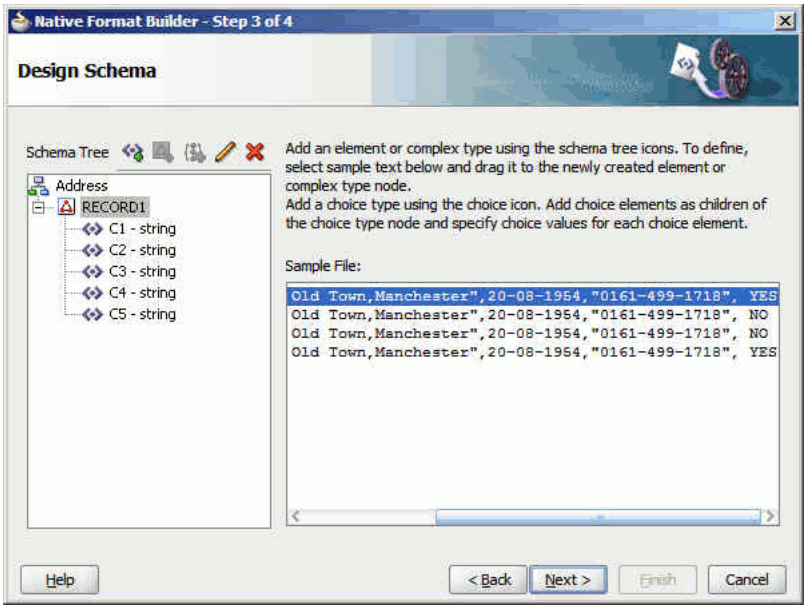
Also, using this use case you can generate the NXSD and test it. Perform the following steps to run this use case:

1. The data in a sample text file, address.txt, appears as below:

```
Name1,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", YES
Name2,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", NO
Name3,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", NO
Name4,"2 Old Street, Old Town,Manchester",20-08-1954,"0161-499-1718", YES
```

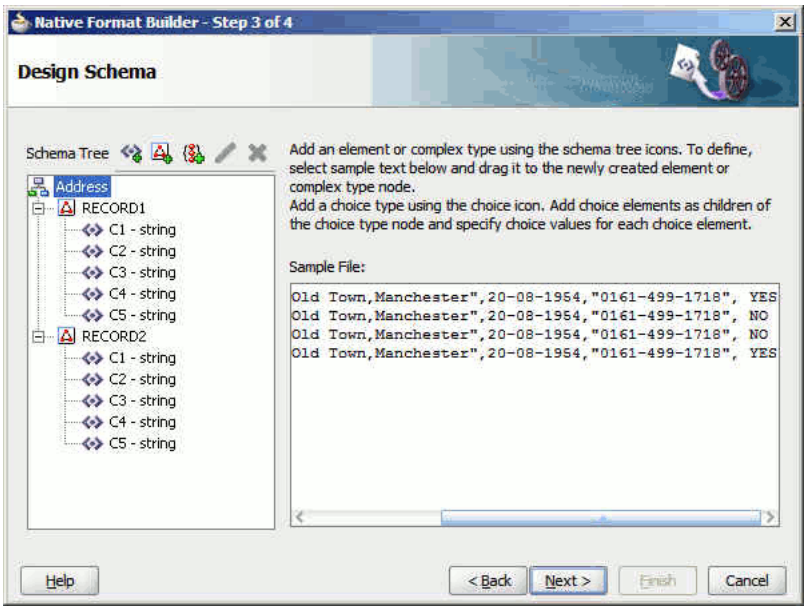
2. Launch the Adapter Configuration Wizard and navigate to the Messages page, and click **Define Schema For Native Format**. The Native Format Builder Welcome page is displayed.
3. Click **Next**. The Choose Type page is displayed.
4. Select **Complex Type** and click **Next**. The Native Format Builder File Description page is displayed.
5. Click **Browse** and select the address.txt file, and enter Address in the Root Element field.
6. Click **Next**. The Native Format Builder Design Schema page is displayed.
7. Click the **Add Complex Type** icon. A Complex Type, <new_complex_type> is created in the Schema Tree under Address.
8. Select the first row of the sample text from the right-hand pane of the Sample File section, and drag and drop it on the <new_complex_type> node. The Complex Type Details dialog is displayed.
9. Enter `RECORD1` in the Complex Type Name field and select **Comma (,)** in the Delimited By list.
10. Click **OK**. The Native Format Builder Design Schema page is displayed, as shown in Figure 6–52.

Figure 6–52 Native Format Builder Design Schema Page



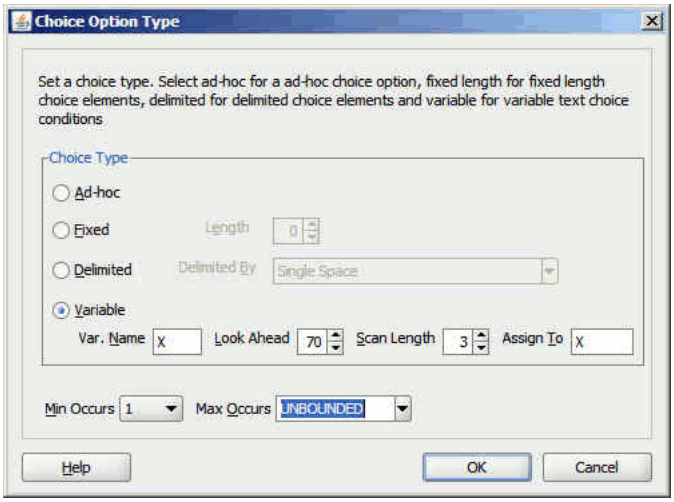
11. Similarly, create another complex type node called RECORD2. The Native Format Builder Design Schema page is displayed, as shown in Figure 6–53.

Figure 6–53 Native Format Builder Design Schema Page



12. Click **Add Choice Node**. The Choice Option Type dialog is displayed.
13. Set the options in the Choice Option Type dialog, as shown in Figure 6–54, and then click **OK**.

Figure 6–54 The Choice Option Type Dialog



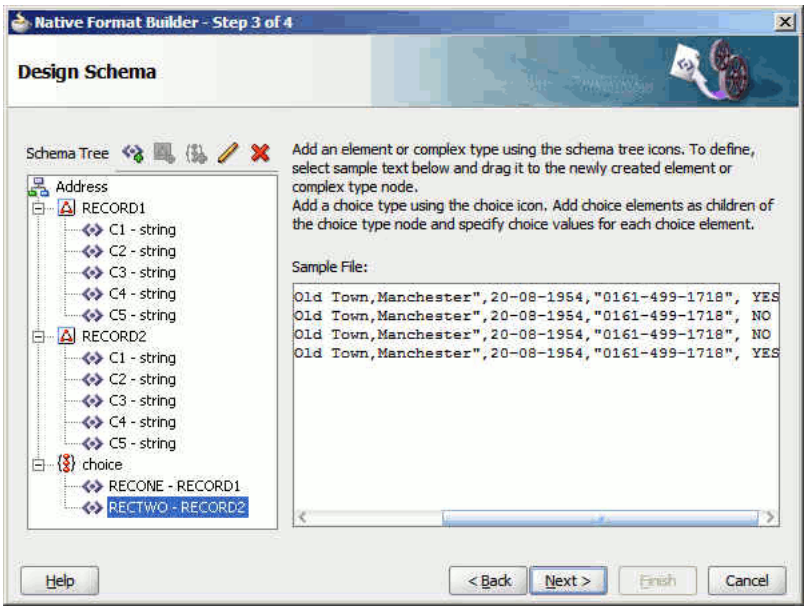
14. Select **choice** and click the **Add Element** icon. A <new_element> is added to the choice node.
15. Click the **Edit Node** icon. The Element Details dialog is displayed.
16. Enter `RECONE` in the Element Name field and select **RECORD1** as the Data Type set choice condition as "YES", and then click **OK**.

17. Follow Step 14 to 16 to create the `RECTWO` choice element for the choice node and set choice condition as "NO ".

> **Note:** There is one space after chars "NO", since you must match the total no. of characters to three.

The Native Format Builder Design Schema dialog is displayed, as shown in Figure 6–55.

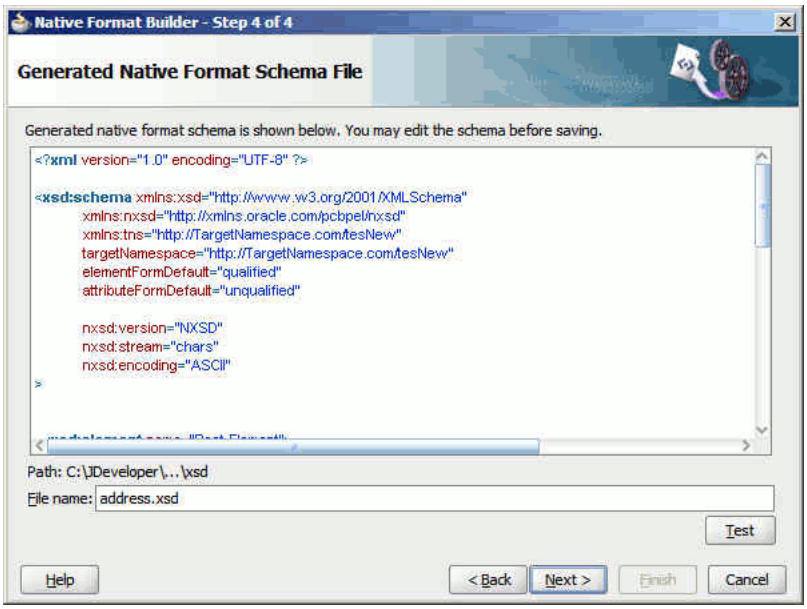
Figure 6–55 Native Format Builder Design Schema Page



18. Drag and drop the RECORD1 complex type to the RECONE element under choice and the RECORD2 complex type to the RECTWO element under choice. The Native Format Builder Design Schema dialog is displayed.

19. Click **Next**. The Generated Native Format Schema File page is displayed, as shown in Figure 6–56, which displays the native format file.

Figure 6–56 Generated Native Format Schema File Page



Native Schema

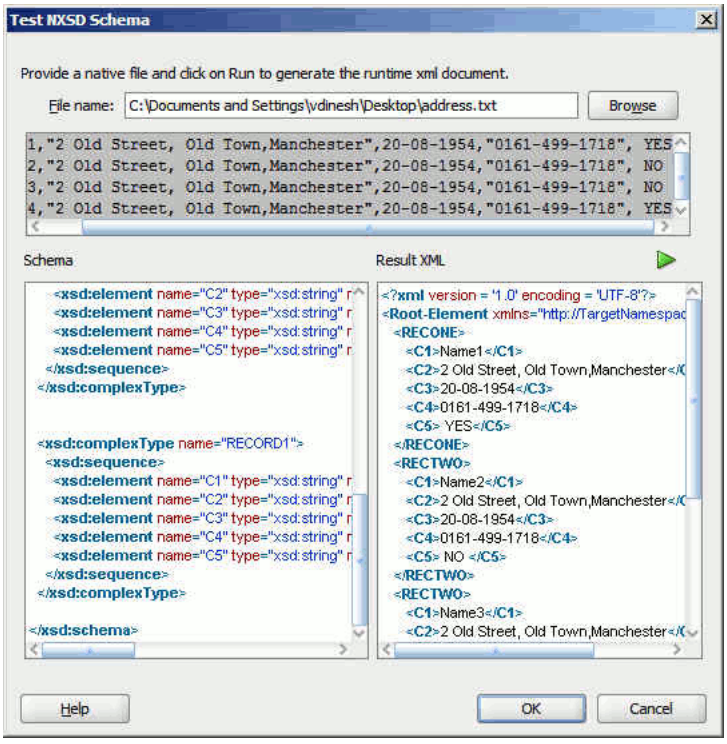
The native schema definition corresponding to the preceding native data can be defined as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            xmlns:tns="http://TargetNamespace.com/tesNew"
            targetNamespace="http://TargetNamespace.com/tesNew"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            nxsd:version="NXSD"
            nxsd:stream="chars"
            nxsd:encoding="ASCII"
>
  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:choice minOccurs="1" maxOccurs="unbounded"
 nxsd:choiceCondition="${X}" nxsd:lookAhead="70" nxsd:scanLength="3"
 nxsd:assignTo="${X}">
        <xsd:element name="RECTWO" type="tns:RECORD2" nxsd:conditionValue="NO "
 />
        <xsd:element name="RECONE" type="tns:RECORD1" nxsd:conditionValue="YES"
 />
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="RECORD2">
    <xsd:sequence>
      <xsd:element name="C1" type="xsd:string" nxsd:style="terminated"
 nxsd:terminatedBy="," />
      <xsd:element name="C2" type="xsd:string" nxsd:style="terminated"
 nxsd:terminatedBy="," />
      <xsd:element name="C3" type="xsd:string" nxsd:style="terminated"
 nxsd:terminatedBy="," />
      <xsd:element name="C4" type="xsd:string" nxsd:style="terminated"
```

```
nxsd:terminatedBy="," />
     <xsd:element name="C5" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="${eol}" />
   </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="RECORD1">
   <xsd:sequence>
     <xsd:element name="C1" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="," />
     <xsd:element name="C2" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="," />
     <xsd:element name="C3" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="," />
     <xsd:element name="C4" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="," />
     <xsd:element name="C5" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="${eol}" />
   </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

20. Click **Test**. The Test NXSD Schema dialog is displayed.
21. Click the **Generate XML** icon. The Result XML is displayed on the right pane of the Test NXSD Schema dialog, as shown in Figure 6–57.

Figure 6–57 Test NXSD Schema Dialog



Translated XML Using the Native Schema

The translated XML looks as follows:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<Root-Element xmlns="http://TargetNamespace.com/tesNew">
   <RECONE>
```

```
      <C1>Name1</C1>
      <C2>2 Old Street, Old Town,Manchester</C2>
      <C3>20-08-1954</C3>
      <C4>0161-499-1718</C4>
      <C5> YES</C5>
   </RECONE>
   <RECTWO>
      <C1>Name2</C1>
      <C2>2 Old Street, Old Town,Manchester</C2>
      <C3>20-08-1954</C3>
      <C4>0161-499-1718</C4>
      <C5> NO </C5>
   </RECTWO>
   <RECTWO>
      <C1>Name3</C1>
      <C2>2 Old Street, Old Town,Manchester</C2>
      <C3>20-08-1954</C3>
      <C4>0161-499-1718</C4>
      <C5> NO </C5>
   </RECTWO>
   <RECONE>
      <C1>Name4</C1>
      <C2>2 Old Street, Old Town,Manchester</C2>
      <C3>20-08-1954</C3>
      <C4>0161-499-1718</C4>
      <C5> YES</C5>
   </RECONE>
</Root-Element>
```

Note: There are 2 recordtypes: RECONE and RECTWO. RECONE takes records that end with character YES and RECTWO takes records that end with character NO.

22. Click **OK**. The Generated Native Format File page is displayed.
23. Click **Next**. The Native Format Builder Finish page is displayed.

6.4.7 Defining Array Type Schema for a Complex File Structure

In this use case, the Native Format Builder uses array.txt, a complex type file, which contains an array of items. The sample data has four names which are separated by a semicolon and ending with a period. In this use case, you would create a schema with array type which has member names separated by a semicolon and array terminated by a period. Also, using this use case you can generate the NXSD and test it.

Perform the following steps to run this use case:

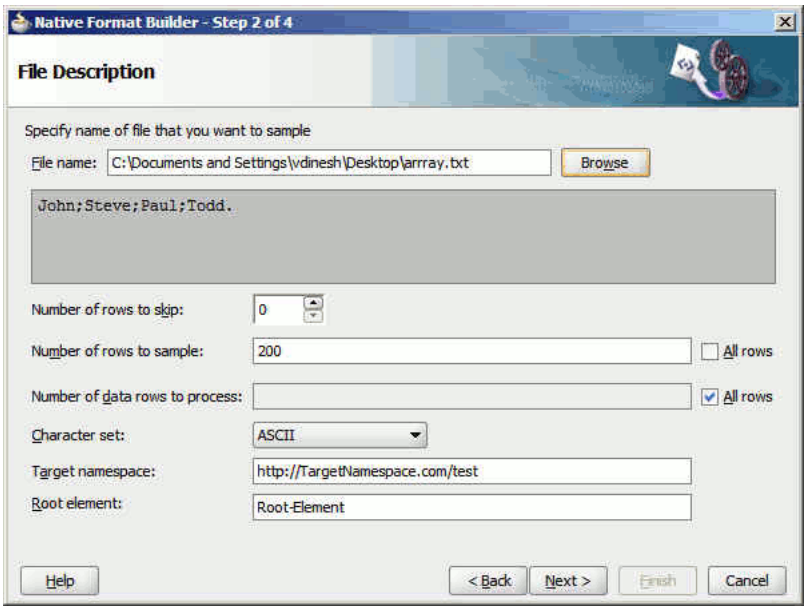
1. The data in a sample text file, array.txt, appears as below:

   ```
   John;Steve;Paul;Todd.
   ```

2. Launch the Adapter Configuration Wizard and navigate to the Messages page and click **Define Schema For Native Format**. The Native Format Builder Welcome page is displayed.
3. Click **Next**. The Choose Type page is displayed.
4. Select **Complex Type** and click **Next**. The Native Format Builder File Description page is displayed.

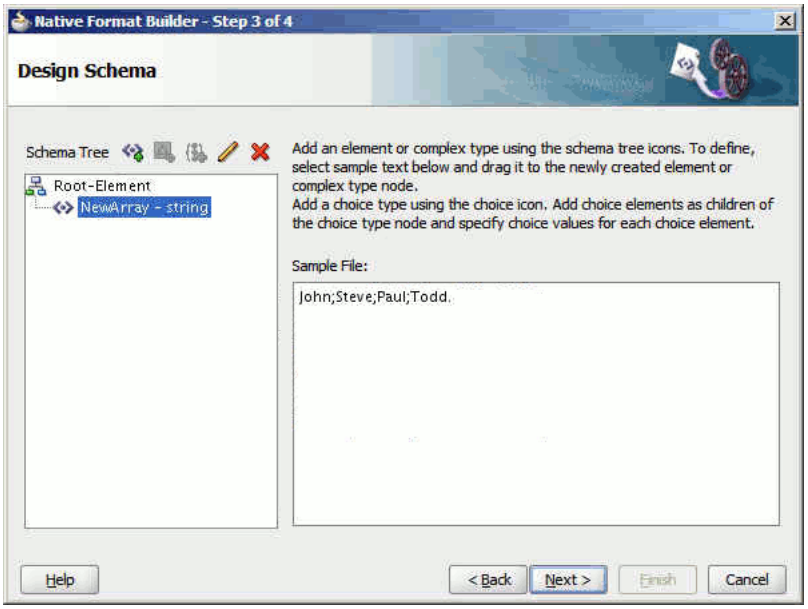
5. Click **Browse** and select the array.txt file, as shown in Figure 6–58. The Native Format Builder File Description page is displayed.

Figure 6–58 Native Format Builder File Description Page



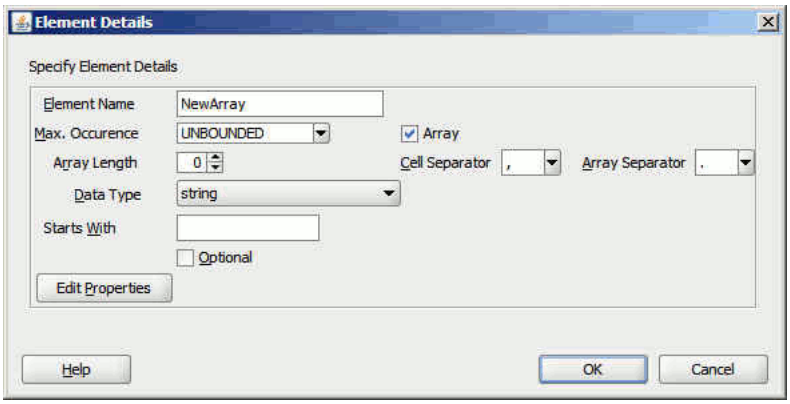
6. Click **Next**. The Native Format Builder Design Schema Page is displayed.
7. Create a global element called NewArray and drag and drop the native data to the newly created global element.
8. Select **NewArray**, as shown in Figure 6–59, and click the **Edit Node** icon. The Element Details dialog is displayed.

Figure 6–59 Native Format Builder Design Schema Page



9. Set the options in the Element Details dialog, as shown in Figure 6–60, and then click **OK**.

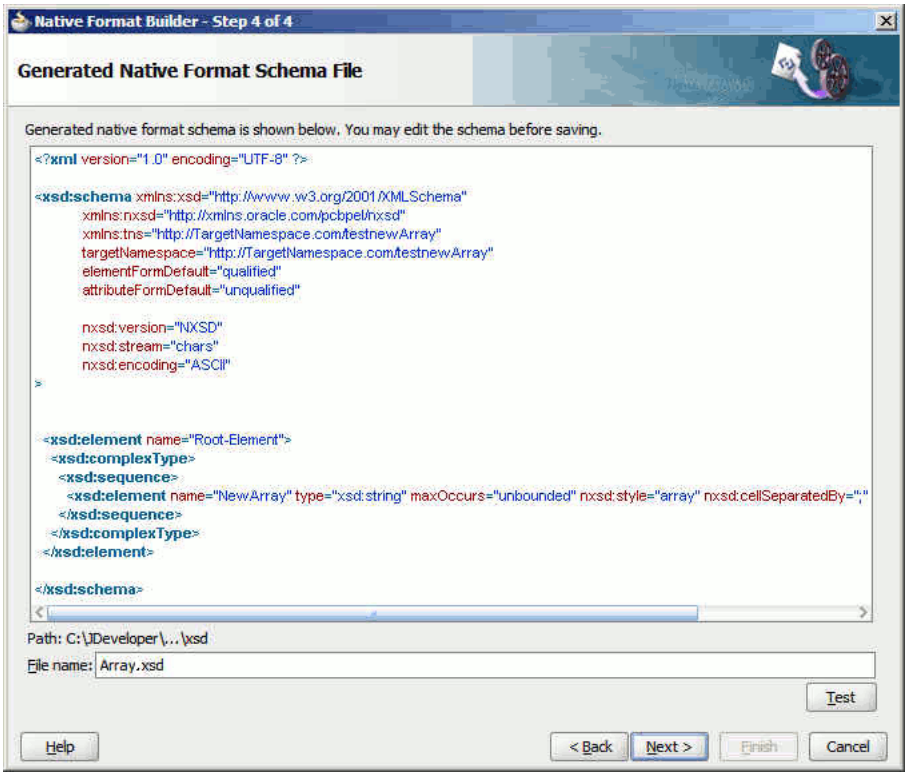
Figure 6–60 Element Details Dialog



The Native Format Builder Design Schema dialog is displayed.

10. Click **Next**. The Generated Native Format Schema File page is displayed, as shown in Figure 6–61, which displays the native format file.

Figure 6–61 Generated Native Format Schema File Page



Native Schema

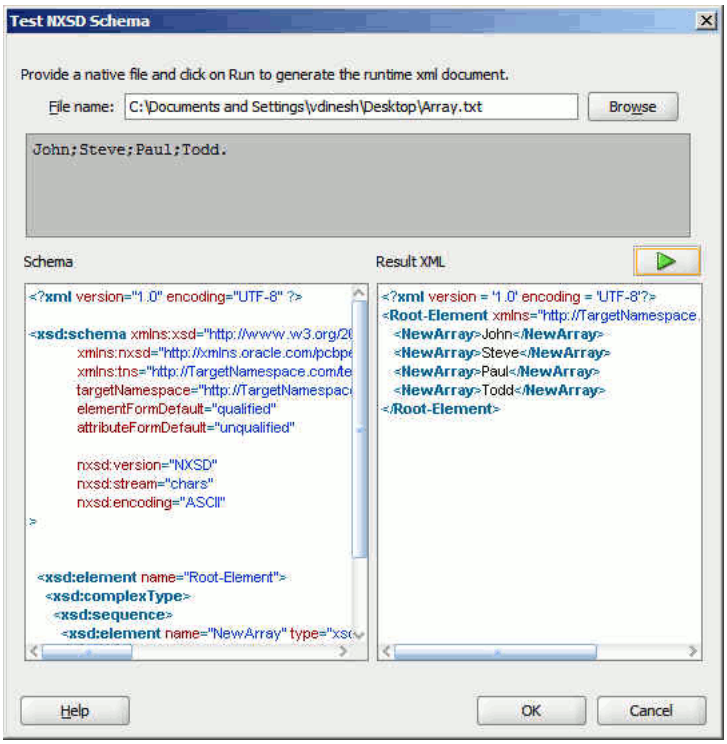
The native schema definition corresponding to the preceding native data can be defined as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            xmlns:tns="http://TargetNamespace.com/testnewArray"
            targetNamespace="http://TargetNamespace.com/testnewArray"
```

```
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            nxsd:version="NXSD"
            nxsd:stream="chars"
            nxsd:encoding="ASCII"
>
  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="NewArray" type="xsd:string" maxOccurs="unbounded"
 nxsd:style="array" nxsd:cellSeparatedBy=";" nxsd:arrayTerminatedBy="." />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

11. Click **Test**. The Test NXSD Schema dialog is displayed.
12. Click the **Generate XML** icon. The Result XML is displayed on the right pane of the Test NXSD Schema dialog, as shown in Figure 6–62.

Figure 6–62 Test NXSD Schema Dialog



Translated XML Using the Native Schema

The translated XML looks as follows:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<Root-Element xmlns="http://TargetNamespace.com/testnewArray">
   <NewArray>John</NewArray>
   <NewArray>Steve</NewArray>
   <NewArray>Paul</NewArray>
   <NewArray>Todd</NewArray>
</Root-Element>
```

13. Click **OK**. The Generated Native Format File page is displayed.
14. Click **Next**. The Native Format Builder Finish page is displayed.

6.4.8 Defining the Schema for a DTD File Structure

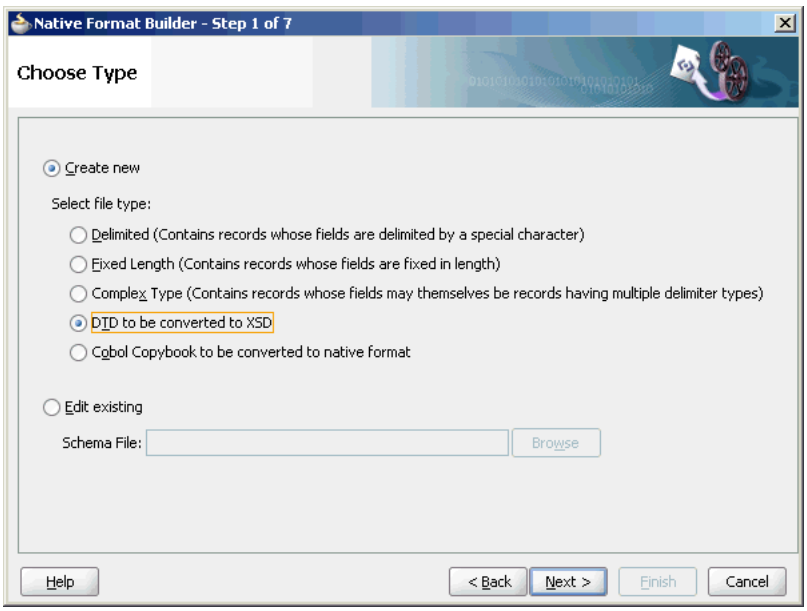
This use case takes you through the procedure for defining the schema for the native data type, DTD file.

Use the **DTD to be converted to XSD** option in the Native Format Builder wizard when creating the XML schema for this native file.

In this use case, the Native Format Builder uses a DTD file type `*.dtd`. You can generate the corresponding NXSD and also test it. Perform the following steps to run the use case:

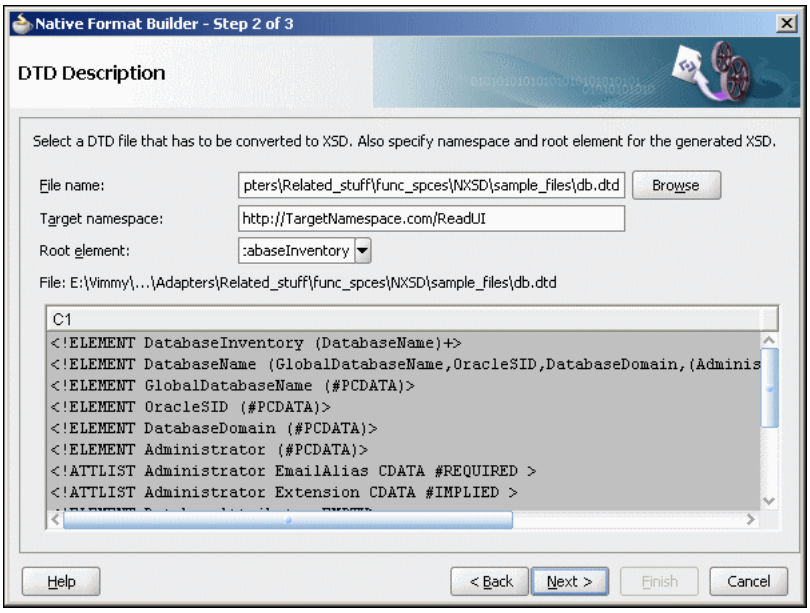
1. Use any DTD file.
2. Launch the Adapter Configuration Wizard and navigate to the Messages page, as displayed in Figure 6–4, and click **Define Schema For Native Format**. The Native Format Builder Welcome page is displayed, as shown in Figure 6–5.
3. Click **Next**. The Choose Type page is displayed, as shown in Figure 6–26.
4. Select **DTD to be converted to XSD**. The Choose Type page is displayed, as shown in Figure 6–63.

Figure 6–63 Native Format Builder Wizard Choose Type Page



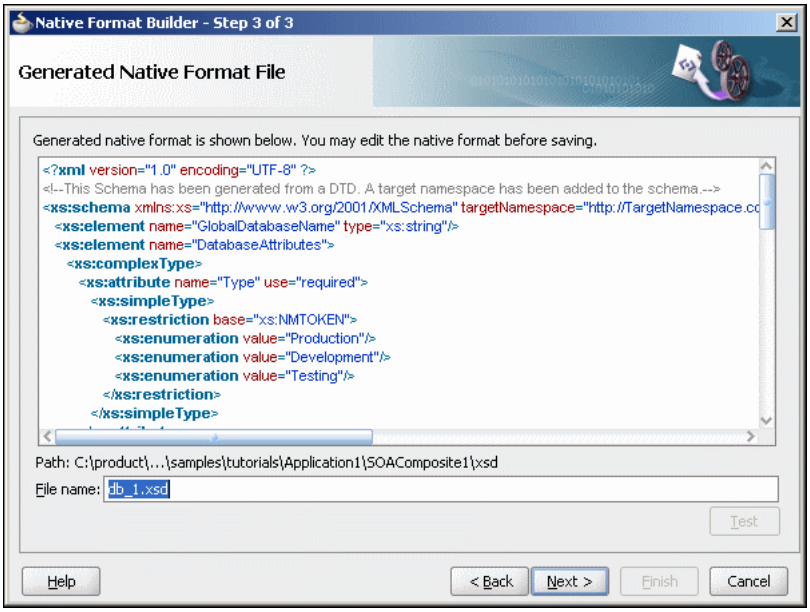
5. Click **Next**. The Native Format Builder DTD Description page is displayed.
6. Click **Browse** and select the `db.dtd` file, and select **DatabaseInventory** from the Root Element list, as displayed in Figure 6–64.

Figure 6–64 Native Format Builder Wizard File Description Page



7. Click **Next**. The Generated Native Format File page is displayed, as shown in Figure 6–65.

Figure 6–65 Native Format Builder Wizard Field Properties Page



The following is the sample native schema that is generated:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--This Schema has been generated from a DTD. A target namespace has been
added to the schema.-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://TargetNamespace.com/ReadUI"
xmlns="http://TargetNamespace.com/ReadUI" nxsd:version="DTD"
 xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd">
   <xs:element name="GlobalDatabaseName" type="xs:string"/>
```

```
  <xs:element name="DatabaseAttributes">
     <xs:complexType>
        <xs:attribute name="Type" use="required">
           <xs:simpleType>
              <xs:restriction base="xs:NMTOKEN">
                 <xs:enumeration value="Production"/>
                 <xs:enumeration value="Development"/>
                 <xs:enumeration value="Testing"/>
              </xs:restriction>
           </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="Version" use="optional" default="9i">
           <xs:simpleType>
              <xs:restriction base="xs:NMTOKEN">
                 <xs:enumeration value="7"/>
                 <xs:enumeration value="8"/>
                 <xs:enumeration value="8i"/>
                 <xs:enumeration value="9i"/>
              </xs:restriction>
           </xs:simpleType>
        </xs:attribute>
     </xs:complexType>
  </xs:element>
  <xs:element name="Comments" type="xs:string"/>
  <xs:element name="Administrator">
     <xs:complexType>
        <xs:simpleContent>
           <xs:extension base="xs:string">
              <xs:attribute name="EmailAlias" use="required"
type="xs:string"/>
              <xs:attribute name="Extension" use="optional" type="xs:string"/>
           </xs:extension>
        </xs:simpleContent>
     </xs:complexType>
  </xs:element>
  <xs:element name="OracleSID" type="xs:string"/>
  <xs:element name="DatabaseName">
     <xs:complexType>
        <xs:sequence>
           <xs:element ref="GlobalDatabaseName"/>
           <xs:element ref="OracleSID"/>
           <xs:element ref="DatabaseDomain"/>
           <xs:element maxOccurs="unbounded" ref="Administrator"/>
           <xs:element ref="DatabaseAttributes"/>
           <xs:element ref="Comments"/>
        </xs:sequence>
     </xs:complexType>
  </xs:element>
  <xs:element name="DatabaseDomain" type="xs:string"/>
  <xs:element name="DatabaseInventory">
     <xs:complexType>
        <xs:sequence>
           <xs:element maxOccurs="unbounded" ref="DatabaseName"/>
        </xs:sequence>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

8. Click **Next**. The Native Format Builder Finish page is displayed.

9. Click **Finish**. The Adapter Configuration Wizard Messages page is displayed containing the generated NXSD.

6.4.9 Defining the Schema for a COBOL Copybook File Structure

This use case shows how the Oracle File and FTP Adapters process a file in COBOL Copybook format (through use of the Native Format Builder wizard) to create a native schema file for translation.

The following COBOL Copybook examples are provided:

- Multiple Root Levels
- Single Root Level, Virtual Decimal, Fixed-Length Array
- Variable Length Array
- Numeric Types

Multiple Root Levels

A COBOL Copybook can have multiple root levels. If all root levels are at `01` level, then each such group implicitly redefines the other.

In this use case, the Native Format Builder uses a fixed-length file type, `po-ccb.cpy`, that contains the purchase order details such as buyer name, address, and items. Every element in this `po-ccb.cpy` native file has a fixed length. The data in the sample text file, `po-ccb.cpy`, appears as follows:

```
05 PO-RECORD.
10 PO-BUYER.
15 PO-UID PIC 9(7).
15 PO-NAME PIC X(15).
15 PO-ADDRESS.
20 PO-STREET PIC X(15).
20 PO-CITY PIC X(10).
20 PO-ZIP PIC 9(5).
20 PO-STATE PIC X(2).
10 PO-ITEM.
15 POITEM OCCURS 3 TIMES.
20 PO-LINE-ITEM.
25 PO-ITEM-ID PIC 9(3).
25 PO-ITEM-NAME PIC X(40).
25 PO-ITEM-QUANTITY PIC 9(2).
25 PO-ITEM-PRICE PIC 9(5)V9(2).
10 PO-TOTALPIC 9(7)V9(2).
```

You can generate the corresponding NXSD and also test it. Perform the following steps to run the use case:

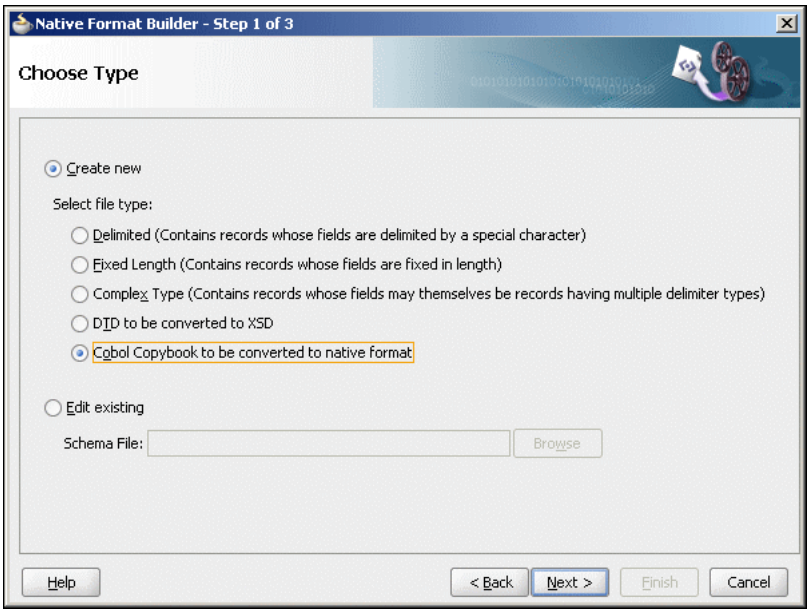
1. Get the following files from the `artifacts.zip` file contained in the `Adapters-105CobolCopyBook` sample.
   - `artifacts/samples/po-ccb.cpy`
   - `artifacts/samples/po-ebcdic.data`

   You can obtain the `Adapters-105CobolCopyBook` sample by accessing the Oracle SOA Sample Code site.

   Copy these files to your samples directory.

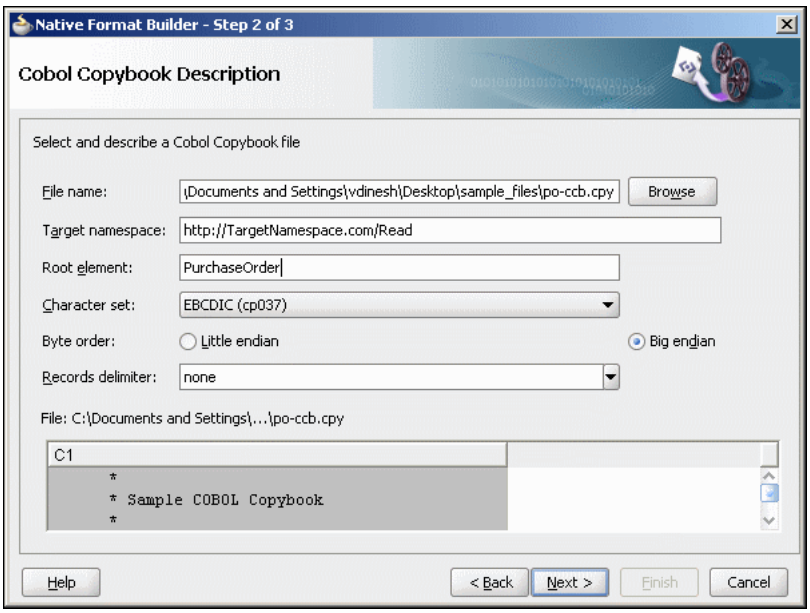
2. Launch the Adapter Configuration Wizard and navigate to the Messages page, as displayed in Figure 6–4, and click **Define Schema For Native Format**. The Native Format Builder Welcome page is displayed, as shown in Figure 6–5.
3. Click **Next**. The Choose Type page is displayed, as shown in Figure 6–26.
4. Select **Cobol Copybook to be converted to native format**. The Choose Type page is displayed, as shown in Figure 6–66.

Figure 6–66 Native Format Builder Wizard Choose Type Page



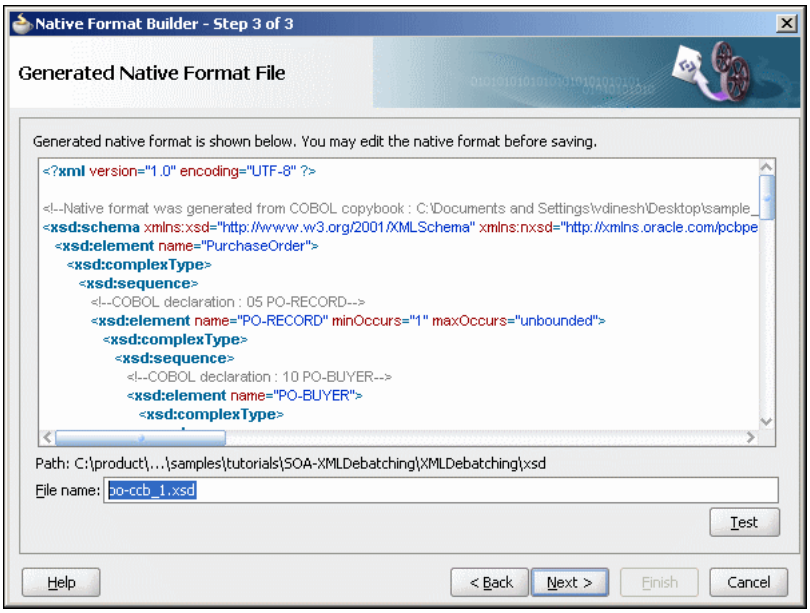
5. Click **Next**. The Native Format Builder Cobol Copybook Description page is displayed.
6. Click **Browse** and select the `po-ccb.cpy` file, as shown in Figure 6–67.

Figure 6–67 Native Format Builder Wizard File Description Page



7. Enter `PurchaseOrder` in the **Root-Element** field, and click **Next**. The Generated Native Format File page is displayed, as shown in Figure 6–68.

Figure 6–68 Native Format Builder Wizard Generated Native Format File Page



The top level payroll records are enclosed in a choice model group. Each payroll record also has two attributes, `nxsd:lookAhead` and `nxsd:lookFor` that help identify the type of record during run-time processing of the data file. So, you must add values for these attributes. For example, assume PAYROLL-F-RECORD occurs when the PAYROLL-F-TRANS-CODE field has a value of FR. The record element then looks as follows:

```
<xsd:element name="PAYROLL-F-RECORD" nxsd:lookAhead="10" nxsd:lookFor="FR">
```

The value 10 indicates the position of the lookahead field. The following COBOL Copybook has multiple root elements at the 05 level:

```
05 ORG-NUM           PIC 99.
05 EMP-RECORD.
  10 EMP-SSN       PIC 9(4)V(6).
  10 EMP-WZT       PIC 9(6).
```

Native Schema

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--Native format was generated from COBOL copybook : C:\Documents and
 Settings\vdinesh\Desktop\sample_files\po-ccb.cpy-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
 xmlns:extn="http://xmlns.oracle.com/pcbpel/nxsd/extensions"
 targetNamespace="http://TargetNamespace.com/Read"
xmlns:tns="http://TargetNamespace.com/Read" elementFormDefault="qualified"
 attributeFormDefault="unqualified" nxsd:version="NXSD" nxsd:encoding="cp037"
 nxsd:byteOrder="bigEndian" nxsd:stream="chars">
   <xsd:element name="PurchaseOrder">
      <xsd:complexType>
         <xsd:sequence>
            <!--COBOL declaration : 05 PO-RECORD-->
            <xsd:element name="PO-RECORD" minOccurs="1" maxOccurs="unbounded">
```

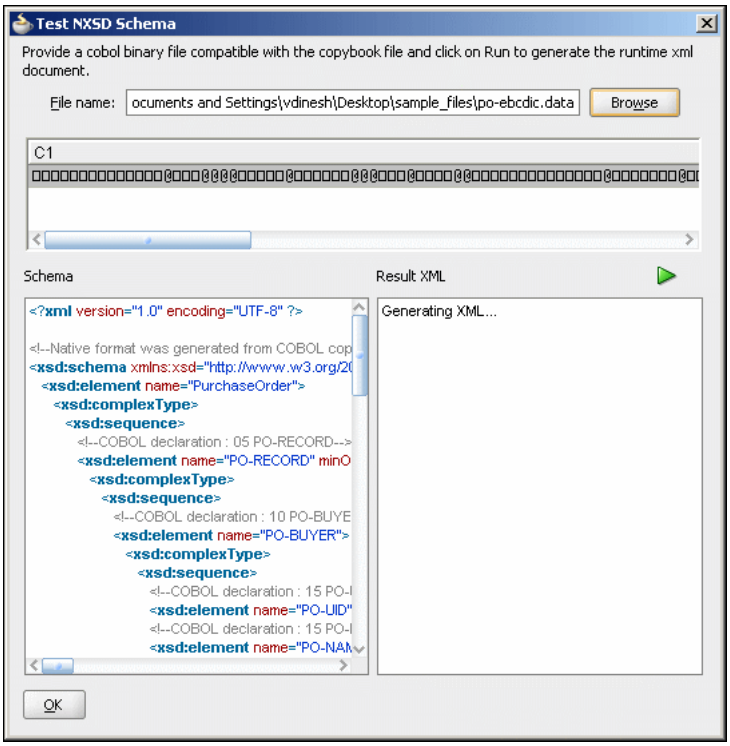
```
                    <xsd:complexType>
                       <xsd:sequence>
                          <!--COBOL declaration : 10 PO-BUYER-->
                          <xsd:element name="PO-BUYER">
                             <xsd:complexType>
                                <xsd:sequence>
                                   <!--COBOL declaration : 15 PO-UID PIC 9(7)-->
                                   <xsd:element name="PO-UID" type="xsd:long"
                                     nxsd:style="fixedLength" nxsd:padStyle="head"
                                     nxsd:paddedBy="0"
                                     nxsd:length="7"/>
                                   <!--COBOL declaration : 15 PO-NAME PIC X(15)-->
                                   <xsd:element name="PO-NAME" type="xsd:string"
                                     nxsd:style="fixedLength" nxsd:padStyle="tail"
                                     nxsd:paddedBy=" "
                                     nxsd:length="15"/>
                                   <!--COBOL declaration : 15 PO-ADDRESS-->
                                   <xsd:element name="PO-ADDRESS">
                                      <xsd:complexType>
                                         <xsd:sequence>
                                           <!--COBOL declaration : 20 PO-STREET PIC
X(15)-->
                                            <xsd:element name="PO-STREET"
                                             type="xsd:string"
                                             nxsd:style="fixedLength"
                                             nxsd:padStyle="tail"
                                             nxsd:paddedBy=" " nxsd:length="15"/>
                                            <!--COBOL declaration : 20 PO-CITY PIC
X(10)-->
                                            <xsd:element name="PO-CITY"
                                             type="xsd:string"
                                             nxsd:style="fixedLength"
                                             nxsd:padStyle="tail"
                                             nxsd:paddedBy=" " nxsd:length="10"/>
                                            <!--COBOL declaration : 20 PO-ZIP PIC
9(5)-->
                                            <xsd:element name="PO-ZIP"
                                             type="xsd:long"
                                             nxsd:style="fixedLength"
                                             nxsd:padStyle="head"
                                             nxsd:paddedBy="0" nxsd:length="5"/>
                                            <!--COBOL declaration : 20 PO-STATE PIC
 X(2)-->
                                            <xsd:element name="PO-STATE"
                                             type="xsd:string"
                                             nxsd:style="fixedLength"
                                             nxsd:padStyle="tail"
                                             nxsd:paddedBy=" " nxsd:length="2"/>
                                         </xsd:sequence>
                                      </xsd:complexType>
                                   </xsd:element>
                                </xsd:sequence>
                             </xsd:complexType>
                          </xsd:element>
                          <!--COBOL declaration : 10 PO-ITEM-->
                          <xsd:element name="PO-ITEM">
                             <xsd:complexType>
                                <xsd:sequence>
                                   <!--COBOL declaration : 15 POITEM OCCURS 3
TIMES-->
```

```
                                <xsd:element name="POITEM" minOccurs="3"
maxOccurs="3">
                                   <xsd:complexType>
                                      <xsd:sequence>
                                         <!--COBOL declaration : 20
PO-LINE-ITEM-->
                                         <xsd:element name="PO-LINE-ITEM">
                                            <xsd:complexType>
                                               <xsd:sequence>
                                 <!--COBOL declaration : 25 PO-ITEM-ID PIC
9(3)-->
                                 <xsd:element name="PO-ITEM-ID"
                                 type="xsd:long"
                                 nxsd:style="fixedLength" nxsd:padStyle="head"
                                 nxsd:paddedBy="0" nxsd:length="3"/>
                                <!--COBOL declaration : 25 PO-ITEM-NAME PIC
X(40)-->
                                 <xsd:element name="PO-ITEM-NAME"
                                 type="xsd:string"
                                 nxsd:style="fixedLength"
                                 nxsd:padStyle="tail"
                                 nxsd:paddedBy="
                                " nxsd:length="40"/>
                                 <!--COBOL declaration : 25 PO-ITEM-QUANTITY PIC
9(2)-->
                                        <xsd:element name="PO-ITEM-QUANTITY"
                                          type="xsd:long"
                                          nxsd:style="fixedLength"
                                          nxsd:padStyle="head"
                                          nxsd:paddedBy="0"
                                          nxsd:length="2"/>
                                 <!--COBOL declaration : 25 PO-ITEM-PRICE PIC
9(5)V9(2)-->
                                          <xsd:element name="PO-ITEM-PRICE"
                                           type="xsd:decimal"
                                           nxsd:style="virtualDecimal"
                                    extn:assumeDecimal="5" extn:picSize="7"/>
                                              </xsd:sequence>
                                           </xsd:complexType>
                                        </xsd:element>
                                     </xsd:sequence>
                                   </xsd:complexType>
                                </xsd:element>
                             </xsd:sequence>
                          </xsd:complexType>
                       </xsd:element>
                       <!--COBOL declaration : 10 PO-TOTAL PIC 9(7)V9(2)-->
                       <xsd:element name="PO-TOTAL"
                         type="xsd:decimal"
                         nxsd:style="virtualDecimal" extn:assumeDecimal="7"
extn:picSize="9"/>
                    </xsd:sequence>
                 </xsd:complexType>
              </xsd:element>
           </xsd:sequence>
        </xsd:complexType>
     </xsd:element>
</xsd:schema>
```

8. Click **Test**. The Test NXSD Schema dialog is displayed.

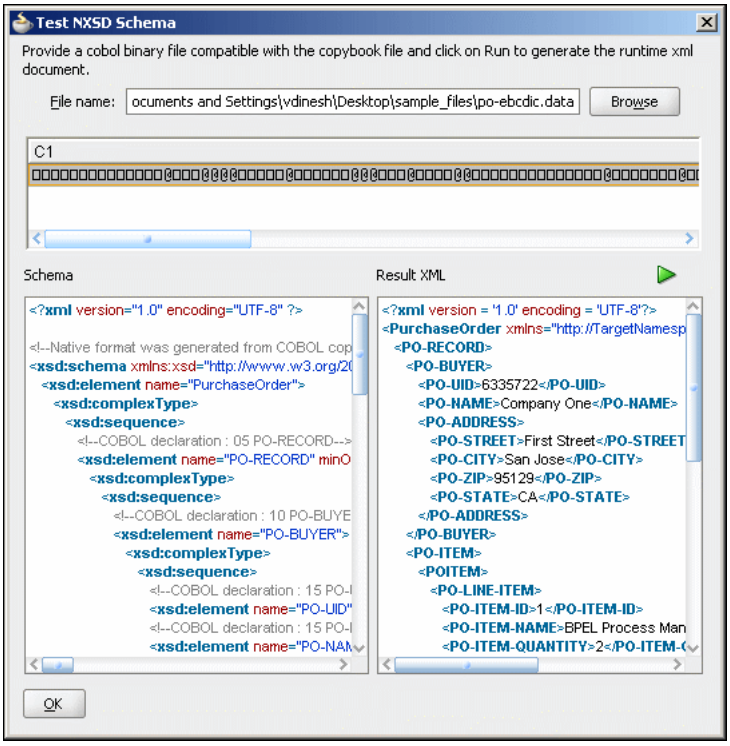
9. Click **Browse** and select the `po-ebcdic.data` file in the **File Name** field. The Test NXSD Schema dialog is displayed, as shown in Figure 6–69.

Figure 6–69 Test NXSD Schema Dialog



10. Click the **Generate XML** icon. The Result XML is displayed on the right pane of the Test NXSD Schema dialog, as shown in Figure 6–70.

Figure 6–70 Test NXSD Schema Dialog



The native data using the corresponding native schema format is translated to the following XML:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<PurchaseOrder xmlns="http://TargetNamespace.com/Read">
   <PO-RECORD>
      <PO-BUYER>
         <PO-UID>6335722</PO-UID>
         <PO-NAME>Company One</PO-NAME>
         <PO-ADDRESS>
            <PO-STREET>First Street</PO-STREET>
            <PO-CITY>San Jose</PO-CITY>
            <PO-ZIP>95129</PO-ZIP>
            <PO-STATE>CA</PO-STATE>
         </PO-ADDRESS>
      </PO-BUYER>
      <PO-ITEM>
         <POITEM>
            <PO-LINE-ITEM>
               <PO-ITEM-ID>1</PO-ITEM-ID>
               <PO-ITEM-NAME>BPEL Process Manager Enterprise
 Edition</PO-ITEM-NAME>
               <PO-ITEM-QUANTITY>2</PO-ITEM-QUANTITY>
               <PO-ITEM-PRICE>40000.0</PO-ITEM-PRICE>
            </PO-LINE-ITEM>
         </POITEM>
         <POITEM>
            <PO-LINE-ITEM>
               <PO-ITEM-ID>2</PO-ITEM-ID>
               <PO-ITEM-NAME>BPEL Process Manager Standard
 Edition</PO-ITEM-NAME>
               <PO-ITEM-QUANTITY>5</PO-ITEM-QUANTITY>
```

```
                  <PO-ITEM-PRICE>50000.0</PO-ITEM-PRICE>
               </PO-LINE-ITEM>
            </POITEM>
            <POITEM>
               <PO-LINE-ITEM>
                  <PO-ITEM-ID>3</PO-ITEM-ID>
                  <PO-ITEM-NAME>BPEL Process Manager Developer
     Edition</PO-ITEM-NAME>
                  <PO-ITEM-QUANTITY>20</PO-ITEM-QUANTITY>
                  <PO-ITEM-PRICE>20000.0</PO-ITEM-PRICE>
               </PO-LINE-ITEM>
            </POITEM>
         </PO-ITEM>
         <PO-TOTAL>730000.0</PO-TOTAL>
      </PO-RECORD>
    </PurchaseOrder>
```

In this (non-`01` level) case, an unbounded sequence of the root level items is generated.

11. Click **OK**. The Generated Native Format File page is displayed.
12. Click **Next**. The Native Format Builder Finish page is displayed.
13. Click **Finish**. The Adapter Configuration Wizard Messages page is displayed, containing the generated NXSD.

Single Root Level, Virtual Decimal, Fixed-Length Array

The following COBOL Copybook has a single root level item `PO-RECORD`. In a single root level case, the level number does not matter because the converter works in the same way. This COBOL Copybook also shows an example of a field declared as a virtual decimal (`PO-ITEM-PRICE`).

```
05 PO-RECORD.
      10 PO-BUYER.
        15 PO-UID    PIC 9(7).
        15 PO-NAME   PIC X(15).
        15 PO-ADDRESS.
            20 PO-STREET PIC X(15).
            20 PO-CITY   PIC X(10).
            20 PO-ZIP    PIC 9(5).
            20 PO-STATE  PIC X(2).
      10 PO-ITEM.
        15 POITEM OCCURS 3 TIMES.
            20 PO-LINE-ITEM.
              25 PO-ITEM-ID       PIC 9(3).
              25 PO-ITEM-NAME     PIC X(40).
              25 PO-ITEM-QUANTITY PIC 9(2).
              25 PO-ITEM-PRICE    PIC 9(5)V9(2).
      10 PO-TOTAL PIC  9(7)V9(2).
```

The generated schema looks as follows:

```
<?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 looks as follows:

```
<?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 looks as follows:

```
<?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 by using different layouts, the generated schema requires modification.

6.5 Command Line Tool for Testing NXSD Translator

You might want to test your nXSD schema to ensure that nXSD annotations are correct and that generated XML/native data conforms to your business semantics. If you want to do that currently, you must write a BPEL process with an inbound or outbound File Adapter partner link, or both, configured with the appropriate nXSD schema and test them on the SOA server. This is both time-consuming and error prone.

A simple standalone test client that can enable you to verify your nXSD schemas.

6.5.1 Prerequisites

Before you use the test client to verify your nXSD schemas, add the following jars in the classpath. These jars (except for `test-translator.jar`) are available as a part of your SOA installation. You must use Java 6 to run the test client.

- `bpm-infra.jar`. This is the nXSD run-time jar available under `$SOA_HOME/soa/modules/oracle.soa.fabric_11.1.1`
- `xmlparserv2.jar`. This is the Oracle XDK library for parsing available under `$FMW_HOME/oracle_common/modules/oracle.xdk_11.1.0`
- `xml.jar` This is the Oracle XDK library for schema validation available under `$FMW_HOME/oracle_common/modules/oracle.xdk_11.1.0`
- `mail.jar` .This is the Java mail API.
- `test-translator.jar`. You must rename the extension from jarr to jar.

6.5.2 Running the Test Tool

Now you can run `java xlator.util.Translate -help` and the usage should be displayed as shown below. When supply the `-help` option, the tool supplies a list of options and defaults.

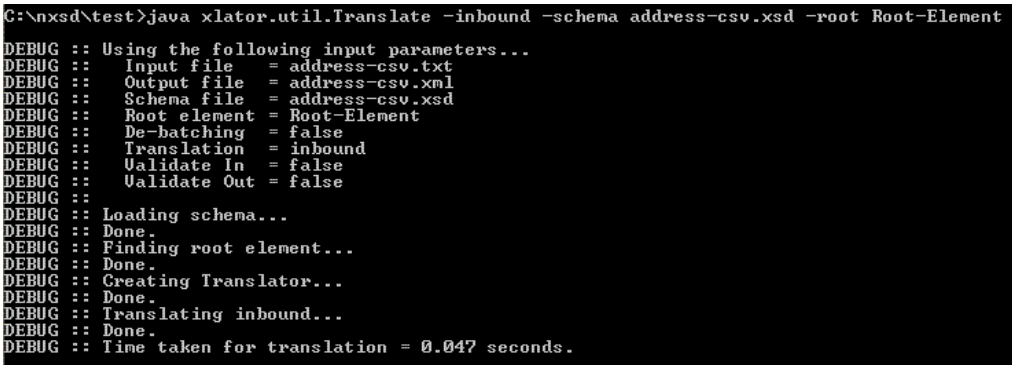
Figure 6–71 Running java xlator.util.Translate -help

```
C:\nxsd\test>java xlator.util.Translate -help

usage: java xlator.util.Translate (options) (Input File)

options:
  -schema (Native Schema)       The schema to use for translation.
  -root (name)                  Localname of the root element declaration to use.
  -inbound  ¦ -outbound         Do inbound/outbound translation.
  -output (Output File)         The translated file.
  -debatch (Publish Size)       Turn debatching on with the specified publish size.
  -validate (in ¦ out ¦ both)   Turn on xml validation of the input/output/both file.
  -debug (on ¦ off)             Turn on/off debug statements.
  -help                         This help screen.

defaults:
  Validation:  off
  De-batching: off
  Debug: on
```

The following example sample execution of the test client converts the `address-csv.txt` file to address-csv.xml.

The command is:`java xlator.util.Translate -inbound -schema address-csv.xsd -root Root-Element -input address-csv.txt -output address-csv.xml`

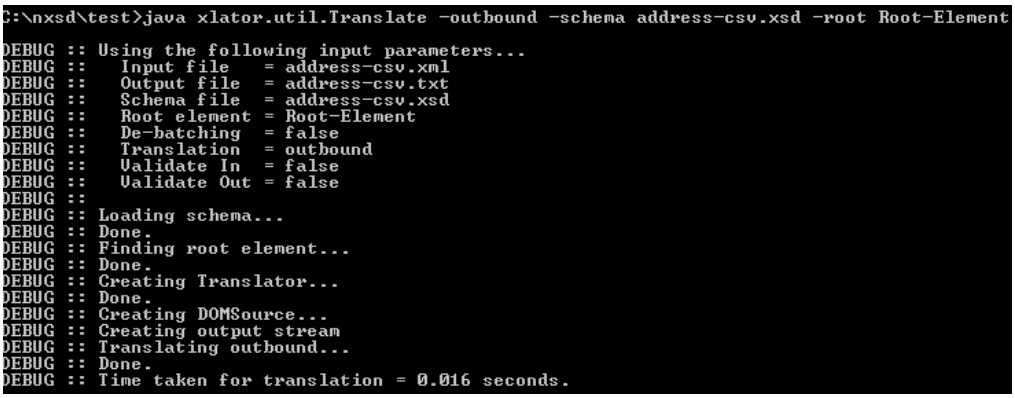
Figure 6–72 Using the Test Tool to Convert txt to xml



Sample execution of the test client to convert `address-csv.xml` to `address-csv.txt`:

```
java xlator.util.Translate -outbound -schema address-csv.xsd -root Root-Element
-input address-csv.xml -output address-csv.txt
```

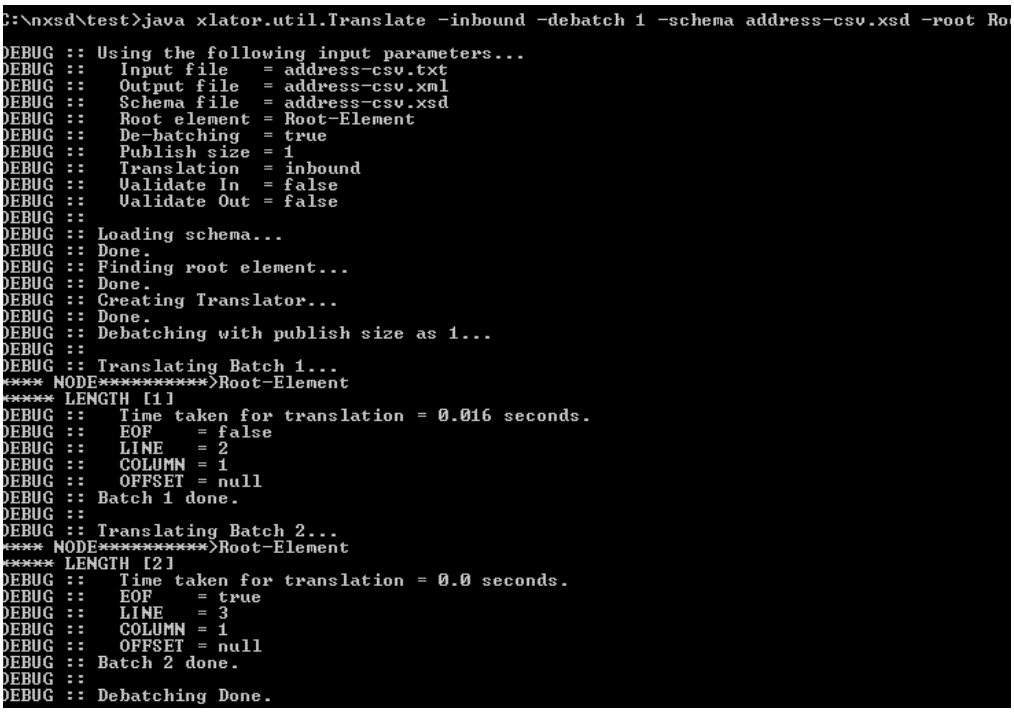
Figure 6–73 Using the Test Tool to Convert xml to txt



The following sample execution of the test client converts `address-csv.txt` to a series of `address-csv.xml_batch_%SEQ%.xml files` using de-batching.

```
java xlator.util.Translate -inbound -debatch 1 -schema address-csv.xsd -root
Root-Element -input address-csv.txt -output address-csv.xml
```

Figure 6–74 Using the Test Tool to Convert address-csv.txt to a Series of batch xml Files



Part II

Message Adapters

Part II contains the following chapters:

- Chapter 7, "Oracle JCA Adapter for AQ"
- Chapter 8, "Oracle JCA Adapter for JMS"
- Chapter 9, "Oracle JCA Adapter for Database"
- Chapter 10, "Oracle JCA Adapter for MQ Series"
- Appendix A, "Oracle JCA Adapter Properties"
- Appendix B, "Oracle JCA Adapter Valves"

7

Oracle JCA Adapter for AQ

This chapter describes how to use the Oracle JCA Adapter for AQ (Oracle AQ Adapter), which enables an Oracle BPEL Process Manager (Oracle BPEL PM) or an Oracle Mediator to interact with a single consumer or a multiconsumer queue.

This chapter includes the following sections:

- Section 7.1, "Introduction to the Oracle AQ Adapter"
- Section 7.2, "Oracle AQ Adapter Features"
- Section 7.3, "Deployment"
- Section 7.4, "Oracle AQ Adapter Use Cases"

7.1 Introduction to the Oracle AQ Adapter

Oracle Streams Advanced Queuing (AQ) provides a flexible mechanism for bidirectional, asynchronous communication between participating applications. Advanced queues are an Oracle database feature, and are therefore scalable and reliable. Other features of Oracle database, such as backup and recovery (including any-point-in-time recovery), logging, transactional services, and system management, are also inherited by advanced queues. Multiple queues can also service a single application, partitioning messages in a variety of ways and providing another level of scalability through load balancing.

This section includes the following sections:

- Section 7.1.1, "Oracle AQ Adapter Integration with Oracle BPEL Process Manager"
- Section 7.1.2, "Oracle AQ Adapter Integration with Oracle Mediator"

For more information on Oracle AQ, see "Introduction to Oracle Streams AQ" in the *Oracle Streams Advanced Queuing User's Guide*.

7.1.1 Oracle AQ Adapter Integration with Oracle BPEL Process Manager

JCA Binding Component is used for the bidirectional integration of the JCA 1.5 resource adapters with Oracle BPEL Process Manager. JCA Binding Component is based on standards and employs the Web service Invocation Framework (WSIF) technology for exposing the underlying JCA interactions as Web services.

For more information about Oracle AQ Adapter architecture, adapter integration with Oracle BPEL Process Manager, and adapter deployments, see Chapter 3, "Adapter Integration with Oracle Application Server Components."

7.1.2 Oracle AQ Adapter Integration with Oracle Mediator

The Mediator Server supports Oracle AQ Adapter and enables you to define inbound and outbound adapter services for each. An inbound adapter service receives data from an Oracle AQ Adapter and transforms it into an XML message. An outbound adapter service sends data to a target application by transforming an XML message into the native format of the given adapter.

Using the Mediator Server, you can send or receive messages from Oracle Advanced Queuing single or multiconsumer queues.

Note: Oracle BPEL PM pre-dates Mediator and most of this guide and the samples implicitly assume use with Oracle BPEL PM. However, the Oracle AQ Adapter works equally well with either Oracle BPEL PM or Mediator. For any mention of Oracle BPEL PM here, you may substitute Mediator, instead.

7.2 Oracle AQ Adapter Features

The Oracle AQ Adapter is both a producer and a consumer of AQ messages. The enqueue operation is exposed as a JCA outbound interaction. The dequeue operation is exposed as a JCA inbound interaction.

The Oracle AQ Adapter supports ADT (Oracle object type), `XMLType`, and `RAW` queues as payloads. It also supports extracting a payload from one ADT member column.

The Oracle AQ Adapter supports normalized properties for enqueue and dequeue operations.

For more information about the properties supported by Oracle AQ Adapter, see Appendix A.3, "Oracle AQ Adapter Properties."

You can obtain the Oracle AQ Adapter samples by accessing the Oracle SOA Sample Code site.

This section includes the following topics:

- Section 7.2.1, "Enqueue-Specific Features (Message Production)"
- Section 7.2.2, "Dequeue and Enqueue Features"
- Section 7.2.3, "Supported ADT Payload Types"
- Section 7.2.4, "Native Format Builder Wizard"
- Section 7.2.5, "Normalized Message Support"
- Section 7.2.6, "Is DOM 2 Compliant"
- Section 7.2.7, "Is Message-Size Aware"
- Section 7.2.8, "Multiple Receiver Threads"
- Section 7.2.9, "DequeueTimeout Property"
- Section 7.2.10, "Control Dequeue Timeout and Multiple Inbound Polling Threads"
- Section 7.2.11, "Stream Payload Support"
- Section 7.2.12, "Oracle AQ Adapter Inbound Retries"
- Section 7.2.13, "Error Handling Support"
- Section 7.2.14, "Performance Tuning"

7.2.1 Enqueue-Specific Features (Message Production)

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

- Correlation Identifier

  In the Adapter Configuration Wizard, you can specify a correlation identifier when defining an enqueue operation, which you use to retrieve specific messages.

- Multiconsumer Queue

  In Oracle Streams AQ, multiple consumers 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 multiple AQ consumer (dequeue operation), either through the default subscription list or with an override recipient list. Under this scenario, a message remains in the queue until it is consumed by all of its intended consumer agents. The Oracle AQ Adapter enqueue header property (`jca.aq.RecipientList`) enables you to specify the override recipient list (string values separated by commas) that can retrieve messages from a queue. All consumers that are added as subscribers to a multiconsumer queue must have unique values for the `Recipient` parameter. Two subscribers cannot have the same values for the `NAME`, `ADDRESS`, and `PROTOCOL` attributes.

- Message Priority

  If you specify the priority of enqueued messages, then the messages are dequeued in priority order. If two messages have the same priority, then the order in which they are dequeued is determined by the enqueue time. You can also create a first-in, first-out (FIFO) priority queue by specifying the enqueue time priority as the sort order of the messages. This priority is a property of the Oracle AQ Adapter enqueue header. The enqueue time is set automatically by the underlying AQ application.

  Here is an example of how to create the FIFO queue:

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE( \
queue_table => 'OE_orders_pr_mqtab', \
sort_list =>'priority,enq_time', \
comment => 'Order Entry Priority \
MultiConsumer Orders queue table',\
multiple_consumers => TRUE, \
queue_payload_type => 'BOLADM.order_typ', \
compatible => '8.1', \
primary_instance => 2, \
secondary_instance => 1);
EXECUTE DBMS_AQADM.CREATE_QUEUE ( \
queue_name => 'OE_bookedorders_que', \
queue_table => 'OE_orders_pr_mqtab');
```

- Time Specification and Scheduling

  In Oracle Streams AQ, you can specify a delay interval and an expiration interval. The delay interval determines when an enqueued message is marked as available to the dequeuers after the message is enqueued. When a message is enqueued with a delay time set, the message is marked in a `WAIT` state. Messages in a `WAIT` state are masked from the default dequeue calls. The expiration time property is used to specify an expiration time, and the message is automatically moved to an exception queue if the message is not consumed before its expiration.

7.2.2 Dequeue and Enqueue Features

Oracle Streams AQ provides the following dequeuing options:

- Poll option
- Notification option

The poll option involves processing the messages as they arrive and polling repeatedly for messages. The Oracle AQ Adapter supports a polling mechanism for consuming AQ messages.

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

- Multiconsumer Queue

  The Oracle 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 Oracle AQ Adapter dequeue header. If the number of retries exceeds the limit, then the message is moved to an exception queue that you specify. The exception queue is a property of the Oracle AQ Adapter enqueue header.

- Rule-Based Subscription

  Oracle Streams AQ provides content-based message filtering and subject-based message filtering. A rule defines one or more consumers' interest in subscribing to messages that conform to that rule. For a subject-based rule, you specify a Boolean expression using syntax similar to the `WHERE` clause of a SQL query. This Boolean expression can include conditions on message properties (current priority and correlation ID), user data properties (object payloads only), and functions (as specified in the `WHERE` clause of a SQL query).

- Oracle AQ Adapter Header Properties

  For more information about Oracle AQ Adapter header properties, see Appendix A.3, "Oracle AQ Adapter Properties."

- Dequeue Condition

  The Dequeue condition is an advanced queuing product feature that Oracle AQ Adapter uses. If a dequeue condition is specified and no messages meet the specified condition, then no dequeue happens.

  A dequeue condition element is a Boolean expression using syntax similar to the `WHERE` clause of a SQL query. This Boolean expression can include conditions on message properties, user object payload data properties, and PL/SQL or SQL functions. Message properties include `priority`, `corrid`, and other columns in the queue table.

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

The Message Selector Rule field is displayed in Step 15 if you select a multisubscriber queue. Enter a subscription rule in the form of a Boolean expression using syntax similar to a SQL `WHERE` clause, such as `priority = 1`, or `TAB.USER_DATA.amount > 1000`. The adapter dequeues only those messages for which this Boolean expression is true.

You must select the **Access to non-payload fields also needed** check box to access header information.

When this check box is selected, the generated WSDL file has additional code in the `type` section:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<?binding.jca Inbound_aq.jca?>
<definitions name="Inbound"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT"
xmlns:imp1="http://www.oracle.com/ipdemo">
    <types>
       <schema attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"
xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT">
            <import namespace="http://xmlns.oracle.com/xdb/SCOTT"
schemaLocation="xsd/SCOTT_MAGAZINE_TYPE.xsd"/>
            <import
namespace="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"
schemaLocation="xsd/aqAdapterInboundHeader.xsd"/>
            <complexType name="HeaderCType">
                <sequence>
                    <element name="QueueHeader" type="hdr:HeaderType"/>
                    <element name="PayloadHeader" type="obj1:MAGAZINE_TYPE"/>
                </sequence>
            </complexType>
            <element name="Header" type="tns:HeaderCType"/>
        </schema>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://www.oracle.com/ipdemo"
schemaLocation="xsd/simpleMagazine.xsd"/>
        </schema>
    </types>
    <message name="simpleMagazine_msg">
        <part name="simpleMagazine" element="imp1:simpleMagazine"/>
    </message>
    <message name="Header_msg">
        <part name="Header" element="tns:Header"/>
    </message>
    <portType name="Dequeue_ptt">
        <operation name="Dequeue">
            <input message="tns:simpleMagazine_msg"/>
        </operation>
    </portType>
    <plt:partnerLinkType name="Dequeue_plt">
        <plt:role name="Dequeue_role">
            <plt:portType name="tns:Dequeue_ptt"/>
        </plt:role>
```

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

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

For more information about Oracle AQ Adapter architecture, adapter integration with Oracle BPEL Process Manager, and adapter deployments, see Chapter 1, "Introduction to Oracle JCA Adapters."

7.2.3 Supported ADT Payload Types

The Oracle AQ Adapter supports the following `RAW` types:

- `BLOB`
- `CHAR`
- `CLOB`
- `DATE`
- `DECIMAL`
- `DOUBLE PRECISION`
- `FLOAT`
- `INTEGER`
- `NUMBER`
- `REAL`
- `SMALLINT`
- `TIMESTAMP`
- `VARCHAR2`

In addition to the `RAW` types mentioned in the preceding list, the Oracle AQ Adapter supports primitive types and varrays of objects.

Note: The Oracle AQ Adapter does not currently support the following data types for ADT columns: `TIMESTAMP WITH LOCAL TIMEZONE` and `TIMESTAMP WITH TIMEZONE`.

If you choose a payload field instead of the whole ADT, then choose one of the following data types as the payload field:

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

7.2.4 Native Format Builder Wizard

JDeveloper BPEL Designer provides the Native Format Builder Wizard to define XSD files of various formats, including for the AQ RAW payload.

For more information about the Native Format Builder wizard, see Chapter 6, "Native Format Builder Wizard."

To obtain sample code that demonstrates usage of the Native Format Builder access the Oracle SOA Sample Code site.

Payload Schema

The payload schemas depend on the payload type. In the whole ADT case, the schema is completely generated by the Adapter Configuration Wizard. In an ADT case where the payload case selected is BLOB, an opaque schema as defined in the following example must be used:

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

In all other cases, you can either provide a schema or use an opaque schema, as shown in Table 7–1.

Table 7–1 Payload Schema

Payload Type	Supported Schema
RAW	User-provided schema or opaque schema.
Whole ADT	Must use a schema generated by the Adapter Configuration Wizard, which is based on the queue structure.
ADT with VARCHAR2 picked as payload	User-provided schema or opaque schema.
ADT with CLOB picked as payload user-provided schema or opaque schema	User-provided schema or opaque schema.
ADT with BLOB picked as payload opaque schema	Opaque schema.
XMLTYPE	User-provided schema or opaque schema.

If you do not have an XSD file but the payload data is formatted (for example, in a comma-delimited value (CSV) format), you can use the Native Format Builder wizard to generate an appropriate XSD. The Adapter Configuration Wizard is integrated with the Native Format Builder wizard. In the Adapter Configuration Wizard Messages window, click **Define Schema for Native Format** to access the Native Format Builder wizard.

7.2.5 Normalized Message Support

Header manipulation and propagation is a key business integration messaging requirement. Oracle BPEL PM, Mediator, Oracle JCA, and B2B rely extensively on header support to solve customers' integration needs. For example, you can preserve a file name from the source directory to the target directory by propagating it through message headers. In Oracle BPEL PM and Mediator, you can access, manipulate, and set headers with varying degrees of UI support.

Note: AQ Adapter inbound and outbound headers supported in the 10.1.3 release are supported in 11*g* through normalized message properties.

For more information, see Section 2.14, "Correlation Support Within Adapters"

Propagating Headers in a Normalized Message:

A normalized message is simplified to have only two parts, properties and payload.Typically, properties are name-value pairs of scalar types. To fit the existing complex headers into properties, properties are flattened into scalar types.

Manipulating Headers in Design Time:

The user experience is simplified while manipulating headers in design time, because the complex properties are predetermined. In the Mediator or BPEL designer, you can manipulate the headers with some reserved key words. For example, currently in Mediator, you can access an inbound File adapter, *fileName* header using the following expression:

```
$nmproperty.InboundFileHeaderType.fileName
```

However, this method does not address the properties that are dynamically generated based on your input. For example, in the AQ Adapter Wizard, you can propagate some fields from an AQ object as headers. Based on your choice, the header definitions are defined. These definitions are not predetermined and hence cannot be accounted for in the list of predetermined property definitions. You cannot design header manipulation of the dynamic properties before they are defined. To address this limitation, you must generate all the necessary services (composite entry points) and references. This restriction applies to services that are expected to generate dynamic properties. Once dynamic properties are generated, they must be stored for each composite. Only then you can manipulate the dynamic properties in Mediator or BPEL designer.

Identifying Properties That Must Be Propagated over the Life Cycle of the Normalized Message:

Some properties must be propagated across the life cycle of the message, whereas some must not be propagated. The properties that must be propagated are referred to as *propagatable* properties, whereas properties that must not be propagated are referred to as *non-propagatable* properties.

7.2.6 Is DOM 2 Compliant

Oracle AQ Adapter is Document Object Model Level 2 (DOM 2) compliant, that is, the AQ adapter can generate document objects that are compliant with DOM2 specification.

7.2.7 Is Message-Size Aware

Oracle AQ Adapter is message-size aware, that is, Oracle AQ Adapter calculates the message size and reports the size back to JCA Binding Component. The API, related to size, exposed by JCA Binding Component can be used for reporting purposes.

7.2.8 Multiple Receiver Threads

Oracle AQ Adapter supports an activation endpoint property, `"adapter.aq.dequeue.threads"`. Setting this property is a preferred way to spawn multiple threads for the inbound message flow between the adapter and the Enterprise Information System (EIS). Earlier versions of the Oracle AQ Adapter relied on the `activationInstances` endpoint property, which was used by JCA Binding Component to initiate multiple endpoints.

7.2.9 DequeueTimeout Property

The `DequeueTimeOut` property supports multiple inbound dequeue threads. The value for this property determines how many seconds the `dequeue()` API waits for messages before it returns and the next polling cycle begins.

Add this property to the `composite.xml` file, as shown in the following example:

```
<service name="Inbound" ui:wsdlLocation="Inbound.wsdl">
<interface.wsdl interface="http://xmlns.oracle.com/pcbpel/adapter/aq/AQ_
InboundRetry_Mediator/AQ2JMSInboundRetry/Inbound%2F#wsdl.interface(Dequeue_ptt)"/>
<binding.jca config="Inbound_aq.jca">
<property name="DequeueTimeOut" type="xs:integer" many="false"override="may">10
</property>
</binding.jca>
</service>
```

7.2.10 Control Dequeue Timeout and Multiple Inbound Polling Threads

Oracle AQ Adapter provides system properties to control dequeue timeout and multiple inbound polling threads for each Java Virtual Machine (JVM), systemwide, instead of for each process.

The system property provided by Oracle AQ Adapter to control dequeue timeout is `oracle.adapter.aq.wait`, and the property that controls inbound polling threads is `adapter.aq.dequeue.threads`.

7.2.11 Stream Payload Support

Oracle AQ Adapter provides support to stream payload. When you enable this feature, the payload is streamed to a database instead of getting manipulated in SOA run time as in a memory DOM. You use this feature while handling large payloads. To enable support to stream payload, you must select the Enable Streaming check box while defining the dequeue operation parameters in Oracle JDeveloper (JDeveloper). When you select the Enable Streaming check box, a corresponding Boolean property `EnableStreaming` is appended to the `ActivationSpec` properties defined in the respective `.jca` file, as shown in the following example. If the EnableStreaming property does not exist, then the default value `false` is assumed. The property is applicable when processing `Raw` messages, `XMLType` messages, and `ADT` type messages for which a payload is specified though an `ADT` attribute.

```
<activation-spec
className="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec">
      <property name="QueueName" value="RAW_IN_QUEUE"/>
      <property name="DatabaseSchema" value="SCOTT"/>
      <property name="EnableStreaming" value="true"/>
</activation-spec>
```

7.2.12 Oracle AQ Adapter Inbound Retries

If you configure the Oracle AQ Adapter inbound retries to retry for more than 5 times by using the `jca.retry.count` service binding property for a retryable exception, then ensure that the queue is created with max_retries value that is greater then the value used for `jca.retry.count`. If nothing is specified, then the queue is created with a `max_retries` value of 5 which would mean that the message ends up in the exception queue after 5 retries and is not be delivered to adapter for further

processing. If `jca.retry.count` is specified with a value of 5 or less, then you do not have to change the queue `max_retries` property.

Use the following code to change the `max_retries` property when creating a queue:

```
begin
DBMS_AQADM.CREATE_QUEUE_TABLE ( queue_table => 'RAW_IN_QUEUE_TABLE',queue_payload_
type => 'RAW');
DBMS_AQADM.CREATE_QUEUE ( queue_name => 'RAW_IN_QUEUE',queue_table=> 'RAW_IN_
QUEUE_TABLE', max_retries=>1500);
DBMS_AQADM.START_QUEUE ( queue_name => 'RAW_IN_QUEUE');
DBMS_AQADM.CREATE_QUEUE_TABLE ( queue_table => 'RAW_OUT_QUEUE_TABLE', queue_
payload_type => 'RAW');
DBMS_AQADM.CREATE_QUEUE ( queue_name  => 'RAW_OUT_QUEUE', queue_table => 'RAW_OUT_
QUEUE_TABLE');
DBMS_AQADM.START_QUEUE ( queue_name => 'RAW_OUT_QUEUE');
end;
```

7.2.13 Error Handling Support

For information about error handling, see Section 2.21, "Error Handling."

7.2.14 Performance Tuning

Oracle AQ Adapter supports performance tuning features.

For more information, see "Oracle AQ Adapter Tuning" in the Oracle Fusion Middleware Performance and Tuning Guide.

7.3 Deployment

The Oracle AQ Adapter comes deployed to the application server as part of the install. It contains a single adapter instance entry `eis/AQ/aqSample`, which points to the data source `jdbc/aqSample`. The data source is not created as part of install and must be created manually. The connection information to the database is inside the data source definition.

When deploying a SOA project that uses the Oracle AQ Adapter instance `eis/AQ/aqSample` that exists at the time of installation, you must first create a data source at `jdbc/aqSample`. On the other hand, if a new adapter instance is preferred, then you must add a new adapter instance and restart the application server. This is because you want to point to a data source other than the one referred in the existing adapter instance `jdbc/aqSample`, or because you chose a name for the adapter instance that does not yet exist. For instance, if you create a connection in JDeveloper named `DBConnection1`, then by default the AQ Adapter service points to `eis/AQ/DBConnection1`, as shown in Figure 7–6.

You can also check which adapter instance the service is pointing to by looking at the `.jca` file, as shown in the following code snippet:

```
<connection-factory location="eis/AQ/aqSample" … />
```

In the preceding example, the location is the JNDI name of the adapter instance at run time.

You can create a new AQ Adapter instance through the Oracle WebLogic Server Administration Console, as mentioned in Section 2.18, "Adding an Adapter Connection Factory" or by directly editing the `weblogic-ra.xml` file. The following are the steps to edit `weblogic-ra.xml`:

1. Search `fmwhome/` for `AqAdapter.rar`.
2. Unzip the file.
3. Edit `META-INF/weblogic-ra.xml` (and possibly `ra.xml`.)
4. Jar the file again.
5. Restart the application server.

The following is a sample adapter instance in `weblogic-ra.xml`:

```
<connection-instance>
  <jndi-name>eis/AQ/aqSample</jndi-name>
  <connection-properties>
    <properties>
      <property>
         <name>XADataSourceName</name>
         <value>jdbc/aqSample</value>
      </property>
      <property>
    <name>DataSourceName</name>
    <value></value>
      </property>
      <property>
    <name>ConnectionString</name>
    <value></value>
      </property>
      <property>
    <name>UserName</name>
    <value></value>
      </property>
      <property>
    <name>Password</name>
    <value></value>
      </property>
      <property>
    <name>DefaultNChar</name>
    <value>false</value>
      </property>
      <property>
    <name>UseDefaultConnectionManager</name>
    <value>false</value>
      </property>
    </properties>
  </connection-properties>
</connection-instance>
```

The mandatory properties are: `jndi-name`, `XADataSourceName` or `DataSourceName`. The `jndi-name` property must match the location attribute in the `.jca` file, and is the name of the adapter instance. The `XADataSourceName` or `DataSourceName` property is the name of the underlying data source (which has the connection information). Specify one of the properties `XADataSourceName` or `DataSourceName`. The usage depends on if the scenario involves and would require adapter to participate in global transaction or if local transaction semantics are sufficient. In the former case `XADataSourceName` must be specified while in the latter case `DataSourceName` must be specified. When specifying `XADataSourceName` property ensure that the physical data source it refers to is XA enabled while when specifying `DataSourceName` property the physical data source it refers to might or might not be XA enabled.

Most Common Mistakes

The following are the two most common mistakes with deployment:

- Not creating an adapter instance entry that matches the location attribute in your `.jca` file (or not creating one at all.)
- Setting the location attribute in the `.jca` file to the name of the data source directly.

For the latter, there is a level of indirection in that you give the name of the adapter instance (`eis/AQ/...`), which itself points to the data source pool (`jdbc/...`). It is a common mistake to miss this indirection and give the name `jdbc/...` directly in the location attribute.

Additional Adapter Instance Properties

There are additional properties in the AQ Adapter instance beyond `xADataSourceName`, `dataSourceName`.

For information about the Oracle AQ Adapter instance properties, see Appendix A.3, "Oracle AQ Adapter Properties."

7.4 Oracle AQ Adapter Use Cases

This section includes the following topics:

- Section 7.4.1, "Generic Use Case"
- Section 7.4.2, "Oracle AQ Adapter ADT Queue"
- Section 7.4.3, "Oracle AQ Adapter RAW Queue"

7.4.1 Generic Use Case

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

This section includes the following topics:

- Section 7.4.1.1, "The Adapter Configuration Wizard Walkthrough"
- Section 7.4.1.2, "Dequeuing and Enqueuing Object and ADT Payloads"
- Section 7.4.1.3, "Dequeuing One Column of the Object Payload"
- Section 7.4.1.4, "Configuring the Enqueue/Dequeue Operation Type"
- Section 7.4.1.5, "Using Correlation ID for Filtering Messages During Dequeue"
- Section 7.4.1.6, "Enqueuing and Dequeuing from Multisubscriber Queues"

7.4.1.1 The Adapter Configuration Wizard Walkthrough

In this example, you create an Oracle AQ Adapter service that dequeues messages to the `service_in_queue` queue, with a payload that is one field within the `service_type` object, and with a user-defined schema.

This section describes the tasks required to configure Oracle AQ Adapter by using the Adapter Configuration Wizard in JDeveloper.

This section includes the following topics:

- Section 7.4.1.1.1, "Meeting Prerequisites"
- Section 7.4.1.1.2, "Creating an Application and an SOA Project"
- Section 7.4.1.1.3, "Defining an Oracle AQ Adapter Service"
- Section 7.4.1.1.4, "Generated WSDL and JCA Files"

7.4.1.1.1 Meeting Prerequisites

This example assumes that you are familiar with basic BPEL constructs, such as activities and partner links, and JDeveloper environment for creating and deploying BPEL composite.

You must have access to a database with the SCOTT schema.

To perform this use case, you require the following files from the `artifacts.zip` file contained in the `adapters-aq-103-adtclobpayload` sample:

- `artifacts/sql/setup_user.sql`
- `artifacts/sql/create_type_service.sql`
- `artifacts/sql/create_queues.sql`
- `artifacts/sql/dequeue_service.sql`
- `artifacts/sql/enqueue_service.sql`

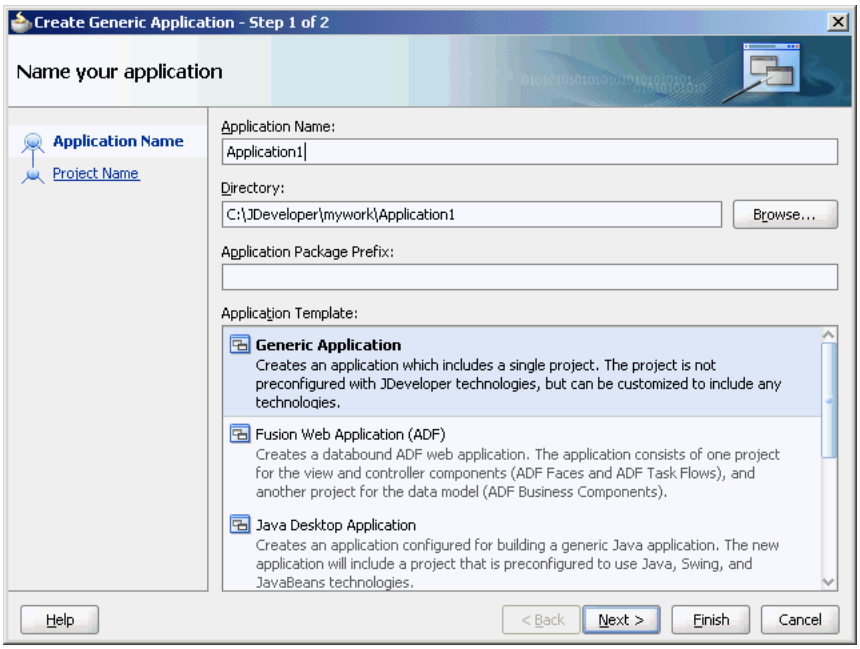
To obtain the `adapters-aq-103-adtclobpayload` sample, access the Oracle Sample SOA Code site.

7.4.1.1.2 Creating an Application and an SOA Project

You must create a JDeveloper application to contain the SOA composite. Perform the following steps to create an application, a SOA project:

1. Open JDeveloper.
2. In the **Application Navigator**, click **New Application**. The Create Generic Application Name your application page is displayed, as shown in Figure 7–1.
3. Enter a name for the application in the **Application Name** field.
4. In the **Application Template** list, choose **Generic Application**.

Figure 7–1 The Create Generic Application Name your application Page



5. Click **Next**.

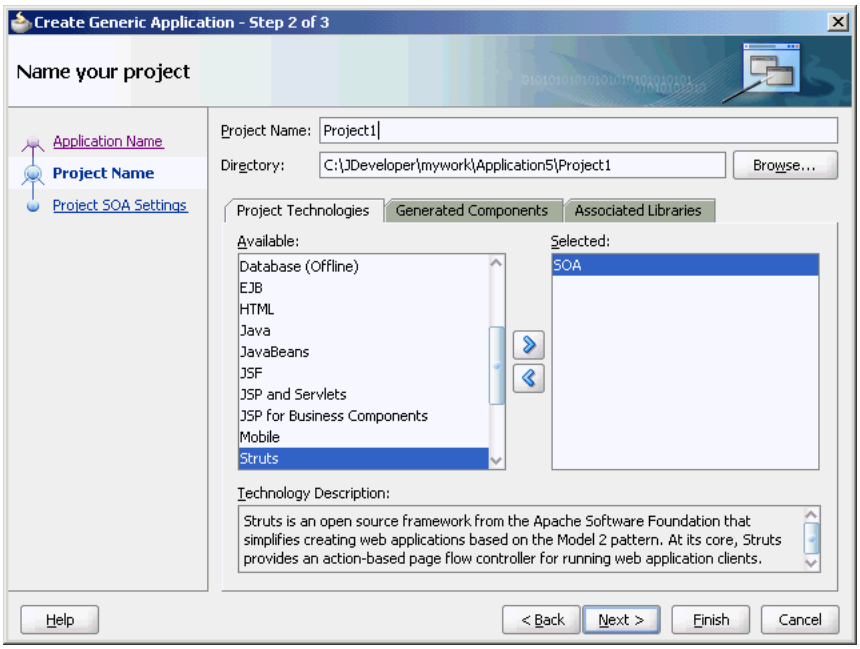
   The Create Generic Application Name your project page is displayed, as shown in *** 'The Create Generic Application Name your Generic project Page' on page 15 ***.

6. In the **Project Name** field, enter a descriptive name.

   For example, `SOAComposite`.

7. In the **Available list** in the **Project Technologies** tab, double-click **SOA** to move it to the Selected list.

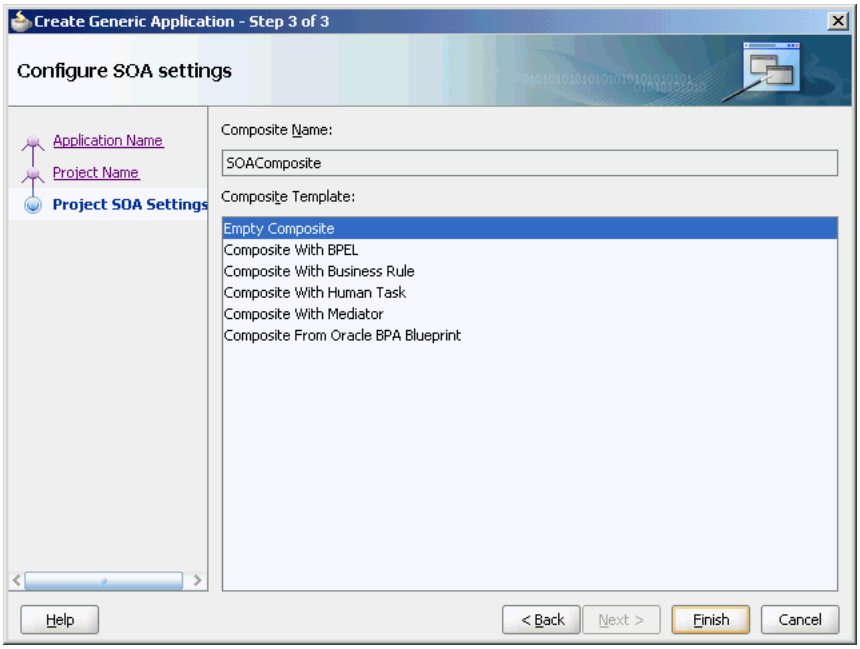
Figure 7–2 The Create Generic Application Name your Generic project Page



8. Click **Next**.

   The Create Generic Application Configure SOA settings page is displayed, as shown in Figure 7–3.

Figure 7–3 The Create Generic Application Configure SOA settings Page

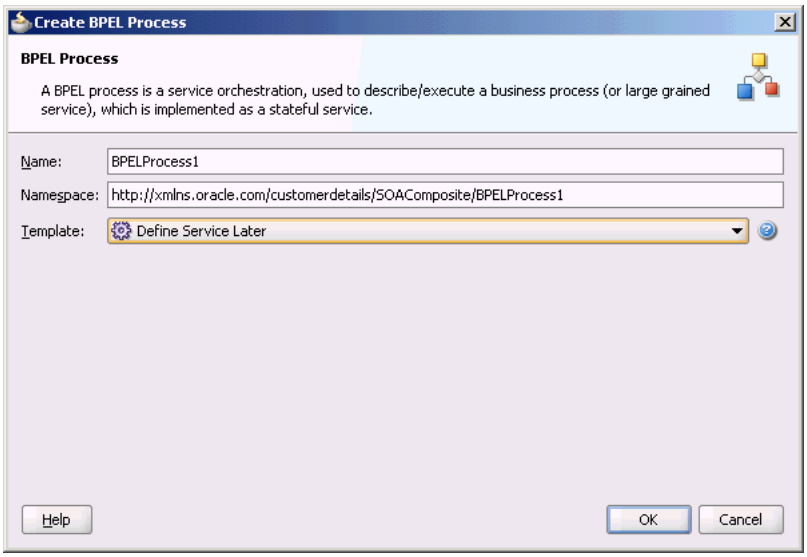


9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

   You have created a new application and an SOA project. This automatically creates an SOA composite.

The Create BPEL Process page is displayed, as shown in Figure 7–4.

Figure 7–4 The Create BPEL Process Page



10. Enter a name for the BPEL process in the **Name** field. For example, `CustomerDetails`.
11. Select **Define Service Later** in the Template list, and then click **OK**.

    You have created the `CustomerDetails` BPEL process.

7.4.1.1.3 Defining an Oracle AQ Adapter Service

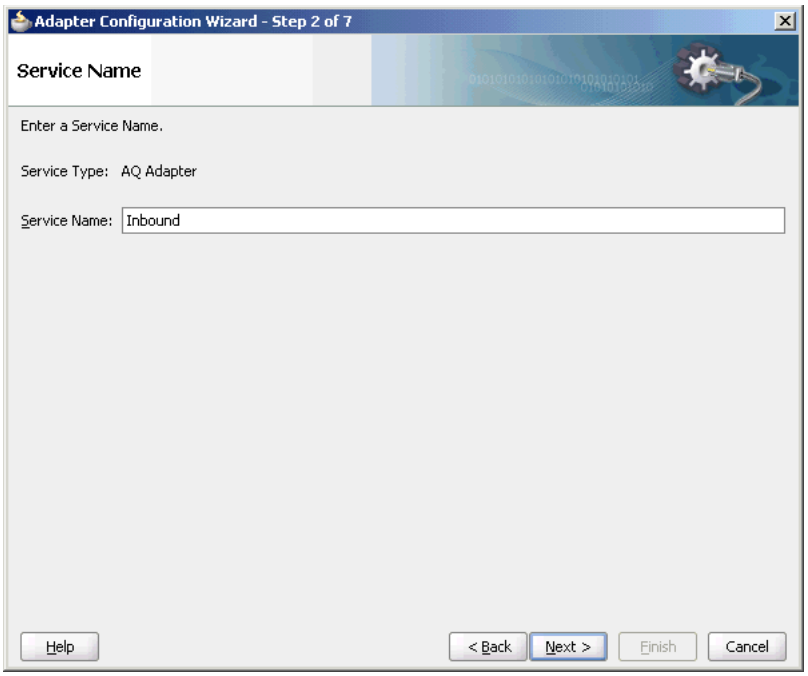
The next step is to define an Oracle AQ Adapter service. Perform the following steps to create an Oracle AQ Adapter service:

1. In the Component Palette, select **SOA**.
2. Drag and drop **AQ Adapter** from the Service Adapters list to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard Welcome page is displayed.
3. Click **Next**.

   The Adapter Configuration Wizard Service Name page is displayed, as shown in Figure 7–5.

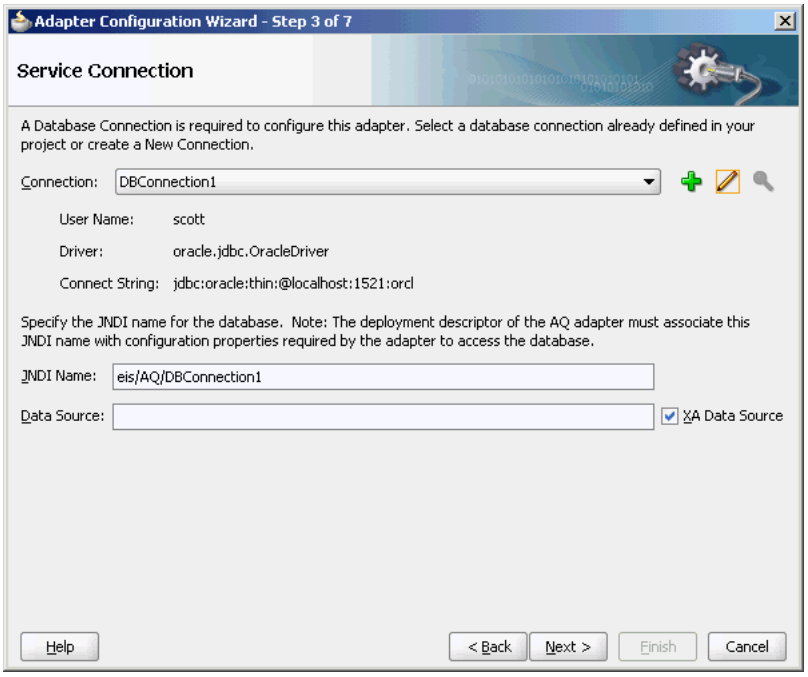
Figure 7–5 The Adapter Configuration Wizard Service Name Page



4. Specify a service name, and then click **Next**.

   The Adapter Configuration Wizard Service Connection page is displayed, as shown in Figure 7–6.

Figure 7–6 Adapter Configuration Wizard Service Connection Page



5. Click the plus icon to create a database connection.

   The Create Database Connection page is displayed.

Note: You must connect to the database where Oracle Applications is running.

6. Enter the following information:
   a. For **Create Connection In**, choose **Application Resources**.
   b. In the **Connection Name** field, specify a unique name for the database connection.

      In this example, type **DBConnection1**.
   c. From the **Connection Type** box, select **Oracle (JDBC)**.
   d. In the **UserName** field, specify the user name to be authorized for access to the database.

      In this example, type `scott`.
   e. In the **Role** field, enter a role, if applicable.

      This must be a specific database role, such as `SYSDBA`, as defined in the database. This field is optional. In this example, leave the **Role** field blank.
   f. In the **Password** field, specify the password to be associated with the specified user name.

      In this example, type `tiger`.
   g. Select **Save Password** and **Deploy Password**.
   h. From the Driver list, select **Thin**.
   i. In the **Host Name** field, enter a value to identify the computer running the Oracle server.

      Use an IP address or a host name that can be resolved by TCP/IP, for example, `myserver`. The default value is `localhost`.
   j. In the **JDBC Port** field, enter a value to identify the TCP/IP port. The default is `1521`.
   k. In the **SID** field, enter a value for the unique system identifier (SID) of an Oracle database instance.

      The default is `XE`.
   l. Click **Test Connection** to determine whether the specified information establishes a connection with the database.

      A Success message is displayed.
   m. Click **OK**.

      The Connection you created is displayed in the Connection field in the Service Name page.

      Notice that the Java Naming and Directory Interface (JNDI) name in the JNDI Name field is populated after you have created the database connection. The JNDI name acts as a placeholder for the connection used when your service is deployed to the BPEL server. Using JNDI as a placeholder enables you to use different databases for development and later production.

      The value specified in the JNDI name must exist in the Oracle AQ Adapter `weblogic-ra.xml` file to ensure that the adapter runs in managed mode. A

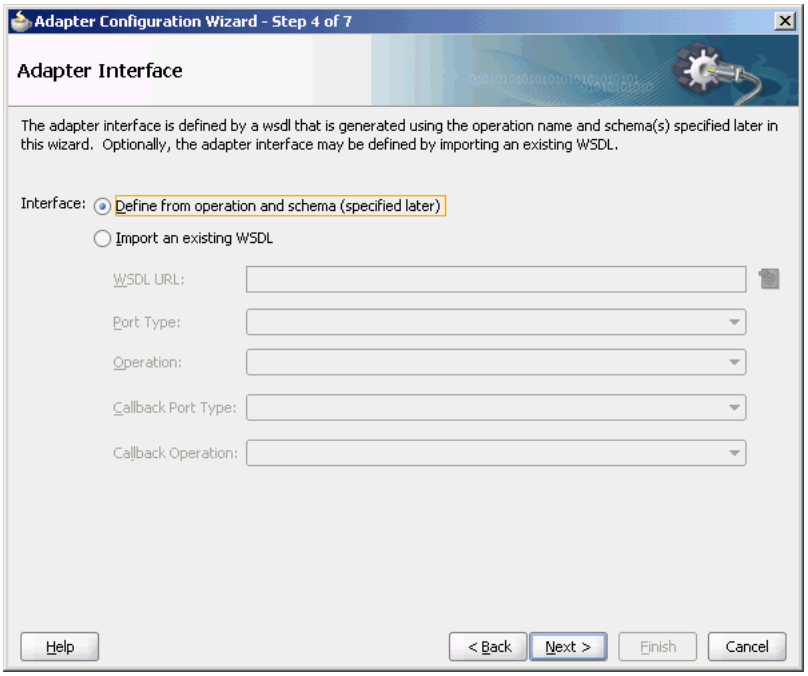
default connection instance `eis/AQ/aqSample` is shipped and can be used as the default value for this field. To use this connection instance, it would still require that a data source is created with the JNDI name `jdbc/aqSample`.

7. Click **Next**.

   The Adapter Configuration Wizard Adapter Interface page is displayed, as shown in Figure 7–7.

8. In the Adapter Interface page, choose **Define from operation and schema (specified later)**.

Figure 7–7 The Adapter Configuration Wizard Adapter Interface Page



9. Click **Next**.

   The Operation page is displayed.

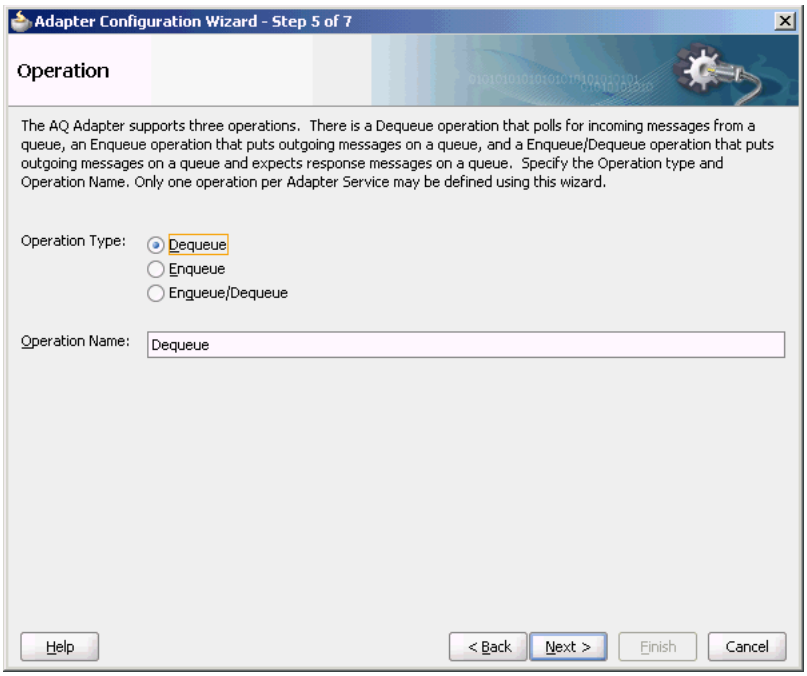
10. Oracle AQ Adapter supports three operations:

    - **Dequeue**: Polls for incoming messages from a queue.
    - **Enqueue**: Puts outgoing messages in a queue.
    - **Enqueue/Dequeue**: Puts outgoing messages in a queue and expects response messages in a queue.

    In this example, select **Dequeue**, as shown in Figure 7–8.

    The operation is automatically named after the operation that you selected. However, you can edit the **Operation Name** field.

Figure 7–8 The Adapter Configuration Wizard Operation Page

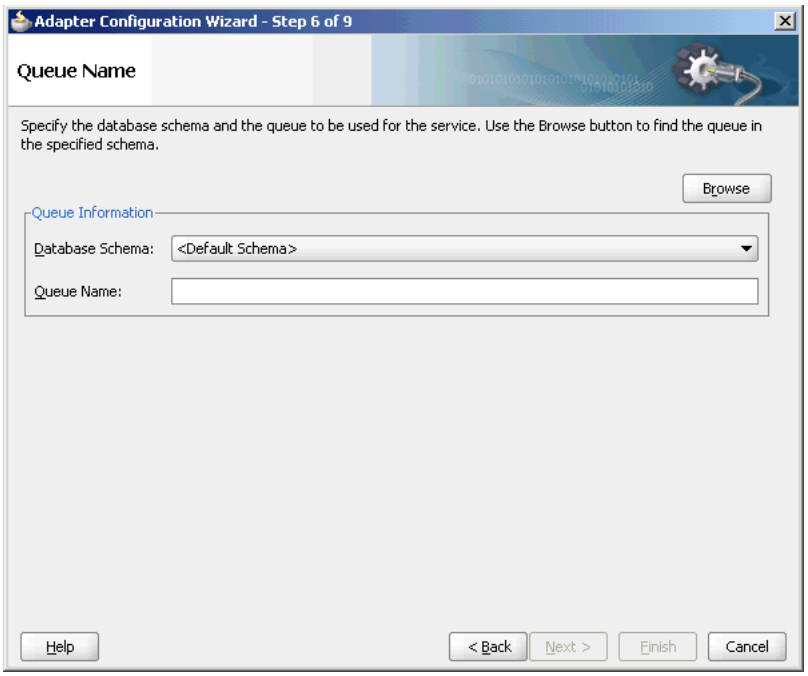


Note: When creating an SOA composite that uses Oracle AQ Adapter with ADT data type if the `SchemaValidation` property is set to `TRUE`, then any `NULL` data type in dequeue message results in `AQ_INVALID_PAYLOAD` error further resulting in message rejection. To avoid message rejection, you must set the `SchemaValidation` property to `false`.

11. Click **Next**.

    The Adapter Configuration Wizard Queue Name page is displayed, as shown in Figure 7–9.

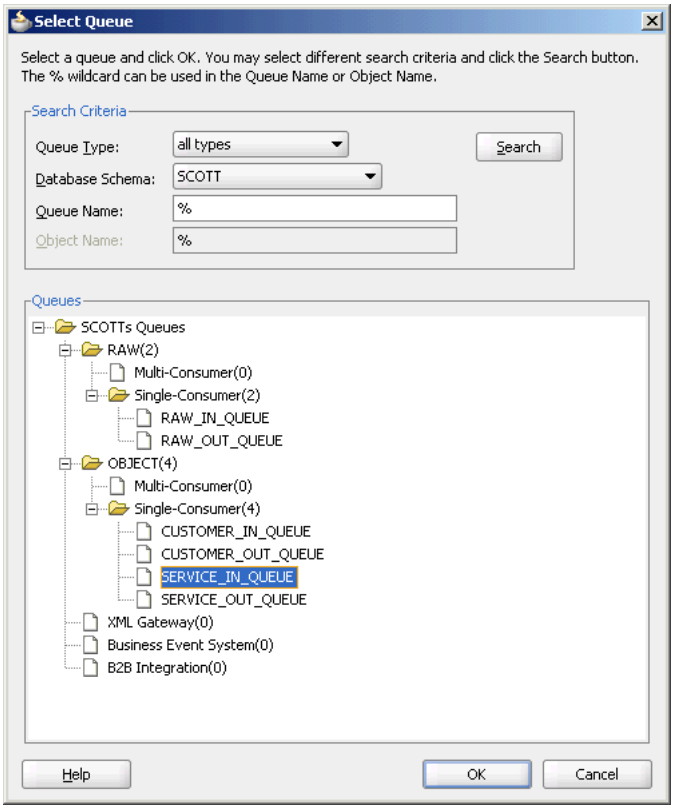
Figure 7–9 The Adapter Configuration Wizard Queue Name Page



12. Select a database schema from the Database Schema list, or click **Browse** to browse for the schema. In this example, click **Browse**.

The Select Queue dialog is displayed, as shown in Figure 7–10.

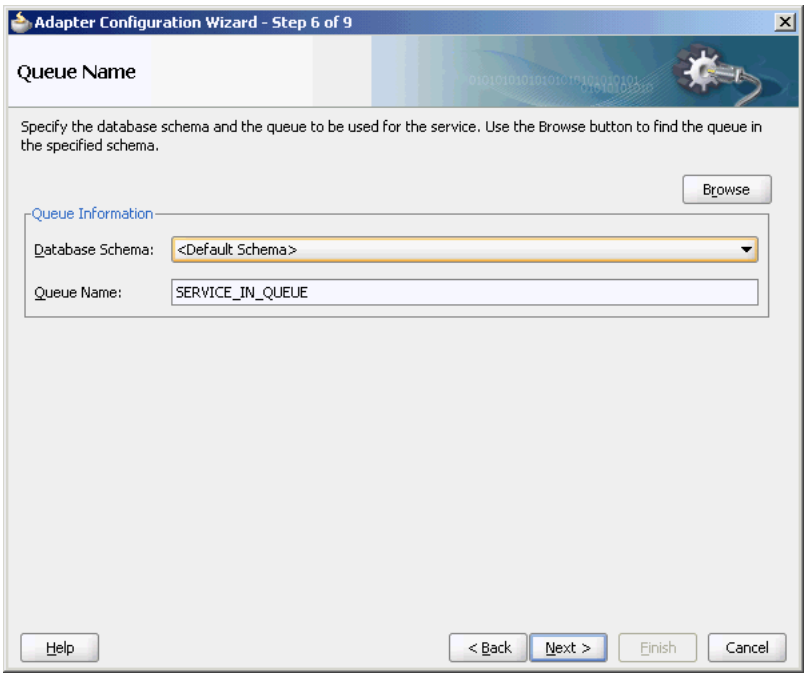
Figure 7–10 The Select Queue Dialog



13. Select the required queue, and then click **OK**.

    In this example, select SERVICE_IN_QUEUE. The Queue Name page is displayed again with the Queue Name field populated with SERVICE_IN_QUEUE, as shown in Figure 7–11.

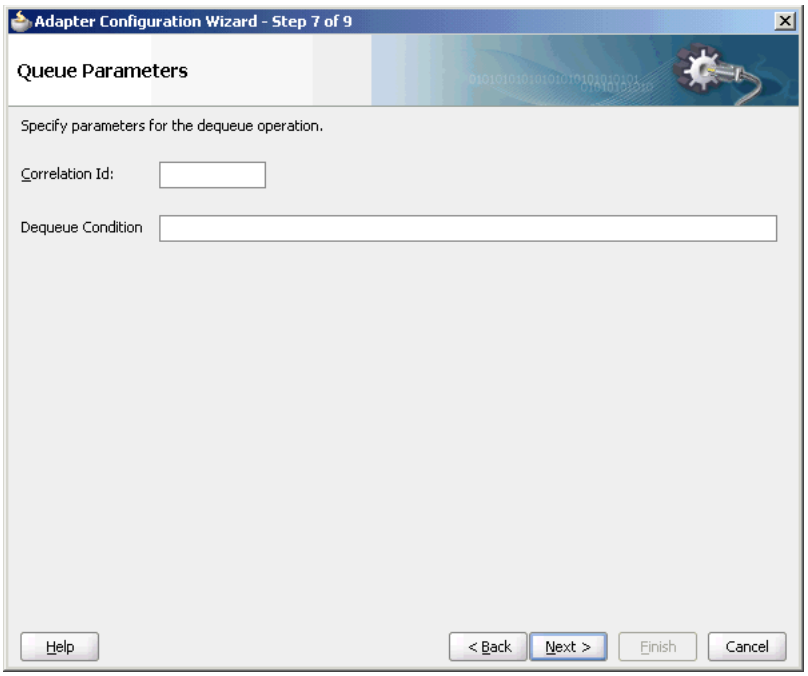
Figure 7–11 The Adapter Configuration Wizard Queue Name Page



14. Click **Next**.

    The Adapter Configuration Wizard Queue Parameters page is displayed, as shown in Figure 7–12.

Figure 7–12 The Adapter Configuration Wizard Queue Parameters Page



15. Enter values for the parameters, and then click **Next**.

    - **Correlation ID**: Enter an optional correlation ID from 1 to 30 characters in length. This is used to identify messages that can be retrieved at a later time by a dequeue activity using the same correlation ID.

      The value to enter is agreed upon between the enqueuing sender and the dequeuing receiver for asynchronous conversations. The correlation ID maps to an AQ header property. Correlation IDs in the inbound direction enable you to be selective about the message to dequeue. This field is optional. If you do not enter a value, then all the messages in the queue are processed.

      If you enter a value for the Correlation ID in the outbound direction, then all outbound messages have the correct ID set to the value entered. You can override this value on a per message basis in the correlation field of the outbound header.

    - **Dequeue Condition**: Displayed only when you select dequeue in the Operation page.

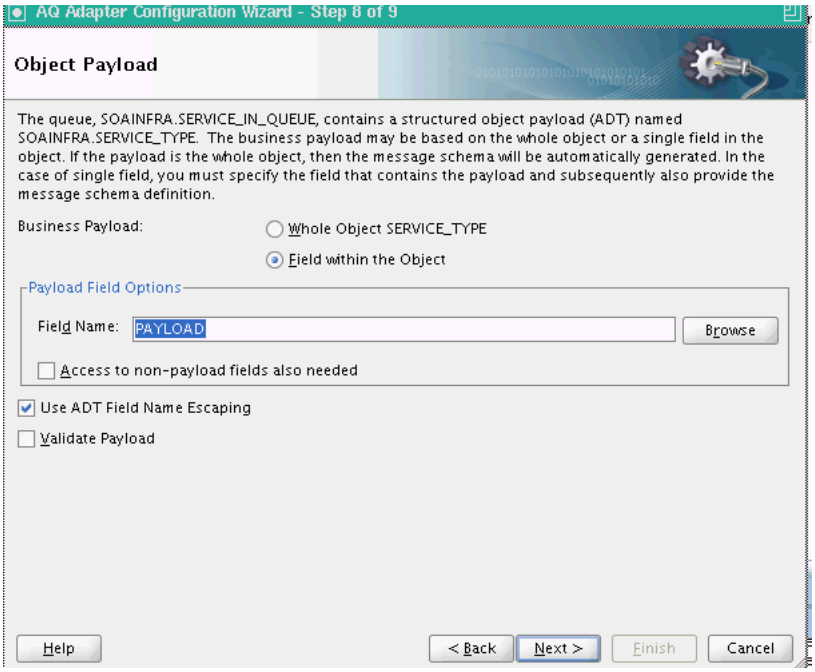
      Enter a Boolean expression similar to the `WHERE` clause of a `SQL` query. This expression can include conditions on message properties, user data properties (`object payloads` only), and PL/SQL or SQL functions. If more than one message satisfies the dequeue condition, then the order of dequeuing is indeterminate, and the sort order of the queue is not honored.

      This field is displayed for inbound single consumer and multiconsumer queues.

16. Click **Next**.

    The Adapter Configuration Wizard Object Payload page is displayed, as shown in Figure 7–13.

Figure 7–13 The Adapter Configuration Wizard Object Payload Page

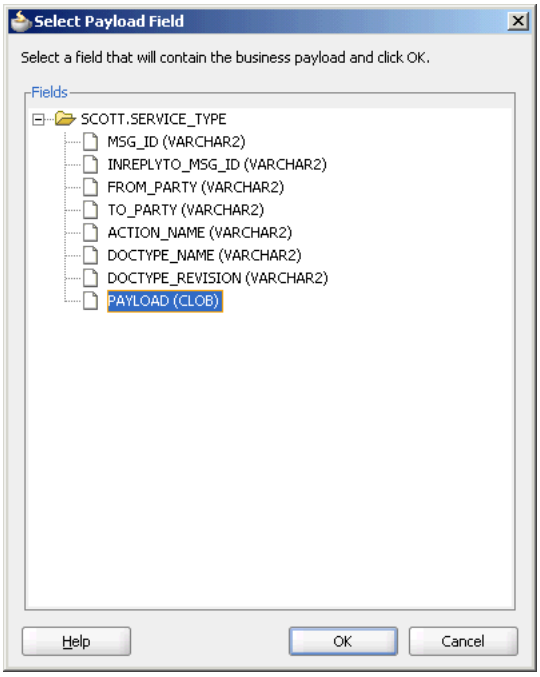


a. In Business Payload, select **Field within the Object**.

b. Click **Browse** in the Payload Fields Options section to select a field that contains the business payload.

The Select Payload Field dialog is displayed, as shown in Figure 7–14.

Figure 7–14 The Select Payload Field Dialog

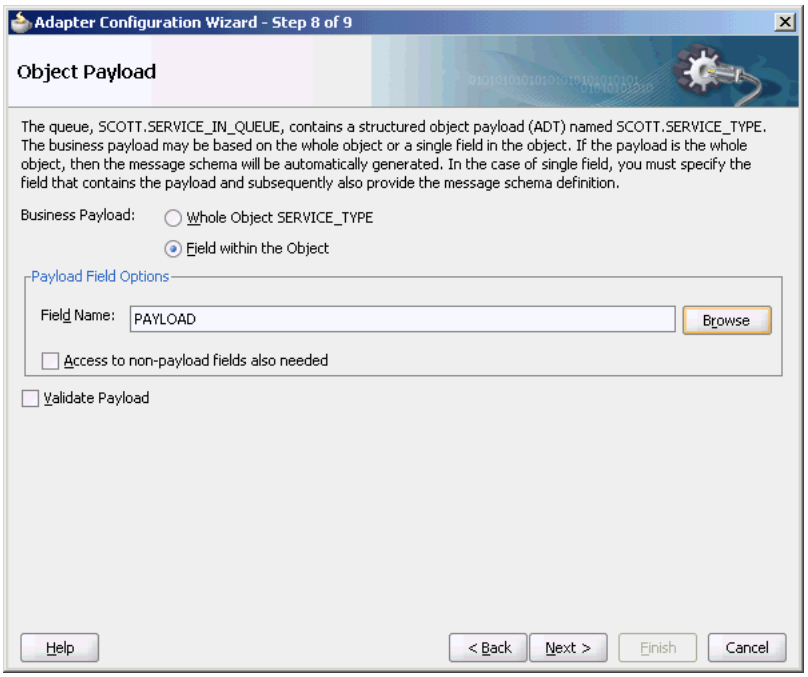


17. Select a field, and then click **OK**.

In this example, select **PAYLOAD (CLOB).**

The Object Payload field is displayed with all the payload details filled up, as shown in Figure 7–15.

Figure 7–15 The Adapter Configuration Wizard Object payload Page



18. Select **Access to non-payload fields also needed**, and then click **Next**.

    The Messages page is displayed.

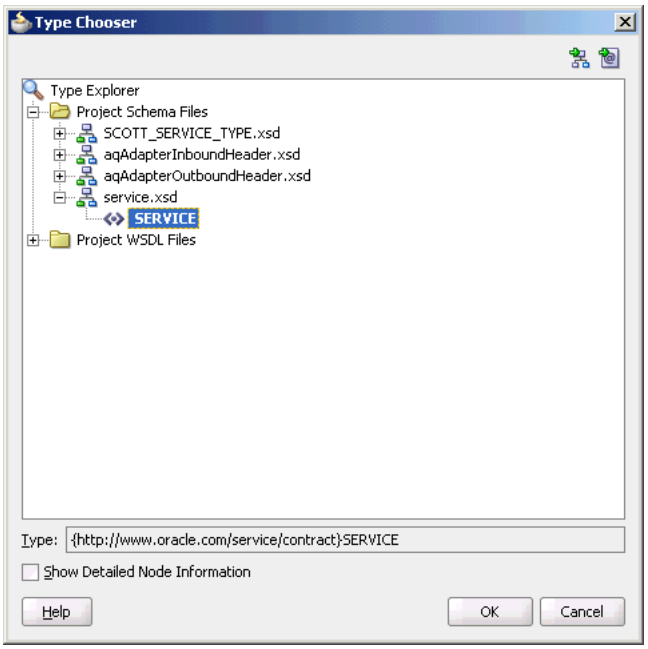
    The Message page has the following options:

    - **Native format translation is not required (Schema is Opaque)**: Select this option if you do not want to specify a schema. Selecting this option disables all the other fields under Message Schema.
    - **Define Schema for Native Format**: Click this to start the Native Format Builder wizard, which guides you through the process of defining the native format.
    - **URL**: You can enter the path for the schema file URL or click **Browse** to browse for the path.
    - **Schema Element**: The name of the schema element.

19. In this example, click the **Browse for schema file** icon to browse for the schema file URL.

    The Type Chooser dialog is displayed, as shown in Figure 7–16.

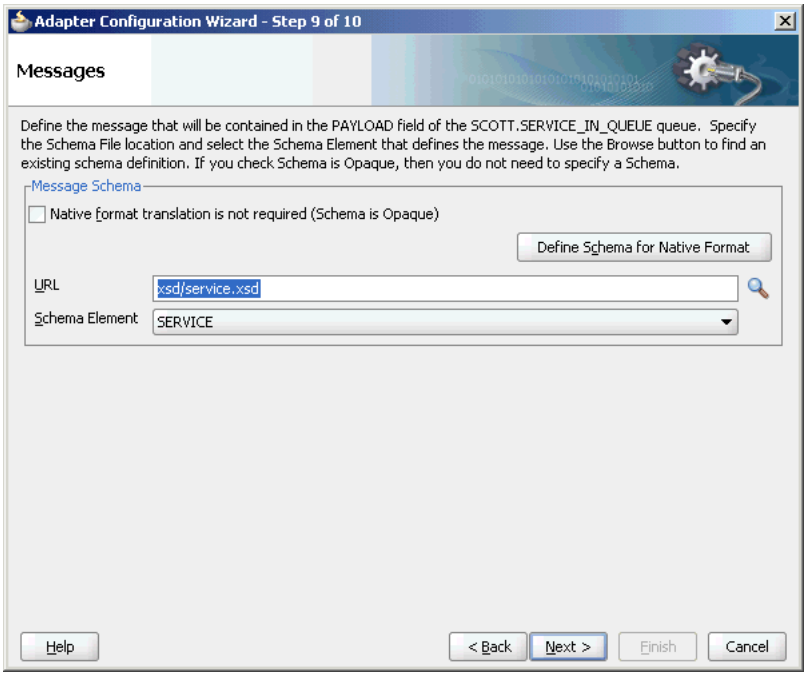
Figure 7–16 The Type Chooser Dialog



20. Select **SERVICE** from the list, as shown in Figure 7–16, and then click **OK**.

The Messages page reappears, with the Schema Location and Schema Element fields populated, as shown in Figure 7–17.

Figure 7–17 The Adapter Configuration Wizard Messages Page



21. Click **Next**.

The Finish screen is displayed. This page shows the path and name of the adapter file that the wizard creates.

22. Click **Finish**.

    You have created an AQ Adapter service with dequeue operation.

23. Click **OK**.

7.4.1.1.4 Generated WSDL and JCA Files

The adapter service generates a WSDL and a JCA file to serve as the defined adapter interface.

The following is the WSDL file generated for the dequeue operation:

```
<definitions name="Inbound"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT"
xmlns:imp1="http://www.oracle.com/service/contract">
    <types>
    <schema attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"
xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT">
       <import namespace="http://xmlns.oracle.com/xdb/SCOTT"
schemaLocation="xsd/SCOTT_SERVICE_TYPE.xsd"/>
        <import namespace="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"
schemaLocation="xsd/aqAdapterInboundHeader.xsd"/>
            <complexType name="HeaderCType">
                <sequence>
                    <element name="PayloadHeader" type="obj1:SERVICE_TYPE"/>
                </sequence>
            </complexType>
            <element name="Header" type="tns:HeaderCType"/>
        </schema>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://www.oracle.com/service/contract"
schemaLocation="xsd/service.xsd"/>
        </schema>
    </types>
    <message name="SERVICE_msg">
        <part name="SERVICE" element="imp1:SERVICE"/>
    </message>
    <message name="Header_msg">
        <part name="Header" element="tns:Header"/>
    </message>
    <portType name="Dequeue_ptt">
        <operation name="Dequeue">
            <input message="tns:SERVICE_msg"/>
        </operation>
    </portType>
    <plt:partnerLinkType name="Dequeue_plt">
        <plt:role name="Dequeue_role">
            <plt:portType name="tns:Dequeue_ptt"/>
        </plt:role>
    </plt:partnerLinkType>
</definitions>
```

Note that the element name QueueHeader, formerly required, is no longer required. Set the QueueHeader properties using Normalized properties `jca.aq.xxxx`. You can set these properties from the BPEL dialog for the Receive and Invoke activities.

7.4.1.2 Dequeuing and Enqueuing Object and ADT Payloads

Dequeuing and enqueuing is covered in Section 7.4.2, "Oracle AQ Adapter ADT Queue".

To enqueue or dequeue the entire object as the payload, perform the following:

- Select **Enqueue** or **Dequeue** in Step 10.
- Select **Whole Object CUSTOMER_TYPE**, and skip to Step 16.

For a working example of an ADT payload use case, refer to any of the following samples:

- `adapters-aq-102-adt`
- `adapters-aq-110-supportedadttypes`

You can obtain these samples by accessing the Oracle SOA Sample Code site.

Note: If you create an ADT type queue and drop both the queue and the data types created for that queue and redeploy the process, then it throws a SQL exception and you must restart the Database. Instead, drop only the queues and not the data types.

7.4.1.3 Dequeuing One Column of the Object Payload

The walkthrough is an example of dequeuing one field or column within an object payload.

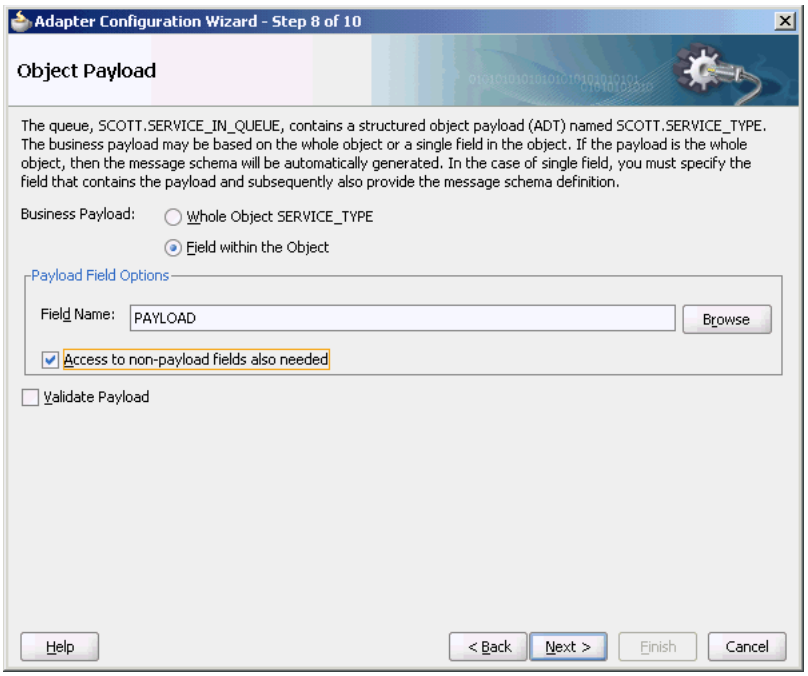
To create an Oracle AQ Adapter that dequeues one field in an object, you must perform the following steps in the Adapter Configuration Wizard Object Payload page:

1. Select **Field within the Object**.
2. Click **Browse** at the end of the **Field Name** field.

   The Select Payload Field dialog is displayed.
3. Select a field that contains the business payload, and then click **OK**.

   The Adapter Configuration Wizard Object Payload page with Field Name field populated with the field that you selected is displayed, as shown in Figure 7–18.

Figure 7–18 The Adapter Configuration Wizard Object Payload Page



4. Select **Access to non-payload fields also needed**, and then click **Next**.

The following segment of the generated JCA file specifies that one field, in this case the field named `PAYLOAD`, is dequeued in addition to payload header fields.

```
<adapter-config name="Inbound" adapter="AQ Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
    <connection-factory location="eis/AQ/aqSample" UIConnectionName="Connection1"
adapterRef=""/>
  <endpoint-activation portType="Dequeue_ptt" operation="Dequeue">
    <activation-spec
className="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec">
      <property name="QueueName" value="SERVICE_IN_QUEUE"/>
      <property name="ObjectFieldName" value="PAYLOAD"/>
      <property name="PayloadHeaderRequired" value="true"/>
      <property name="SchemaValidation" value="false"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

For a working example of an ADT `CLOB` use case where one field or column within an object payload is dequeued, refer to the following samples:

- `adapters-aq-103-adtclobpayload`
- `adapters-aq-105-adtclobopaquepayload`

You can obtain these samples by accessing the Oracle SOA Sample Code site.

7.4.1.4 Configuring the Enqueue/Dequeue Operation Type

This use case walks you through the procedure for configuring the Enqueue/Dequeue operation type of the Oracle AQ Adapter, which lets the Oracle AQ Adapter put outgoing messages on a queue and expect response messages on a different queue.

This section includes the following topics:

- Section 7.4.1.4.1, "Meeting Prerequisites"
- Section 7.4.1.4.2, "Creating an Application and an SOA Project"
- Section 7.4.1.4.3, "Defining an Oracle AQ Adapter Service"
- Section 7.4.1.4.4, "Wiring Services and Activities"
- Section 7.4.1.4.5, "Deploying with JDeveloper"
- Section 7.4.1.4.6, "Monitoring Using the Fusion Middleware Control Console"
- Section 7.4.1.4.7, "Generated WSDL and JCA Files"

7.4.1.4.1 Meeting Prerequisites

To perform this use case, you must have access to a database with the SCOTT schema. Also, you require the following files from the `artifacts.zip` file contained in the `adapters-aq-104-requestreply` sample:

- `create_queues.sql`
- `drop_queues.sql`
- `enqueue.sql`
- `SendReply.sql`
- `setup_user.sql`

To obtain the adapters-aq-104-requestreply sample code, access the Oracle SOA Sample Code site.

7.4.1.4.2 Creating an Application and an SOA Project

You must create a JDeveloper application to contain the SOA composite. Follow the steps documented in Section 7.4.1.1.2, "Creating an Application and an SOA Project" to create an application, and an SOA project.

7.4.1.4.3 Defining an Oracle AQ Adapter Service

Perform the following steps to create an Oracle AQ Adapter service to put outgoing messages on a queue and expect response messages on a queue:

1. In the Component Palette, select **SOA**.
2. Drag and drop **AQ Adapter** from the Service Adapters list to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard Welcome page is displayed.
3. Click **Next**.

   The Adapter Configuration Wizard Service Name page is displayed, as shown in Figure 7–5.
4. Specify a service name, and then click **Next**.

   The Adapter Configuration Wizard Service Connection page is displayed.
5. Click the plus icon to create a database connection.

   The Create Database Connection page is displayed.

> **Note:** You must connect to the database where Oracle Applications is running.

6. Enter the following information:
   a. For **Create Connection In**, choose **Application Resources**.
   b. In the **Connection Name** field, specify a unique name for the database connection.
   c. From the **Connection Type** box, select **Oracle (JDBC)**.
   d. In the **UserName** field, specify the user name to be authorized for access to the database.

      In this example, type `scott`.
   e. In the **Role** field, enter a role, if applicable.

      This must be a specific database role, such as `SYSDBA`, as defined in the database. This field is optional. In this example, leave the **Role** field blank.
   f. In the **Password** field, specify the password to be associated with the specified user name.

      In this example, type `tiger`.
   g. Select **Save Password** and **Deploy Password**.
   h. From the Driver list, select **thin**.
   i. In the **Host Name** field, enter a value to identify the computer running the Oracle server.

      Use an IP address or a host name that can be resolved by TCP/IP, for example, `myserver`. The default value is `localhost`.
   j. In the **JDBC Port** field, enter a value to identify the TCP/IP port. The default is `1521`.
   k. In the **SID** field, enter a value for the unique system identifier (SID) of an Oracle database instance.

      The default is `XE`.
   l. Click **Test Connection** to determine whether the specified information establishes a connection with the database.

      A Success message is displayed.
   m. Click **OK**.

      The Connection you created is displayed in the Connection field in the Service Connection page.

      Also, the JNDI Name field is populated after you created the database connection.

      The value specified in the JNDI name must exist in the Oracle AQ Adapter `weblogic-ra.xml` file to ensure that the adapter runs in managed mode. A default connection instance `eis/AQ/aqSample` is shipped and can be used as the default value for this field. To use this connection instance, it would still require that a data source is created with the JNDI name `jdbc/aqSample`.
7. Click **Next**.

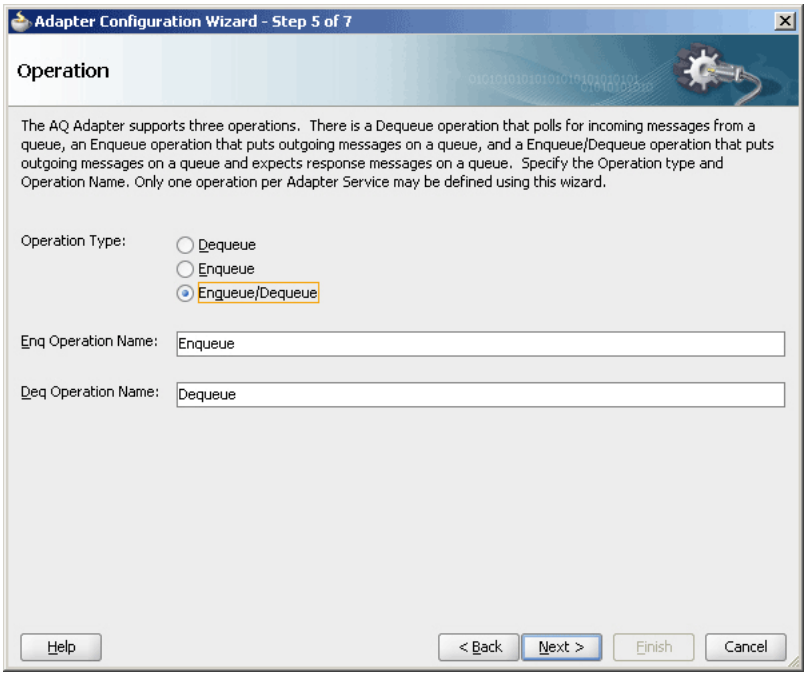
   The Adapter Configuration Wizard Adapter Interface page is displayed.
8. In the Adapter Interface page, choose **Define from operation and schema (specified later)**.

9. Click **Next**.

   The Operation page is displayed.

10. Select **Enqueue/Dequeue**, as shown in Figure 7–8.

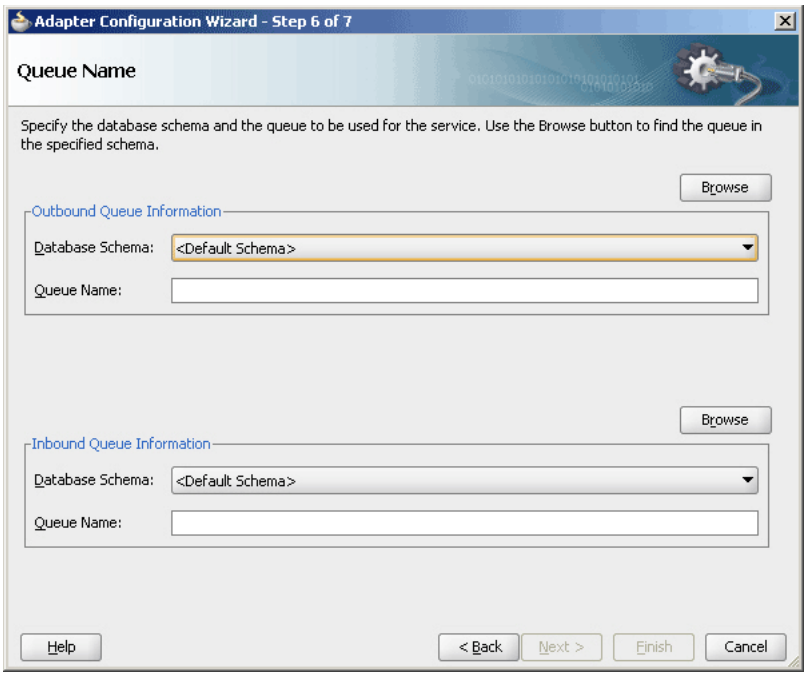
Figure 7–19 The Adapter Configuration Wizard Operation Page



11. Click **Next**.

    The Adapter Configuration Wizard Queue Name page is displayed, as shown in Figure 7–9.

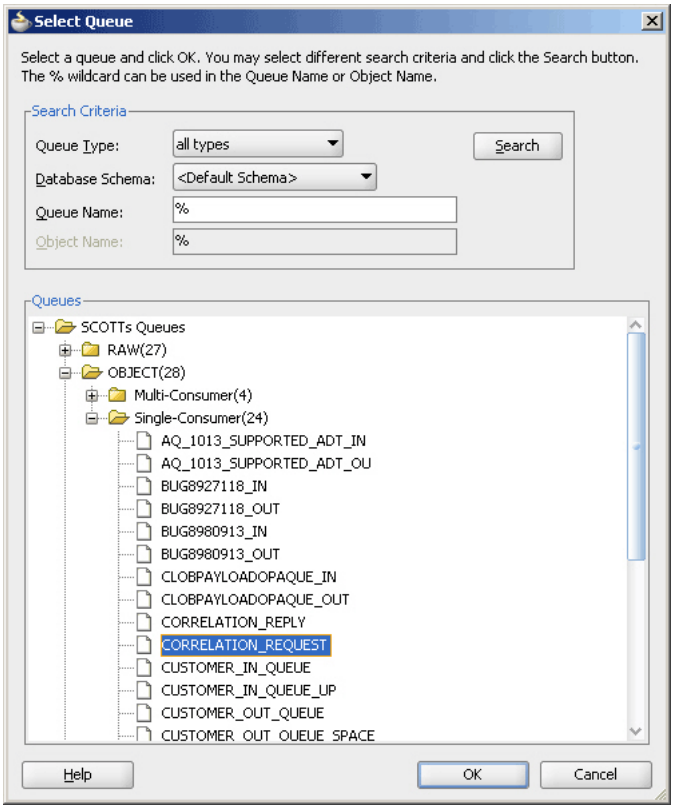
Figure 7–20 The Adapter Configuration Wizard Queue Name Page



12. Click **Browse** to browse for a request queue.

The Select Queue dialog is displayed, as shown in Figure 7–21.

Figure 7–21 The Select Queue Dialog



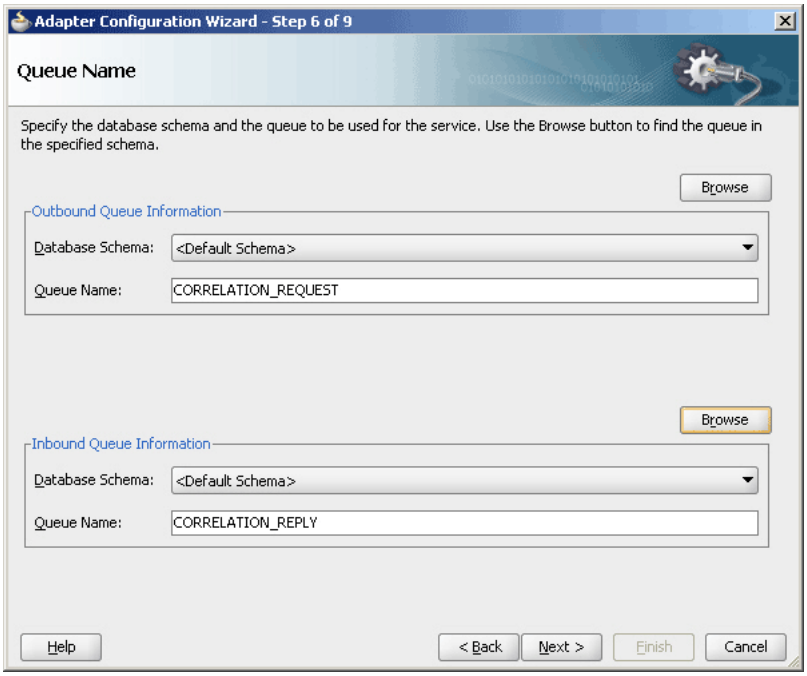
13. Select the required queue, and then click **OK**.

    In this example, select **CORRELATION_REQUEST**. The Queue Name page is displayed with the Queue Name field populated with CORRELATION_REQUEST, as shown in Figure 7–22.

14. Repeat Step 12 and 13 for the enqueue queue information.

    The Queue Name page is displayed, as shown in Figure 7–22.

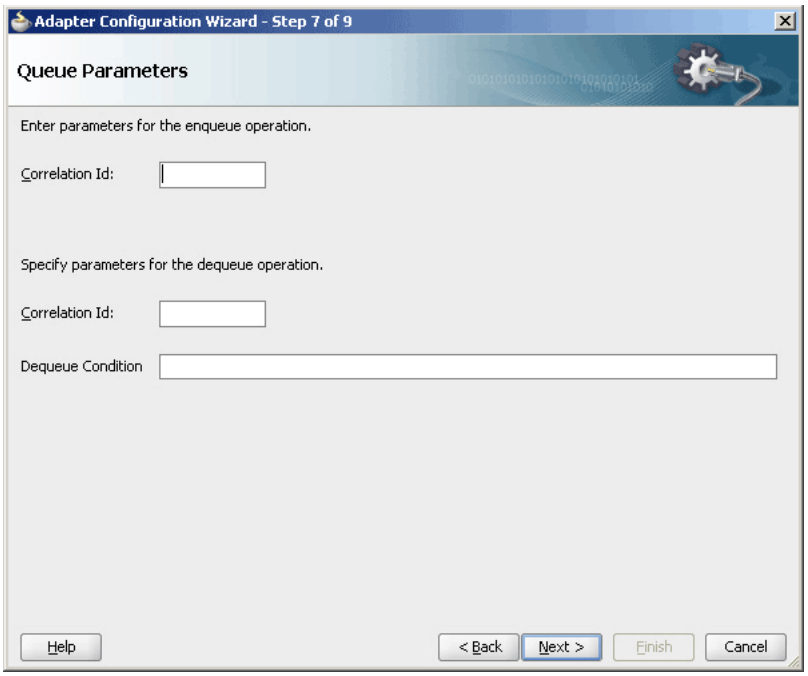
Figure 7–22 The Adapter Configuration Wizard Queue Name Page



15. Click **Next**.

    The Adapter Configuration Wizard Queue Parameters page is displayed, as shown in Figure 7–23.

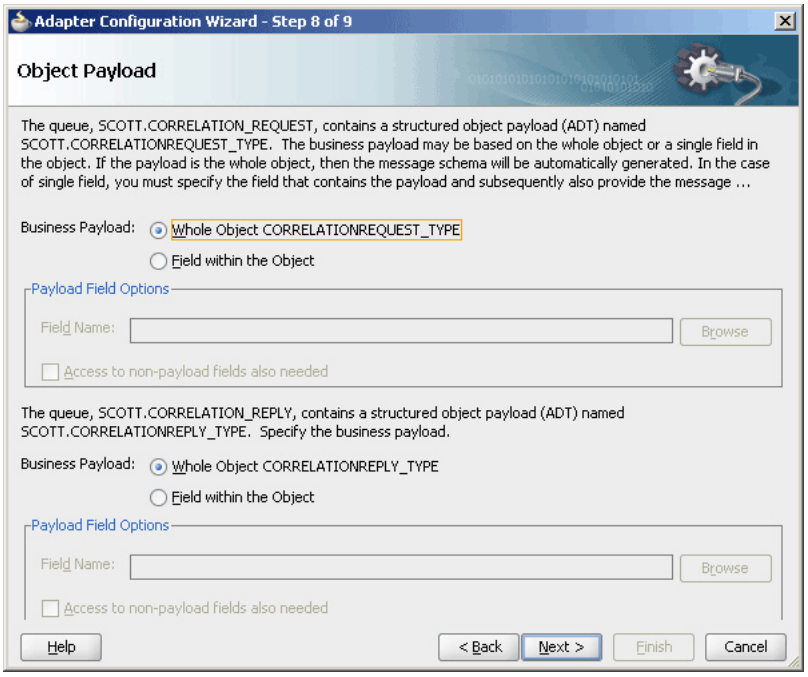
Figure 7–23 The Adapter Configuration Wizard Queue Parameters Page



16. Click **Next**.

    The Adapter Configuration Wizard Object Payload page is displayed, as shown in Figure 7–24.

Figure 7–24 The Adapter Configuration Wizard Object Payload Page



17. Select the Business Payload options, **Whole Object CORRELATIONREQUEST_TYPE** and **Whole Object CORRELATIONREPLY_TYPE**.

18. Click **Next**.

The Finish screen is displayed. This page shows the path and name of the adapter file that the wizard creates.

19. Click **Finish**.

    You have created an AQ Adapter service for synchronous enqueue/dequeue operations.

20. Click **OK**.

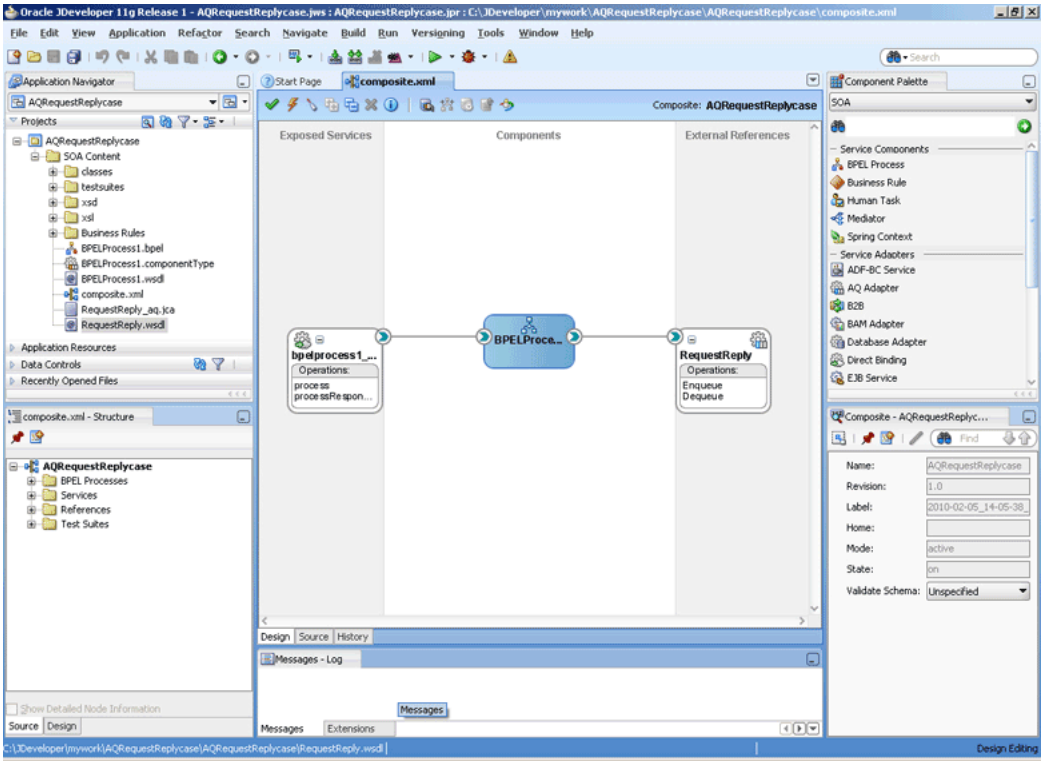
7.4.1.4.4 Wiring Services and Activities

You must assemble or wire the BPEL process and the Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in RequestReply in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 7–25.

Figure 7–25 The JDeveloper - Composite.xml



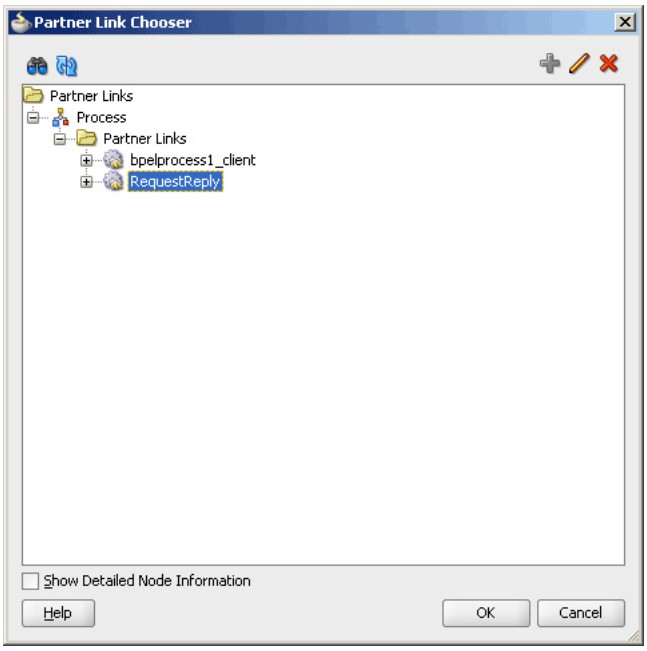
2. Click **File**, **Save All**.

Add Invoke Activity

1. Double-click **BPELProcess1**. The BPELProcess1.bpel page is displayed.
2. Drag and drop an **Invoke** activity from the Component Palette to the design area.
3. Double-click the **Invoke** activity. The Invoke dialog is displayed.
4. Enter a name for the invoke activity in the **Name** field.

5. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed, as shown in Figure 7–26.

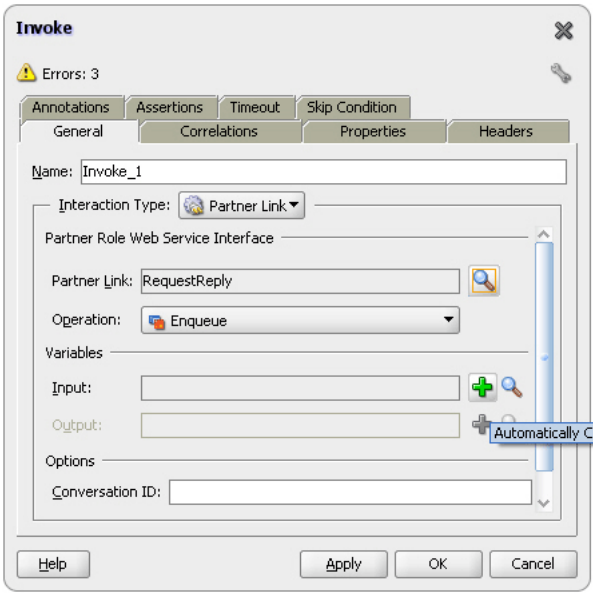
Figure 7–26 The Partner Link Chooser Dialog



6. Select **RequestReply**, and click **OK**.

7. Click the **Automatically Create Input Variable** icon to the right of the Input variable field in the Invoke dialog, as shown in Figure 7–27. The Create Variable dialog is displayed.

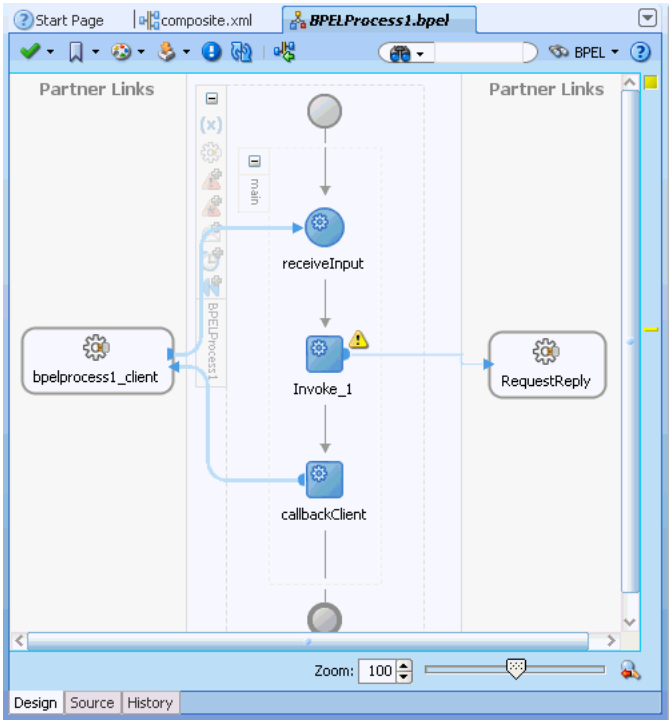
Figure 7–27 The Invoke Dialog



8. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.

9. Click **OK**. The JDeveloper BPELProcess1.bpel page appears, as shown in *** 'The JDeveloper - BPELProcess1.bpel Page' on page 38 ***.

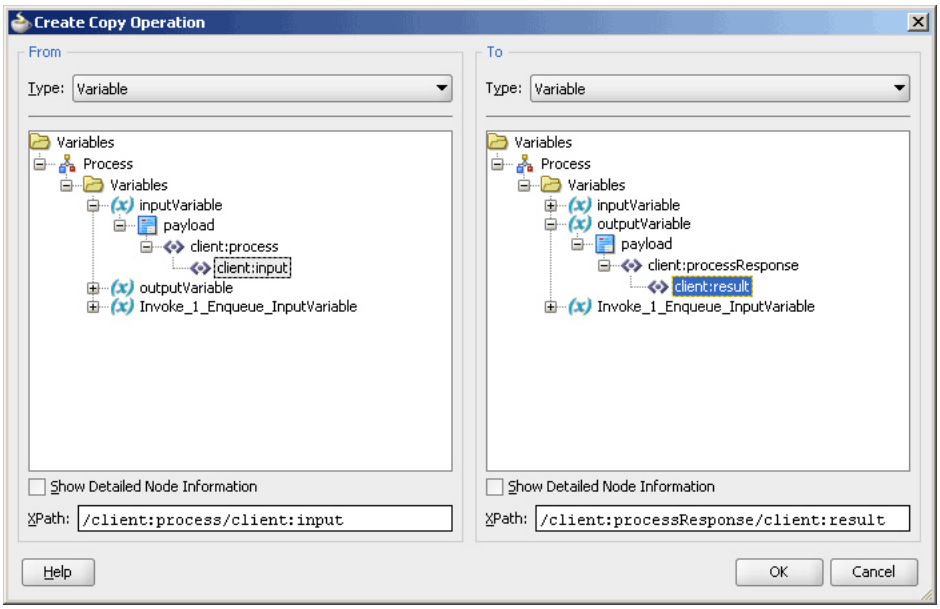
Figure 7–28 The JDeveloper - BPELProcess1.bpel Page



Add an Assign Activity

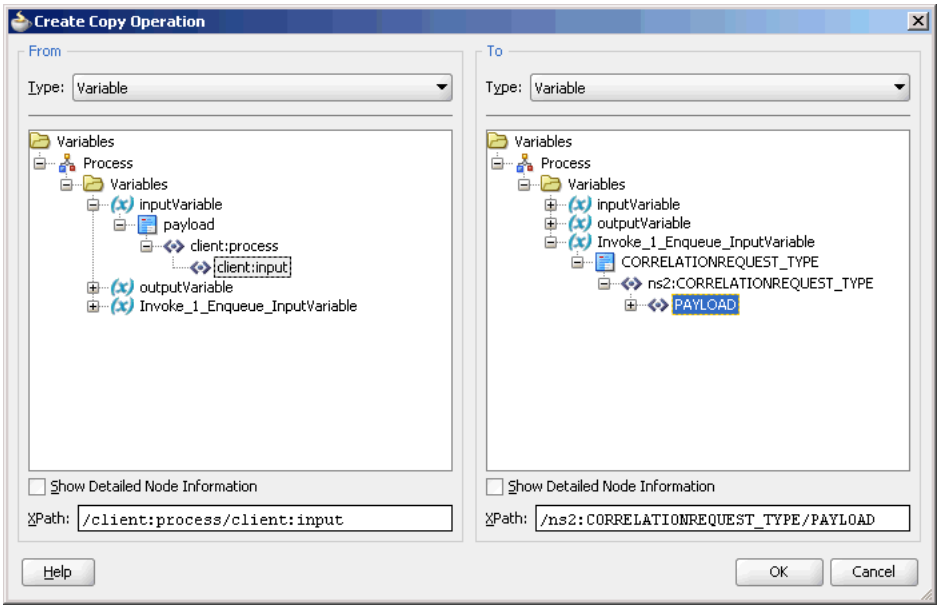
1. Drag and drop an **Assign** activity from the Component Palette to the design area.
2. Double-click the **Assign** activity. The Assign dialog is displayed.
3. Enter a name for the Assign activity in the **Name** field.
4. Click the **Copy Operation** tab.
5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
6. Create a copy operation from inputVariable to outputVariable, as shown in Figure 7–29.

Figure 7–29 The Create Copy Operation Dialog



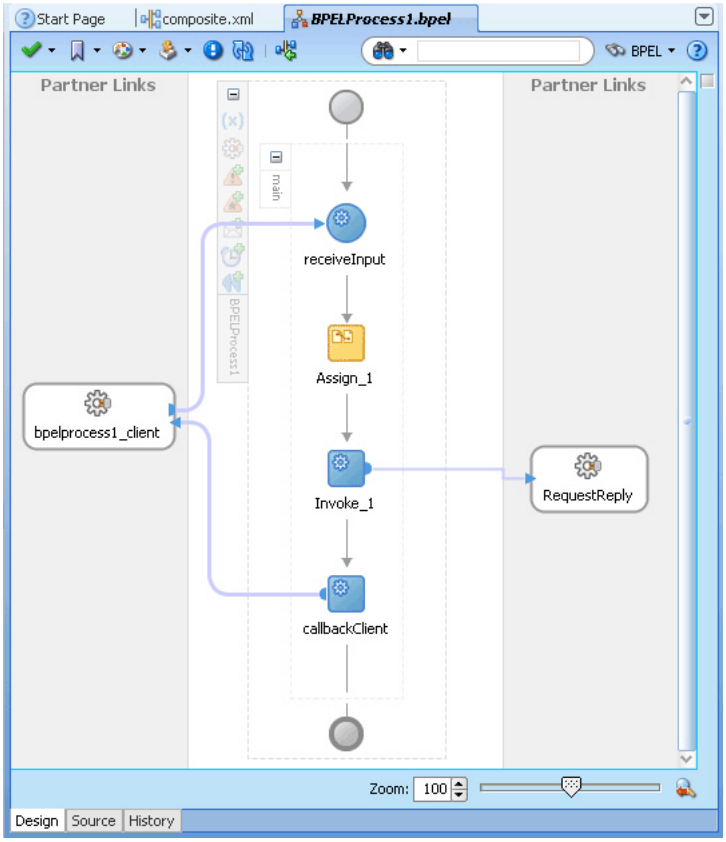
7. Click **OK** in the Create Copy Operation dialog.
8. Create another copy operation from inputVariable to Invoke_1_Enqueue_ InputVariable, as shown in Figure 7–30.

Figure 7–30 The Create Copy Operation Dialog



9. Click **OK** in the Create Copy Operation dialog.
10. Click **OK** to return to the JDeveloper BPELProcess1.bpel page, as shown in Figure 7–31.

Figure 7–31 The JDeveloper - BPELProcess1.bpel

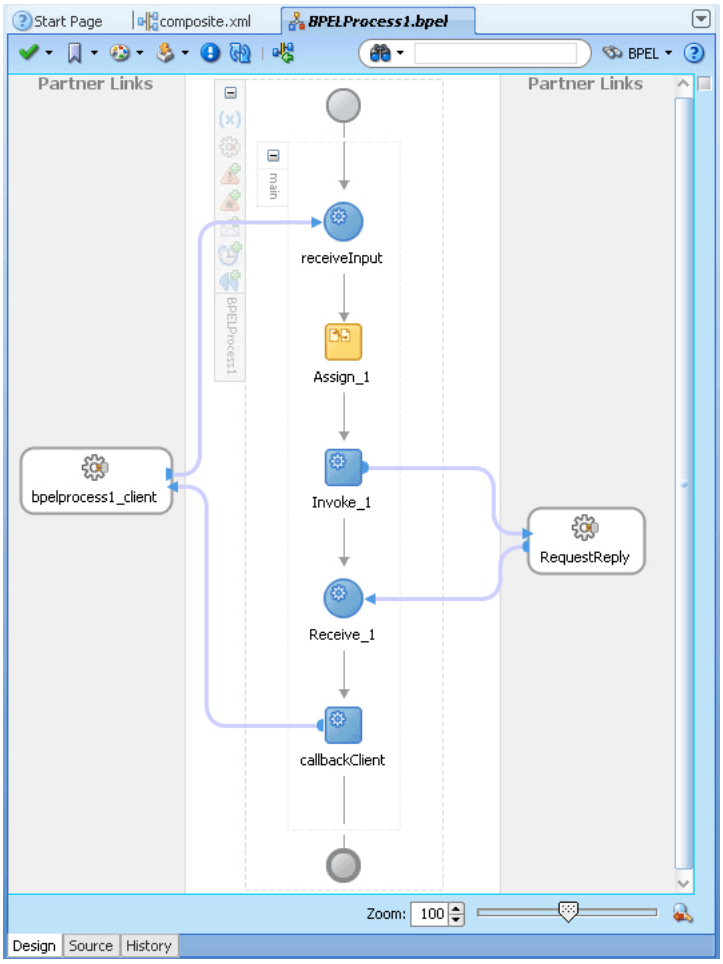


11. Click **File**, **Save All**.

Add a Receive Activity

1. Drag and drop a **Receive** activity from the Component Palette to the design area.
2. Double-click the **Receive** activity. The Receive dialog is displayed.
3. Enter a name for the Receive activity in the **Name** field.
4. Click **Browse Partner Links** at the end of the Partner Link field. The Partner Link Chooser dialog is displayed.
5. Select **RequestReply**, and click **OK**.
6. Click the **Auto-Create Variable** icon to the right of the Variable field in the Receive dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.
8. Check **Create Instance**, and click **OK**. The JDeveloper BPELProcess1.bpel page appears, as shown in *** 'The JDeveloper - BPELProcess1.bpel' on page 41 ***.

Figure 7–32 The JDeveloper - BPELProcess1.bpel



7.4.1.4.5 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps.

The following are the steps to deploy the application profile by using JDeveloper:

1. Create an application server connection by using the procedure described in Chapter 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application by using the procedure described in Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

7.4.1.4.6 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed composite by using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed is displayed in the Application Navigator.
2. Click an instance. The Flow Trace page is displayed.
3. Click the BPEL component instance. The Audit page is displayed.
4. Click the **Flow-Debug** tab to debug the instance.

7.4.1.4.7 Generated WSDL and JCA Files

The following WSDL file is generated for the Enqueue/Dequeue operation:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<?binding.jca Inbound_aq.jca?>
<definitions name="Inbound"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT"
xmlns:imp1="http://www.oracle.com/ipdemo">
    <types>
        <schema attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"
xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT">
            <import namespace="http://xmlns.oracle.com/xdb/SCOTT"
schemaLocation="xsd/SCOTT_MAGAZINE_TYPE.xsd"/>
            <import namespace="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"
schemaLocation="xsd/aqAdapterInboundHeader.xsd"/>
            <complexType name="HeaderCType">
                <sequence>
                    <element name="QueueHeader" type="hdr:HeaderType"/>
                    <element name="PayloadHeader" type="obj1:MAGAZINE_TYPE"/>
                </sequence>
            </complexType>
            <element name="Header" type="tns:HeaderCType"/>
        </schema>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://www.oracle.com/ipdemo"
schemaLocation="xsd/simpleMagazine.xsd"/>
        </schema>
<?xml version = '1.0' encoding = 'UTF-8'?>
   <?binding.jca Inbound_aq.jca?>
    <definitions name="Inbound"
    targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/
     xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
      xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
      xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT"
     xmlns:imp1="http://www.oracle.com/ipdemo">
     <types>
 <schema attributeFormDefault="qualified" elementFormDefault="qualified"
    targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/Inbound/"
xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"

xmlns:obj1="http://xmlns.oracle.com/xdb/SCOTT">
<import namespace="http://xmlns.oracle.com/xdb/SCOTT"
schemaLocation="xsd/SCOTT_MAGAZINE_TYPE.xsd"/>
<import namespace="http://xmlns.oracle.com/pcbpel/adapter/aq/inbound/"

     schemaLocation="xsd/aqAdapterInboundHeader.xsd"/>
<complexType name="HeaderCType">
 <sequence>
<element name="PayloadHeader" type="obj1:MAGAZINE_TYPE"/>
```

```
 </sequence>
</complexType>

   <element name="Header" type="tns:HeaderCType"/>
</schema>

        <schema xmlns="http://www.w3.org/2001/XMLSchema">
       <import namespace="http://www.oracle.com/ipdemo"
        schemaLocation="xsd/simpleMagazine.xsd"/>
</schema>
</types>
<message name="simpleMagazine_msg">

   <part name="simpleMagazine" element="imp1:simpleMagazine"/>
</message>
     <message name="Header_msg">
     <part name="Header" element="tns:Header"/>
</message>
<portType name="Dequeue_ptt">
    <operation name="Dequeue">
       <input message="tns:simpleMagazine_msg"/>
    </operation>
</portType>
    <plt:partnerLinkType name="Dequeue_plt">
        <plt:role name="Dequeue_role">
            <plt:portType name="tns:Dequeue_ptt"/>
        </plt:role>
   </plt:partnerLinkType>
</definitions>
```

The element name QueueHeader is no longer required. As Aman mentioned earlier, the QueueHeader properties are now set using Normalized properties jca.aq.xxxx. You can set these properties from the BPEL dialog for the Receive and Invoke activities.

The following JCA file is generated for the Enqueue/Dequeue operation:

```
<adapter-config name="RequestReply" adapter="AQ Adapter"
wsdlLocation="RequestReply.wsdl"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">

  <connection-factory location="eis/AQ/aqSample" UIConnectionName="aqSample"
adapterRef=""/>
  <endpoint-activation portType="Dequeue_ptt" operation="Dequeue"
UITransmissionPrimitive="Request-response">
    <activation-spec
className="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec">
      <property name="QueueName" value="CORRELATION_REPLY"/>
    </activation-spec>
  </endpoint-activation>

  <endpoint-interaction portType="Enqueue_ptt" operation="Enqueue"
UITransmissionPrimitive="Request-response">
    <interaction-spec
className="oracle.tip.adapter.aq.outbound.AQEnqueueInteractionSpec">
      <property name="QueueName" value="CORRELATION_REQUEST"/>
    </interaction-spec>
  </endpoint-interaction>

</adapter-config>
```

7.4.1.5 Using Correlation ID for Filtering Messages During Dequeue

Perform the following steps to set up an adapter that dequeues messages with a certain correlation ID only.

- Select **Dequeue** operation in Step 10.
- Enter the correlation ID in Step 15.

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

For a working example of this use case where an Oracle AQ Adapter dequeues messages enqueued with that same correlation ID, refer to the following samples:

- `adapters-aq-106-messagerejection`
- `adapters-aq-109-nativecorrelation`
- `adapters-aq-112-prioritymessageselector`
- `adapters-aq-113-payloadbasedmessageselector`

You can obtain these samples by accessing the Oracle SOA Sample Code site.

7.4.1.6 Enqueuing and Dequeuing from Multisubscriber Queues

Multisubscriber queues are accessible by multiple users, and sometimes, those users are concerned only with certain types of messages within the queue. For example, you may have a multiuser queue for loan applications where loans below $100,000 can be approved by regular loan-approval staff, whereas loans over $100,000 must be approved by a supervisor. In this case, the BPEL process can use one adapter to enqueue loan applications for big loans for supervisors, and another adapter to enqueue loan applications for smaller loans for regular staff in the same multisubscriber queue.

Specify an adapter that enqueues to a multisubscriber queue, and include queue parameters in the **Recipients** field.

In Step 15, specify **Bob** in the **Recipients** field.

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

```
<adapter-config name="Inbound" adapter="AQ Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
    <connection-factory location="eis/AQ/aqSample" UIConnectionName="aqSample"
adapterRef=""/>
  <endpoint-interaction portType="Enqueue_ptt" operation="Enqueue">
    <interaction-spec
className="oracle.tip.adapter.aq.outbound.AQEnqueueInteractionSpec">
      <property name="QueueName" value="PURCHASEORDER_APPROVAL"/>
      <property name="RecipientList" value="Bob"/>
    </interaction-spec>
  </endpoint-interaction>
</adapter-config>
```

When dequeuing from a multisubscriber queue, the Queue Parameters window is displayed.

The **Consumer** field is where you specify the consumer name, or the name of the queue subscriber. This must match the **Recipient** entry on the enqueue process for the message to be dequeued. When subscribing to a multiconsumer queue, this field is required.

The following code is from a JCA file generated by defining an Oracle AQ Adapter with a consumer name:

```
<adapter-config name="Dequer_Bob" adapter="AQ Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
    <connection-factory location="eis/AQ/manas" UIConnectionName="aqSample"
adapterRef=""/>
  <endpoint-activation portType="Dequeue_ptt" operation="Dequeue">
    <activation-spec
className="oracle.tip.adapter.aq.inbound.AQDequeueActivationSpec">
      <property name="QueueName" value="PURCHASEORDER_APPROVAL"/>
      <property name="Consumer" value="Bob"/>
      <property name="SchemaValidation" value="false"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

For a working example of this use case which demonstrates enqueuing and dequeuing from multisubscriber queues, refer to the following samples:

- `adapters-aq-114-multiconsumeroutbound`

You can obtain these samples by accessing the Oracle SOA Sample Code site.

7.4.2 Oracle AQ Adapter ADT Queue

In this sample, the business process receives a message from the AQ Adapter, copies the payload to an outbound message, and invokes the AQ Adapter with the outbound message.The queues involved are ADT queues. In this scenario, where the user has chosen to use whole ADT as the payload, the AQ Adapter Wizard has generated the schema in `SCOTT_CUSTOMER_TYPE.xsd`, according to the queue structure. During run time, an XML file that matches the schema is created by the adapter for each message.

This section includes the following topics:

- Section 7.4.2.1, "Meeting Prerequisites"
- Section 7.4.2.2, "Creating an Application and an SOA Project"
- Section 7.4.2.3, "Creating an Inbound Oracle AQ Adapter"
- Section 7.4.2.4, "Creating an Outbound Oracle AQ Adapter"
- Section 7.4.2.5, "Wiring Services and Activities"
- Section 7.4.2.6, "Configuring Routing Service"
- Section 7.4.2.8, "Deploying with JDeveloper"
- Section 7.4.2.7, "Configuring the Data Sources in the Oracle WebLogic Server Administration Console"

7.4.2.1 Meeting Prerequisites

You must have access to a database with the SCOTT schema.

To perform this use case, you require the following SQL files from the `artifacts.zip` file contained in the `adapters-aq-102-adt` sample. These files are located in the `artifacts/sql` subdirectory of the `artifacts.zip` fle. Execute the SQL files in the order shown below:

- `setup_user.sql`

- `create_type_customer.sql`
- `create_queues.sql`
- `enqueue_customer.sql`
- `dequeue_customer.sql`

To obtain the adapters-aq-102-adt sample code, access the Oracle SOA Sample Code site

7.4.2.2 Creating an Application and an SOA Project

You must create an JDeveloper application to contain the SOA composite. Use the following steps to create an application and an SOA project:

1. In the **Application Navigator** of JDeveloper, click **New Application**.

   The Create Generic Application Name your application page is displayed.

2. Enter `ADT` in the **Application Name** field, and click **Next**.

   The Create Generic Application Name your project page is displayed.

3. Enter `ADT` in the **Project Name** field.

4. In the Available list in the Project Technologies tab, double-click **SOA** to move it to the Selected list.

5. Click **Next**.

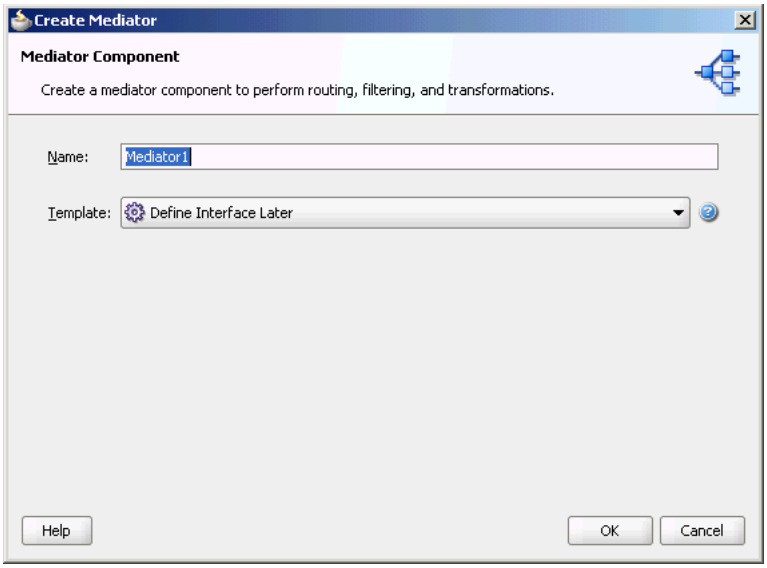
   The Create Generic Application Configure SOA settings page is displayed.

6. Select **Composite With Mediator** from the Composite Template list, and then click **Finish**.

   You have created a new application and an SOA project.

   The Create Mediator page is displayed, as shown in Figure 7–33.

Figure 7–33 The Create Mediator Page



7. Enter a name for the Mediator component in the **Name** field. In this example, retain the default name `Mediator1`.

8. Select **Define Interface Later** in the Template list, and then click **OK**.

   You have created a mediator component.

7.4.2.3 Creating an Inbound Oracle AQ Adapter

The following are the steps to create an inbound Oracle AQ Adapter service:

1. In the Component Palette, select **SOA**.
2. Drag and drop **AQ Adapter** from the Service Adapters list in the Component Palette to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard is displayed.
3. Click **Next**.

   The Service Name page is displayed.
4. Specify a name for the service in the Service Name page. In this example, type `dequeue`.
5. Click **Next**.

   The Service Connection page is displayed. A database connection is required to configure an Oracle AQ Adapter. You can either create a new connection or select an existing database connection.
6. Click the **Create a new database connection** icon to create a database connection.

   The Create Database Connection page is displayed.
7. Create a database connection, as mentioned in Step 6 of Section 7.4.1.1.3, "Defining an Oracle AQ Adapter Service."
8. Click **OK** to complete the process of creating a new database connection.

   The Service Connection page is displayed, providing a summary of the database connection.
9. Click **Next**.

   The Adapter Interface page is displayed.
10. In the Adapter Interface page, select **Define from operation and schema (specified later)**.
11. Click **Next**.

    The Operation page is displayed.
12. Select **Dequeue**.
13. Accept the default operation name, and then click **Next**.

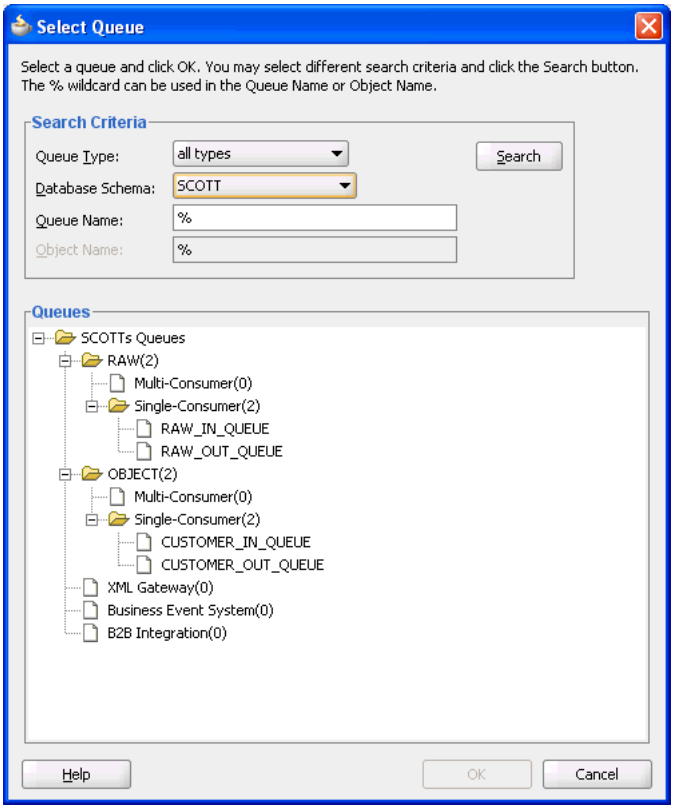
    The Queue Name page is displayed.
14. Select a database schema from the list, or click **Browse** to browse for the schema. In this example, click **Browse**.

    The Select Queue dialog is displayed.
15. In the Select Queue dialog, perform the following steps:
    a. For Queue Type, select **all types**.
    b. For Database Schema, select **Scott**.
    c. Retain the default values for the other fields.

d. Under Queues, select **CUSTOMER_IN_QUEUE**.

Figure 7–34 shows the Select Queue dialog.

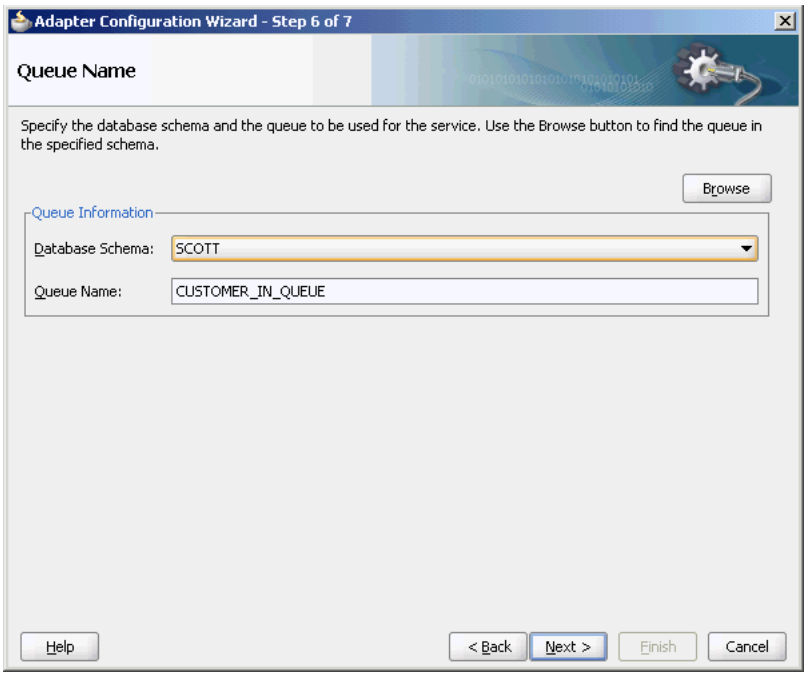
Figure 7–34 Selecting a Queue for the Inbound Operation



16. Click **OK**.

The Queue Name dialog with all the fields populated is displayed, as shown in Figure 7–35.

Figure 7–35 The Queue Name Page



17. Click **Next**.

    The Queue Parameters page is displayed.

18. In the Queue Parameters page, leave the fields empty, and then click **Next**.

    The Object Payload page is displayed.

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

    In this example, select **Whole Object CUSTOMER_TYPE**.

20. Click **Next**.

    The Finish screen is displayed. This page shows the path and name of the adapter file that the wizard creates.

21. Click **Finish**.

    You have defined an inbound Oracle AQ Adapter

7.4.2.4 Creating an Outbound Oracle AQ Adapter

The following are the steps to create an outbound Oracle AQ Adapter service:

1. In the Component Palette, select **SOA**.
2. Drag and drop **AQ Adapter** from the Service Adapters list in the Component Palette to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard is displayed.

3. Click **Next**.

   The Service Name page is displayed.

4. In the Service Name field, enter `enqueue` and click **Next**.

   The Service Connection page is displayed.

5. For Connection, select **MyConnection**, and then click **Next**.

   The Adapter Interface page is displayed.

6. In the Adapter Interface page, select **Define from operation and schema (specified later)**, and then click **Next**.

   The Operation page is displayed.

7. In the Operation page, select **Enqueue**, and accept the default operation name.

8. Click **Next**.

   The Queue Name page is displayed.

9. In the Queue Name page, select a database schema from the list, or click **Browse** to browse for the schema. In this example, click **Browse**.

   The Select Queue dialog is displayed.

10. In the Select Queue dialog, perform the following steps:

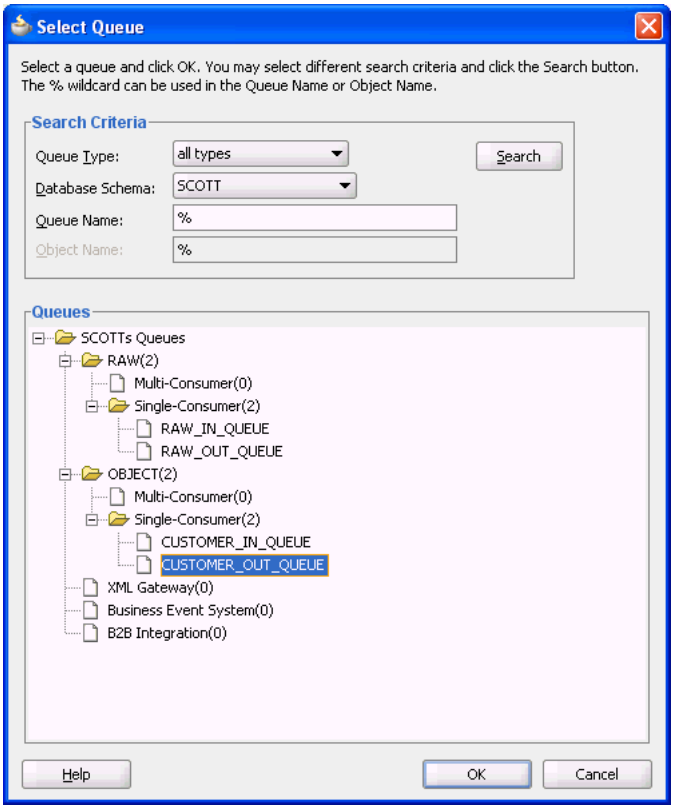
    a. For Queue Type, select **all types**.

    b. For Database Schema, select **Scott**.

    c. Retain the default values for the other fields.

    d. Under Queues, select **CUSTOMER_OUT_QUEUE**, as shown in Figure 7–36.

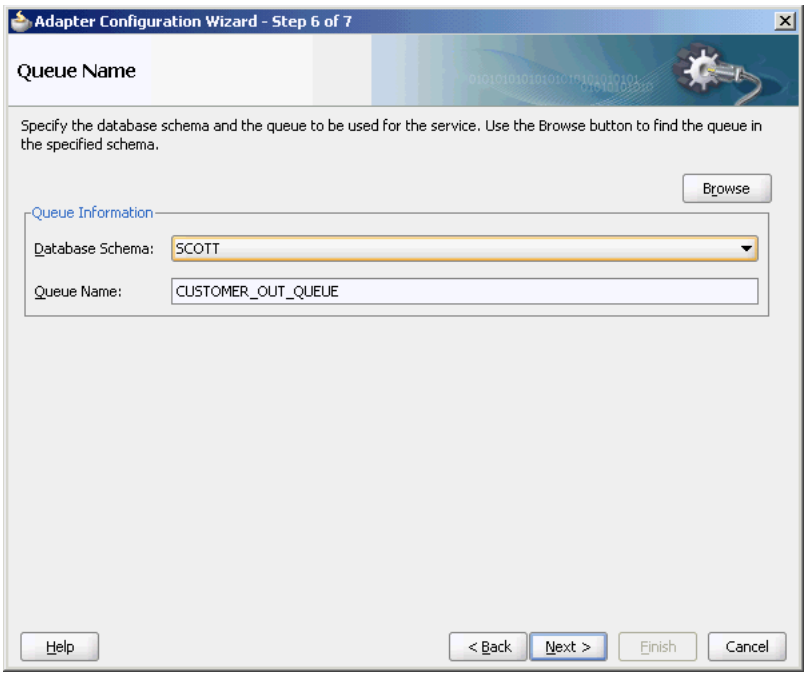
Figure 7–36 Selecting a Queue for the Outbound Operation



11. Click **OK**.

    The Queue Name page with all the fields populated is displayed, as shown in Figure 7–37.

Figure 7–37 The Queue Name Page



12. Click **Next**.

    The Queue Parameters page is displayed.

13. In the Queue Parameters page, leave the fields empty, and then click **Next**.

    The Object Payload page is displayed.

14. Select a business payload, either the entire object, or just one field within the object. In this example, select **Whole Object CUSTOMER_TYPE**.

15. Click **Next**.

    The Finish screen is displayed. This page shows the path and name of the adapter file that the wizard creates.

16. In the Finish window, click **Finish**.

    You have defined an outbound Oracle AQ Adapter.

7.4.2.5 Wiring Services and Activities

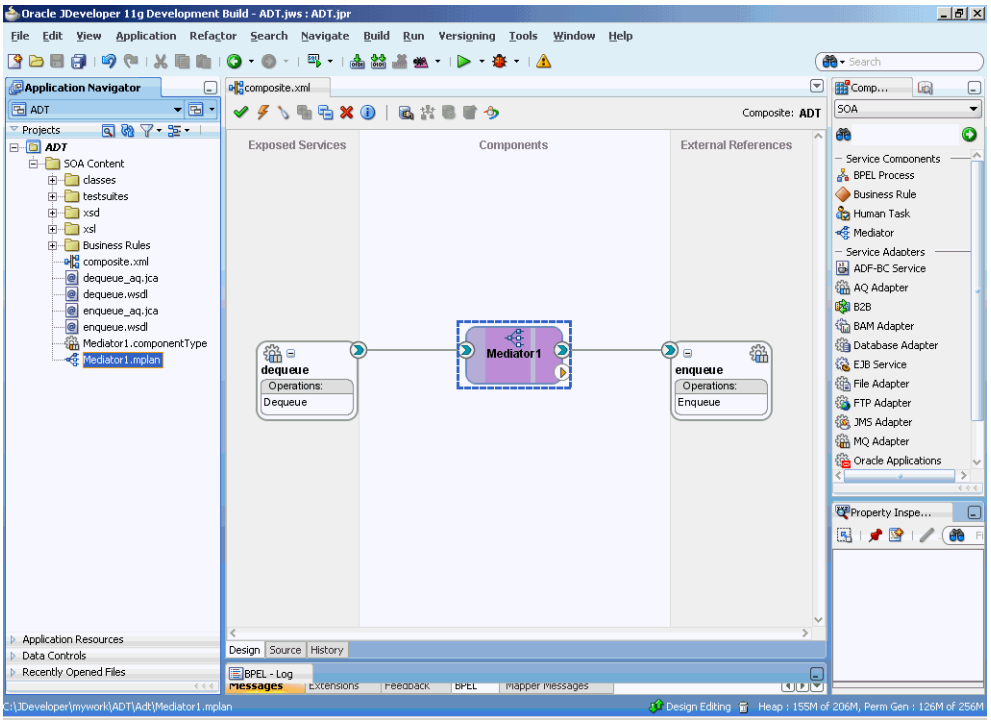
You must assemble or wire the three components that you have created: Inbound adapter service, Mediator component, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the Inbound adapter in the Exposed Services area to the drop zone that appears as a green triangle in the Mediator component in the Components area.

2. Drag the small triangle in the Mediator component in the Components area to the drop zone that appears as a green triangle in the Outbound adapter in the External References area.

   The JDeveloper composite.xml is displayed, as shown in Figure 7–38.

Figure 7–38 The JDeveloper composite.xml



3. Click **File**, **Save All**.

7.4.2.6 Configuring Routing Service

The following are the steps to configure the routing service:

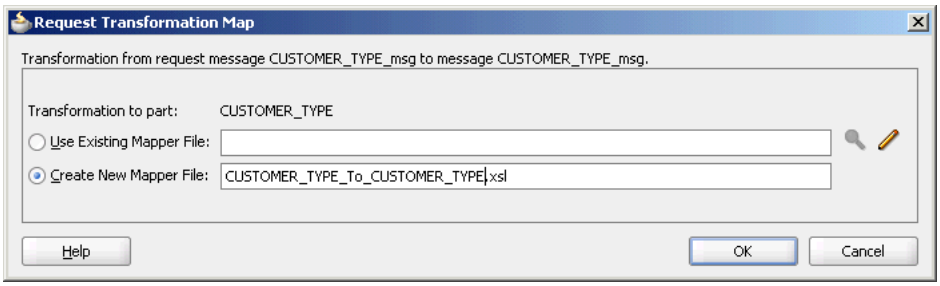
1. Double-click **Mediator1**.

   The Mediator1.mplan window is displayed.

2. Click the **Select an existing mapper file or create a new one...** icon that is displayed at the end of the Transform Using field.

   The Request Transformation Map dialog is displayed, as shown in Figure 7–39.

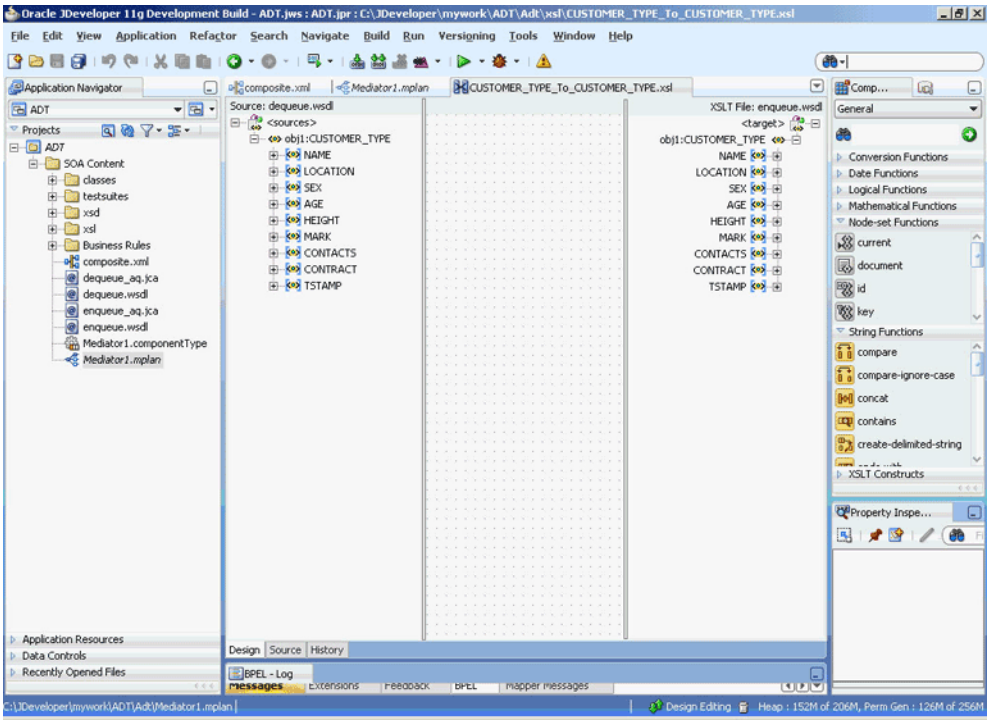
Figure 7–39 The Request Transformation Map Dialog



3. Select **Create New Mapper File**, and then click **OK**.

   The Transformation window is displayed, as shown in Figure 7–40.

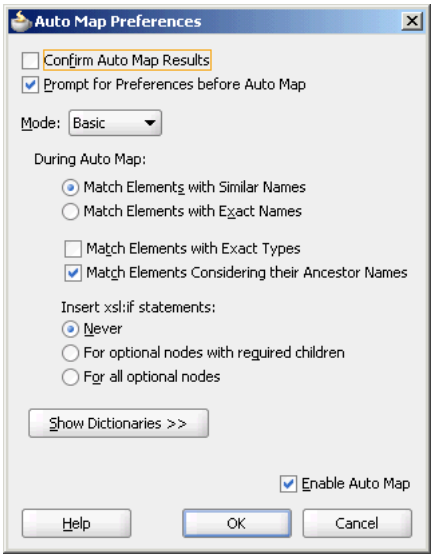
Figure 7–40 The Transformation Window



4. Select the source root elements on the left-hand side of the mapper and drag them over to the destination root elements on the right-hand side to set the map preferences.

   The Auto Map Preferences dialog is displayed, as shown in Figure 7–41.

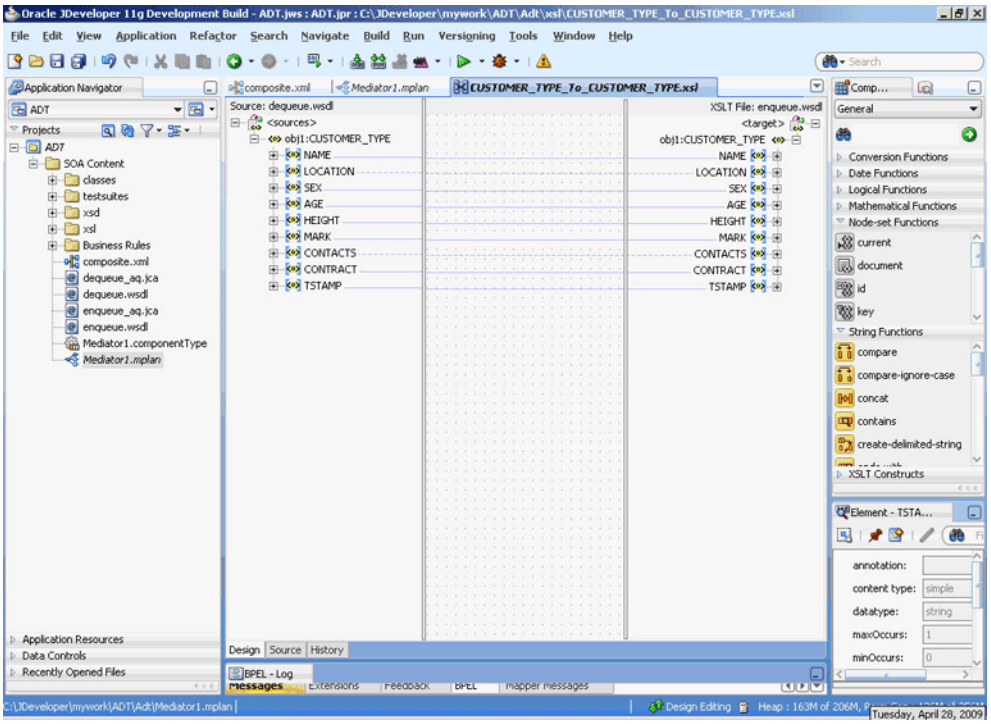
Figure 7–41 The Auto Map Preferences Dialog



5. Click **OK**.

   The middle pane of the application window appears as shown in Figure 7–42.

Figure 7–42 The Application Window After Setting the Map Preferences



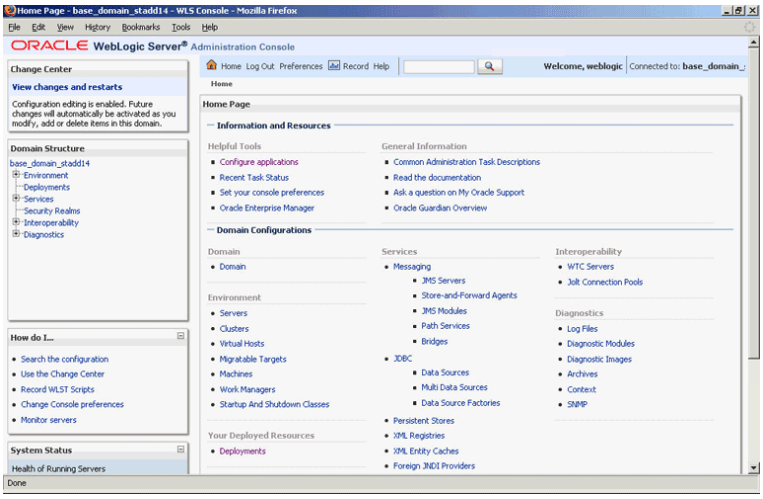
6. Save and close the tab for the mapper.
7. Save and close the tab for the routing service.

7.4.2.7 Configuring the Data Sources in the Oracle WebLogic Server Administration Console

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

   The Home page of the Oracle WebLogic Server Administration Console is displayed, as shown in Figure 7–43.

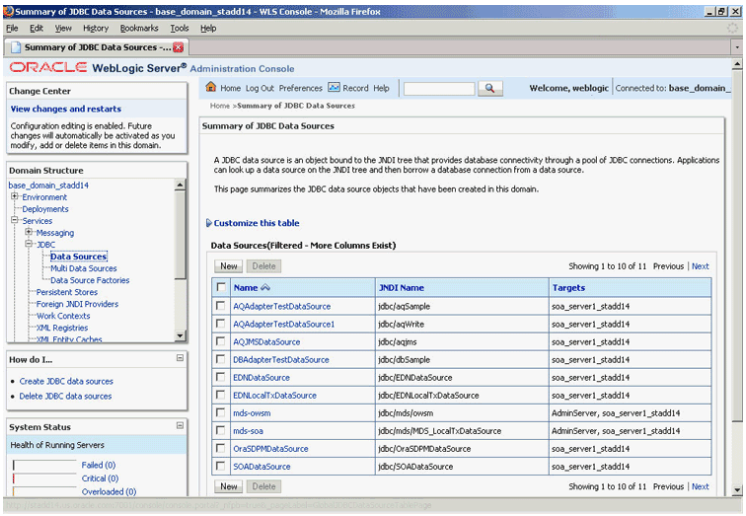
Figure 7–43 Oracle WebLogic Server Administration Console Home Page



3. Under Domain Structure, select **Services**, **JBDC**, and then click **DataSources**.

   The Summary of JDBC Data Sources page is displayed, as shown Figure 7–44.

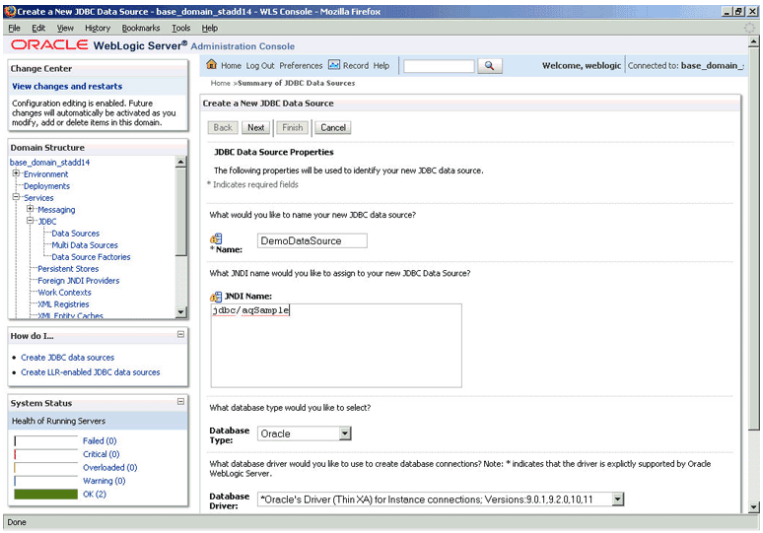
Figure 7–44 The Summary of JDBC Data Sources Page



4. Click **New**. The Create a New JDBC Data Source page is displayed.

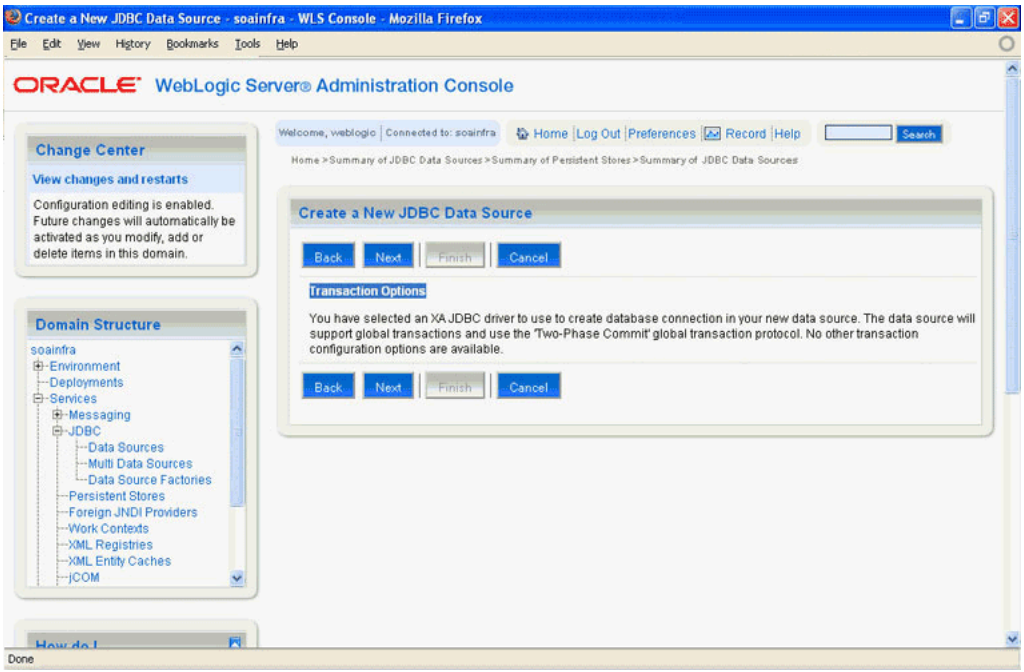
5. Enter the values for the properties to be used to identify your new JDBC data source, as shown in Figure 7–45.

Figure 7–45 The Create a New JDBC Data Source Page



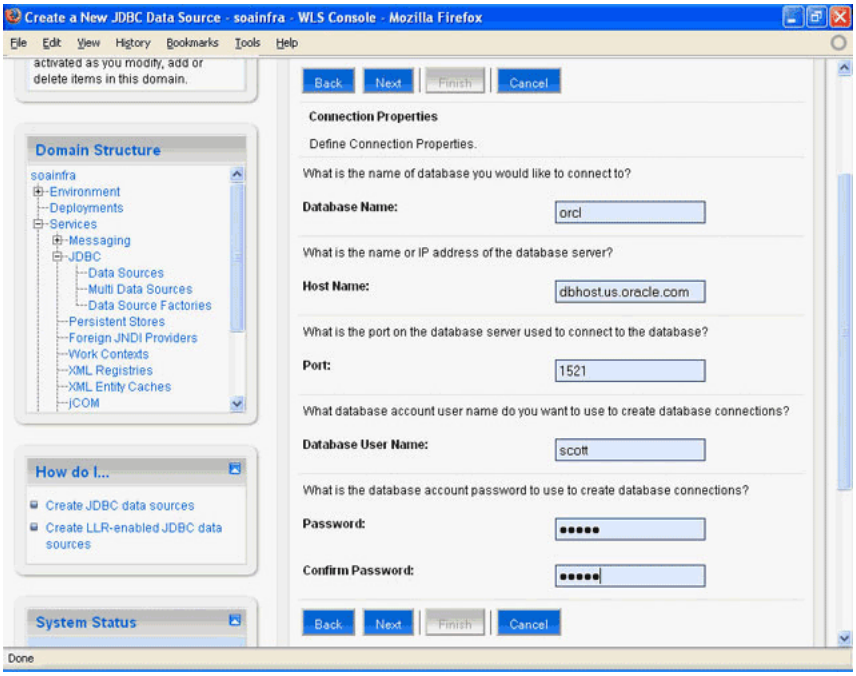
6. Click **Next**. The Create a New JDBC Data Source Transaction Options page is displayed, as shown in Figure 7–46.

Figure 7–46 The Create a New JDBC Data Source Transaction Options Page



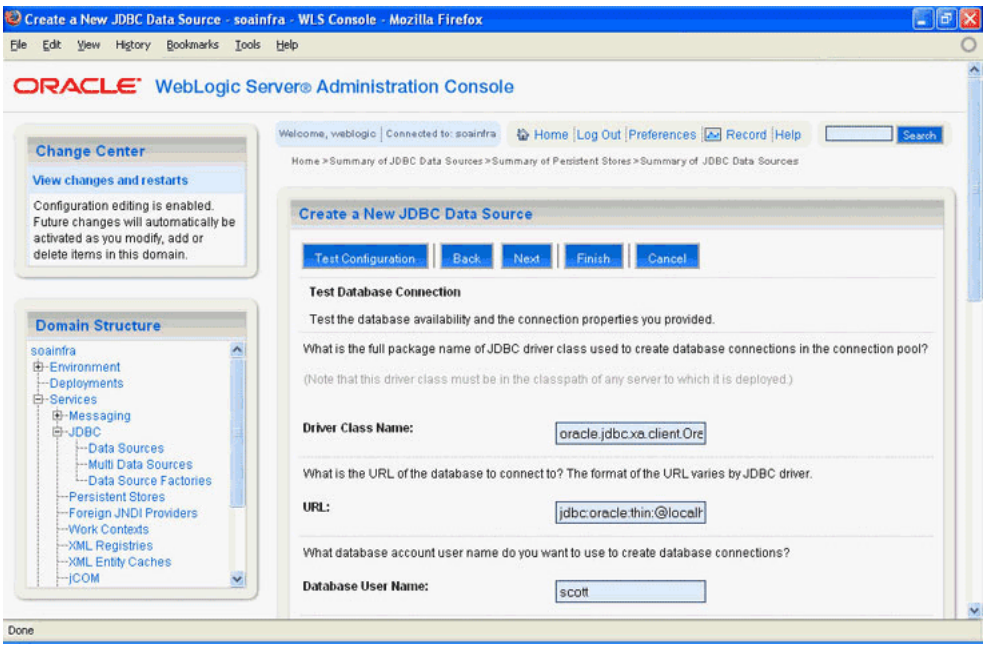
7. Click **Next**. The Create a New JDBC Data Source Connection Properties page is displayed, as shown in Figure 7–47.

Figure 7–47 The Create a New JDBC Data Source Connection Properties Page



8. Enter the connection properties in the Connection Properties page.
9. Click **Next**. The Create a New JDBC Data Source Test Database Connection page is displayed, as shown in Figure 7–48.

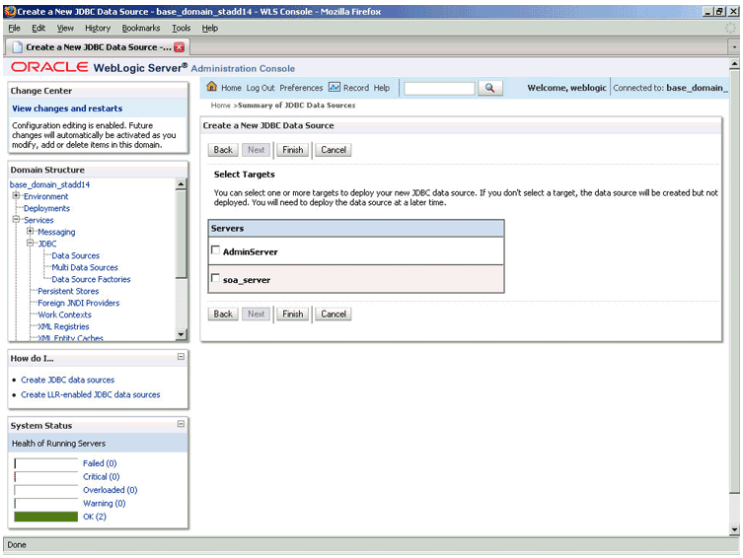
Figure 7–48 The Create a New JDBC Data Source Test Database Connection Page



10. Click **Test Configuration** to test the database availability and the connection properties you provided. A message stating that the connection test succeeded is displayed at the top of the Create a New JDBC Data Source Test Database Connection page.

11. Click **Next**. The Create a New JDBC Data Source Select Targets page is displayed, as shown in Figure 7–49.

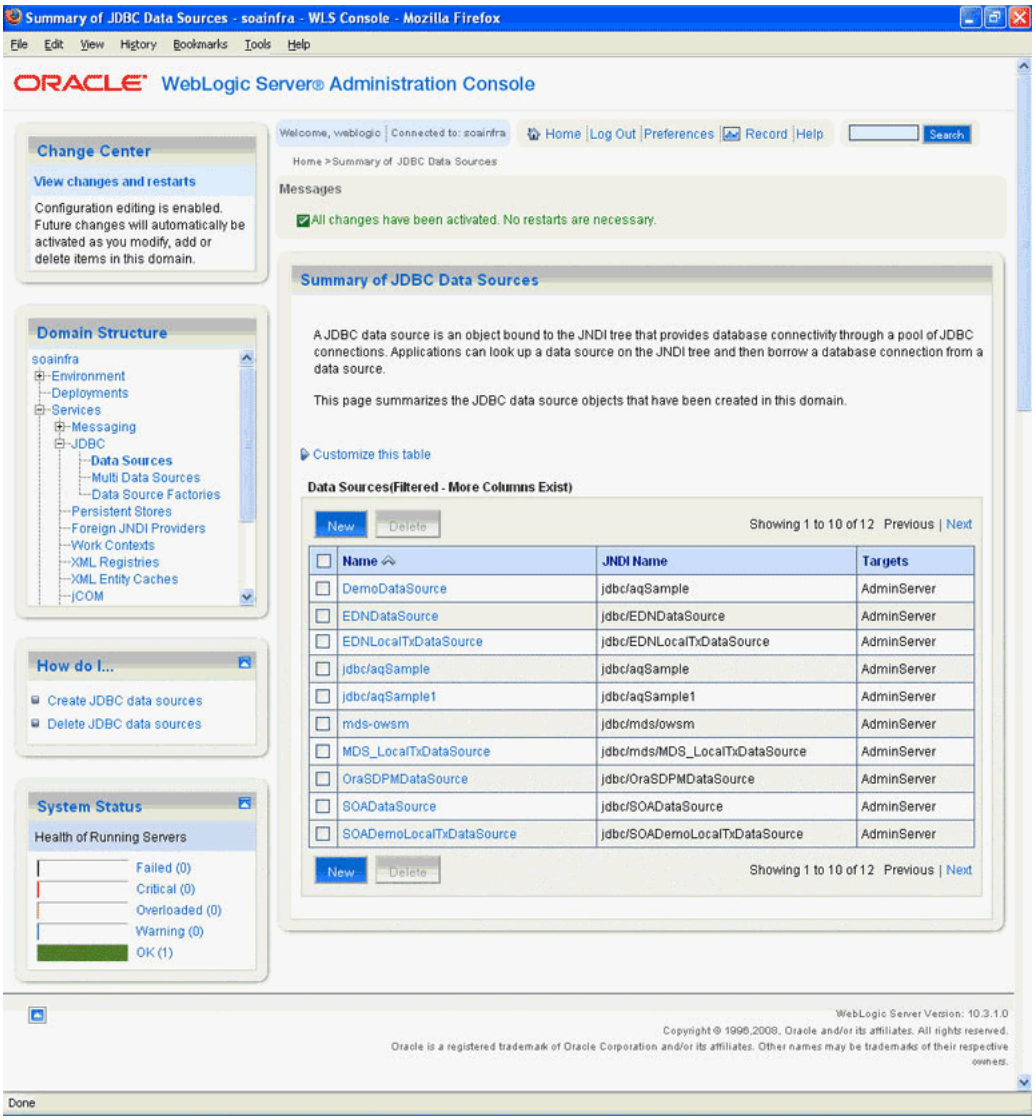
Figure 7–49 The Create a New JDBC Data Source Select Targets Page



12. Select a target, and then click **Finish**.

    The Summary of JDBC Data Sources page is displayed, as shown in Figure 7–50. This page summarizes the JDBC data source objects that have been created in this domain. The data source that you created appears in this list.

Figure 7–50 The Summary of JDBC Data Sources Page



13. Close the Oracle WebLogic Server Administration Console.

7.4.2.8 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps.

The following are the steps to deploy the application profile by using JDeveloper:

1. Create an application server connection by using the procedure described in Chapter 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application by using the procedure described in Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

7.4.2.9 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed composite by using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed is displayed in the Application Navigator.
2. In the Last 5 Instances pane, there is an entry of a new instance. This new instance is the instance that was triggered when you enqueued a message.
3. Click an instance. The Flow Trace page is displayed.
4. Click the **Mediator1** component instance. The Audit page is displayed.
5. Click the **Flow-Debug** tab to debug the instance.

7.4.3 Oracle AQ Adapter RAW Queue

This use case demonstrates how to use Oracle AQ Adapter to dequeue from and enqueue to an AQ RAW queue.

This section includes the following topics:

- Section 7.4.3.1, "Prerequisites"
- Section 7.4.3.2, "Creating an Application and an SOA Project"
- Section 7.4.3.3, "Creating an Inbound Adapter Service"
- Section 7.4.3.4, "Creating an Outbound Adapter Service"
- Section 7.4.3.5, "Wiring Services and Activities"
- Section 7.4.3.7, "Deploying with JDeveloper"
- Section 7.4.3.6, "Configuring the Data Sources in the Oracle WebLogic Server Administration Console"
- Section 7.4.3.8, "Monitoring Using the Fusion Middleware Control Console"

7.4.3.1 Prerequisites

You must have access to a database with the `SCOTT` schema.

To perform this use case, you require the following SQL files from the `artifacts.zip` file contained in the `adapters-aq-101-raw` sample. These files are located in the `artifacts/sql` subdirectory of the `artifacts.zip` fle. Execute the SQL files in the order shown below:

- `setup_user.sql`
- `create_queues.sql`
- `enqueue_raw.sql`
- `dequeue_raw.sql`
- `artifacts/schemas/emp.xsd`

To obtain the `artifacts.zip` contained in the `adapters-aq-101-raw` sample code, access the Oracle SOA Sample Code site.

To obtain the `adapters-aq-101-raw` sample code, access the Oracle SOA Sample Code site.

7.4.3.2 Creating an Application and an SOA Project

You must create an JDeveloper application to contain the SOA composite. To create an application and an SOA project, perform the following steps:

1. Open JDeveloper.

2. In the Application Navigator, click **New Application**.

   The Create Generic Application Name your Application page is displayed.

3. Enter `Rawqueue` in the **Application Name** field.

4. In the Application Template list, select **Generic Application**.

5. Click **Next**.

   The Create Generic Application Name your project page is displayed.

6. In the **Project Name** field, enter a descriptive name, for example, `Raw`.

7. In the Available list in the Project Technologies tab, double-click **SOA** to move it to the Selected list.

8. Click **Next**.

   The Create Generic Application Configure SOA settings page is displayed.

9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

   You have created a new application and an SOA project. This automatically creates an SOA composite.

   The Create BPEL Process page is displayed.

10. Enter a name for the BPEL process in the **Name** field. For example, `BPELRawqueue`.

11. Select **Define Service Later** in the Template list, and then click **OK**.

    The `Rawqueue` application and the `Raw` project appear in the design area.

12. Copy the `emp.xsd` file to the `XSD` folder in your project (see Section 7.4.3.1, "Prerequisites" for the location of this file).

7.4.3.3 Creating an Inbound Adapter Service

Perform the following steps to create an inbound Oracle AQ Adapter service that dequeues the message to a queue:

1. In the Component Palette, select **SOA**.

2. Drag and drop **AQ Adapter** from the Service Adapters list in the Component Palette to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard is displayed.

3. Click **Next**.

   The Service Name page is displayed.

4. In the Service Name field, enter `Raw-Dequeuer`, and then click **Next**.

   The Service Connection page is displayed.

5. Create a database connection, as mentioned in Step 6 of Section 7.4.1.1.3, "Defining an Oracle AQ Adapter Service."

6. Click **Next**.

   The Adapter Interface page is displayed.

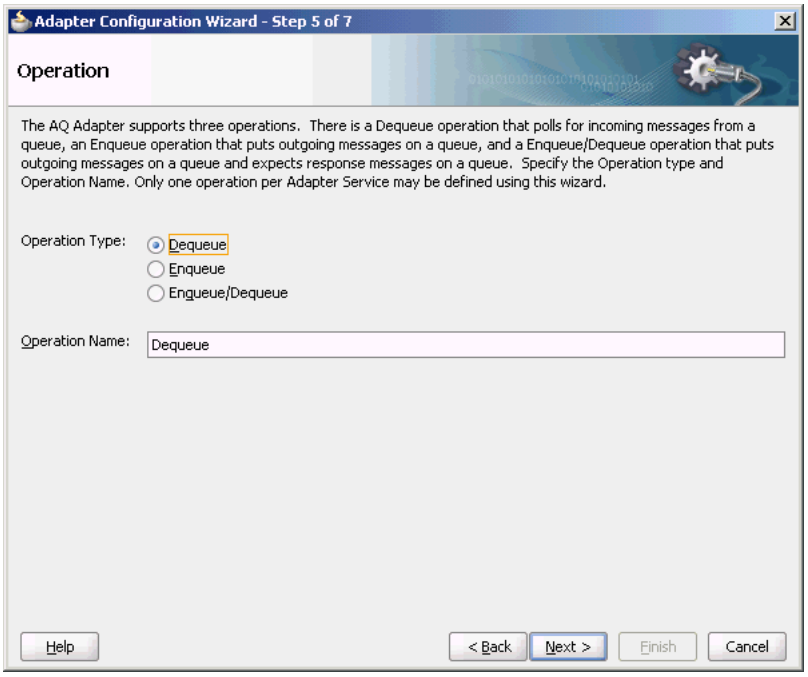
7. In the Adapter Interface page, select **Define from operation and schema (specified later)**, and then click **Next**.

   The Operation page is displayed.

8. In the Operation page, select **Dequeue**, as shown in Figure 7–51.
9. Accept the default operation name, and click **Next**.

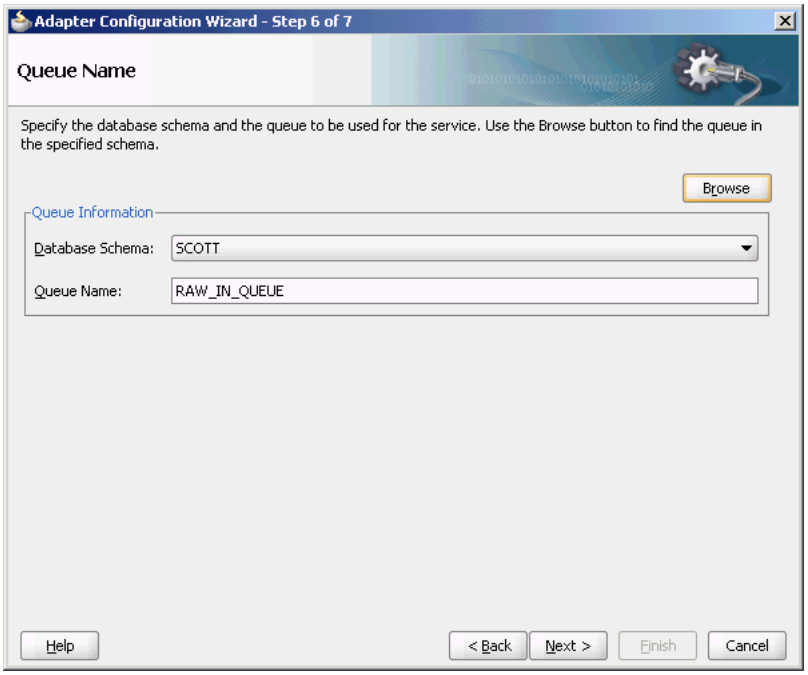
   The Queue Name page is displayed.

Figure 7–51 The Adapter Configuration Wizard Operation Page



10. In the Queue Name page, select **SCOTT** as Database Schema and **RAW_IN_QUEUE** as Queue Name, as shown in Figure 7–52.

Figure 7–52 The Adapter Configuration Wizard Queue Name Page



11. Click **Next**.

    The Queue Parameters page is displayed.

12. Enter the Correlation ID and a Dequeue condition, and then click **Next**.

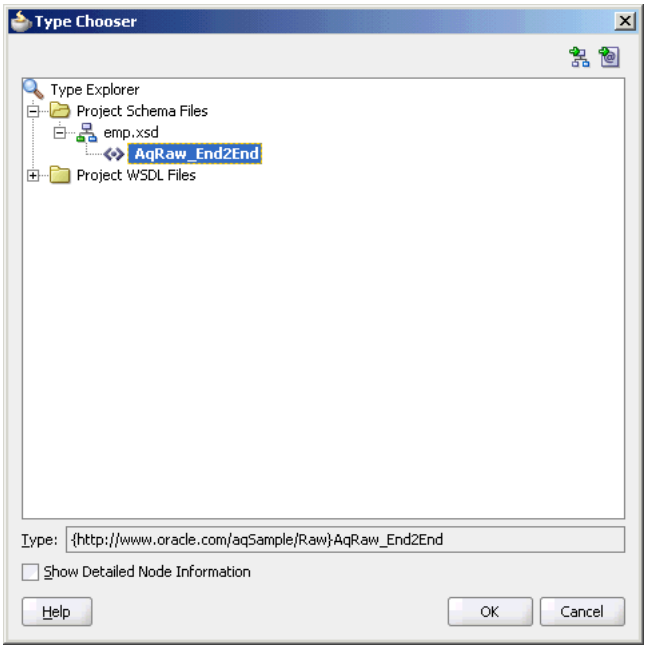
    The Messages page is displayed.

13. Click **Browse** at the end of the URL field.

    The Type Chooser dialog is displayed.

14. Select **Project Schema Files**, **emp.xsd**, and then **AQRaw_End2End**, as shown in Figure 7–53.

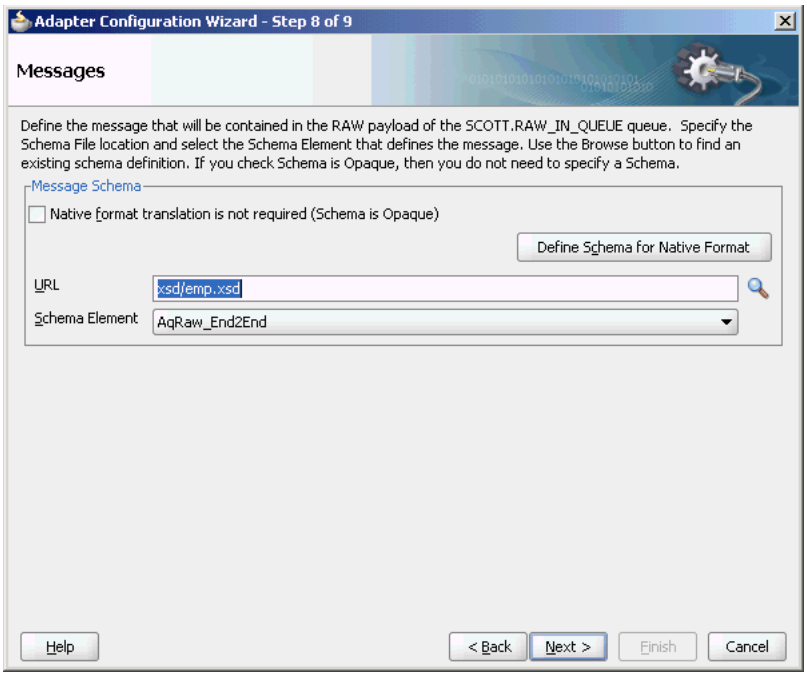
Figure 7–53 The Type Chooser Dialog



15. Click **OK**.

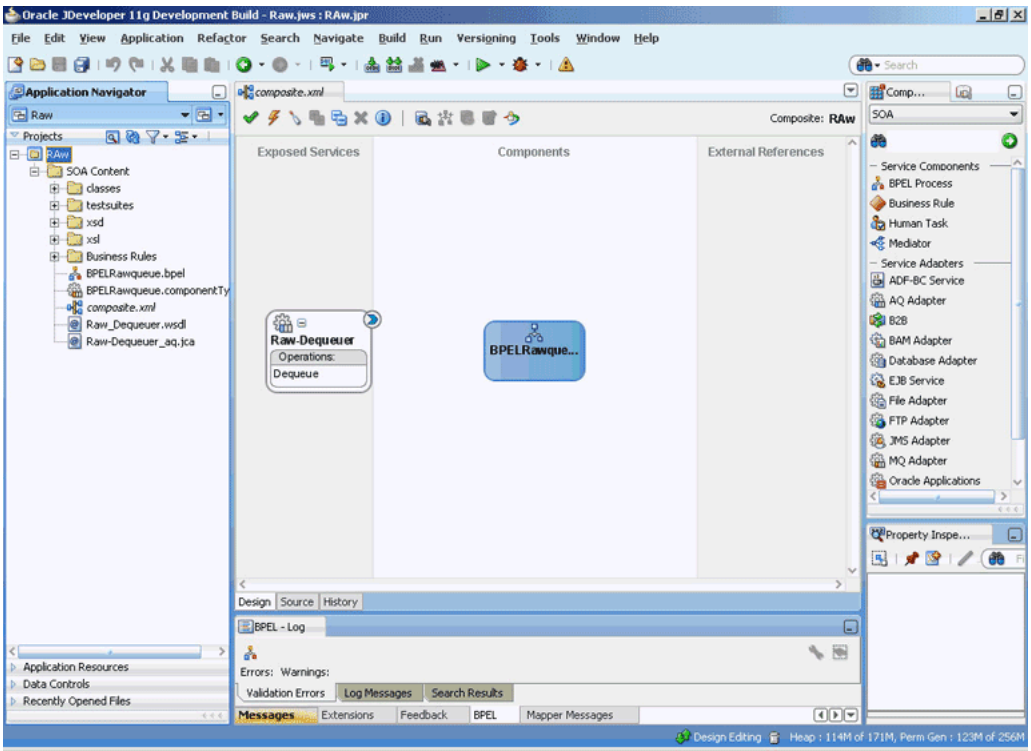
    The `emp.xsd` schema file is displayed in the URL field in the Message dialog, as shown in Figure 7–54.

Figure 7–54 The Adapter Configuration Wizard Messages Page



16. Click **Next**. The Finish page is displayed.
17. Click **Finish**. You have configured the Oracle AQ Adapter service, and the `composite.xml` page is displayed, as shown in Figure 7–55.

Figure 7–55 The JDeveloper Window Composite.xml Page



7.4.3.4 Creating an Outbound Adapter Service

Perform the following steps to create an adapter service that enqueues the request messages and dequeue the corresponding response messages (report) from a queue:

1. Drag and drop **AQ Adapter** from the Service Adapters list in the Component Palette to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard Welcome page is displayed.

2. Click **Next**. The Service Name page is displayed.
3. Enter `Raw-Enqueuer` in the **Service Name** field, and click **OK**.

   The Service Connection page is displayed.

4. Select **XA Datasource**, and then click **Next**.

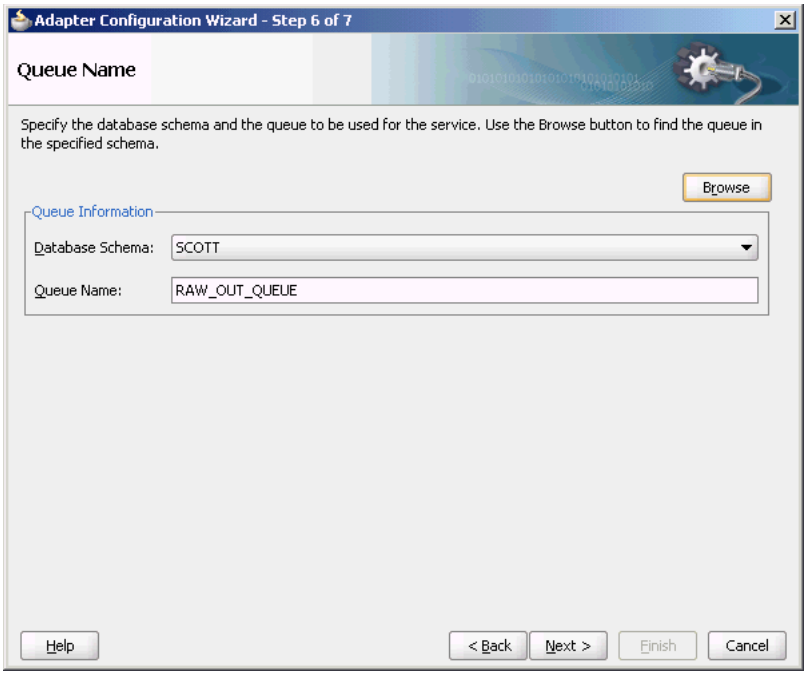
   The Operation page is displayed.

5. Select **Enqueue**.
6. Accept the default operation name, and click **Next**.

   The Queue Name page is displayed.

7. Select **SCOTT** as Database Schema and **RAW_OUT_QUEUE** as Queue Name, as shown in Figure 7–56.

Figure 7–56 The Adapter Configuration Wizard Queue Name Page



8. Click **Next**.

   The Queue Parameters page is displayed.

9. Enter the Correlation ID, and then click **Next**.

   The Messages page is displayed.

10. Click **Browse for schema file** at the end of the URL field.

    The Type Chooser dialog is displayed.

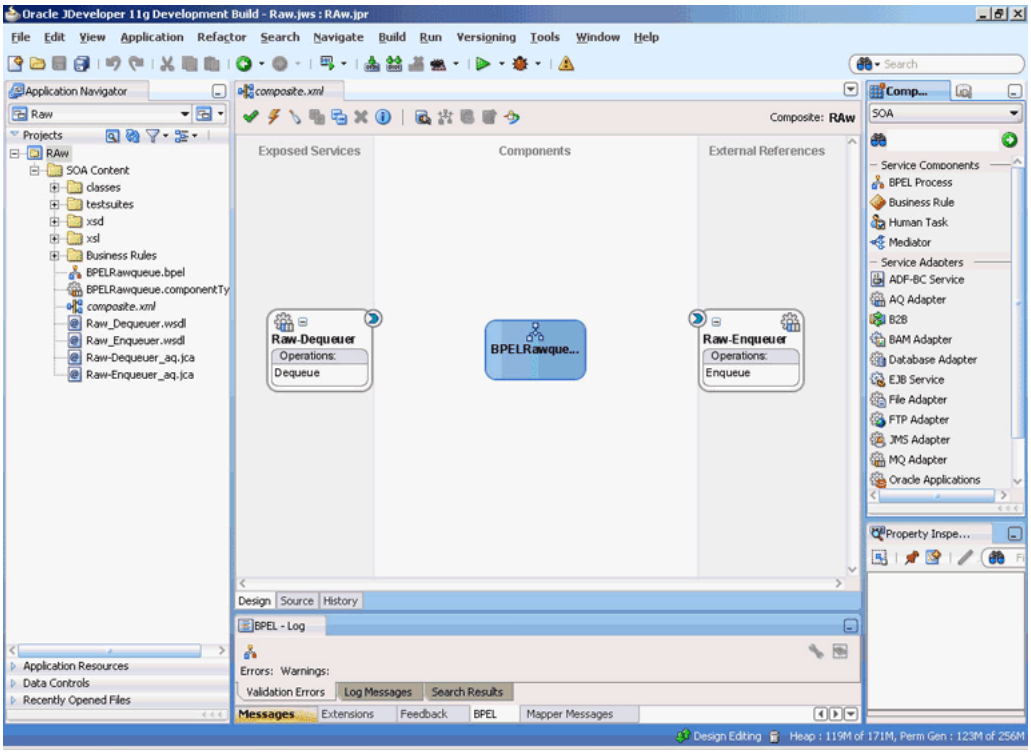
11. Select **Project Schema Files**, **emp.xsd**, and **AQRaw_End2End**, as shown in Figure 7–53.
12. Click **Next**.

    The emp.xsd schema file is displayed in the URL field in the Message dialog, as shown in Figure 7–54.
13. Click **Next**.

    The Finish page is displayed.
14. Click **Finish**.

    You have configured the Oracle AQ Adapter service, and the composite.xml page is displayed, as shown in Figure 7–57.

Figure 7–57 The JDeveloper Window Composite.xml Page



7.4.3.5 Wiring Services and Activities

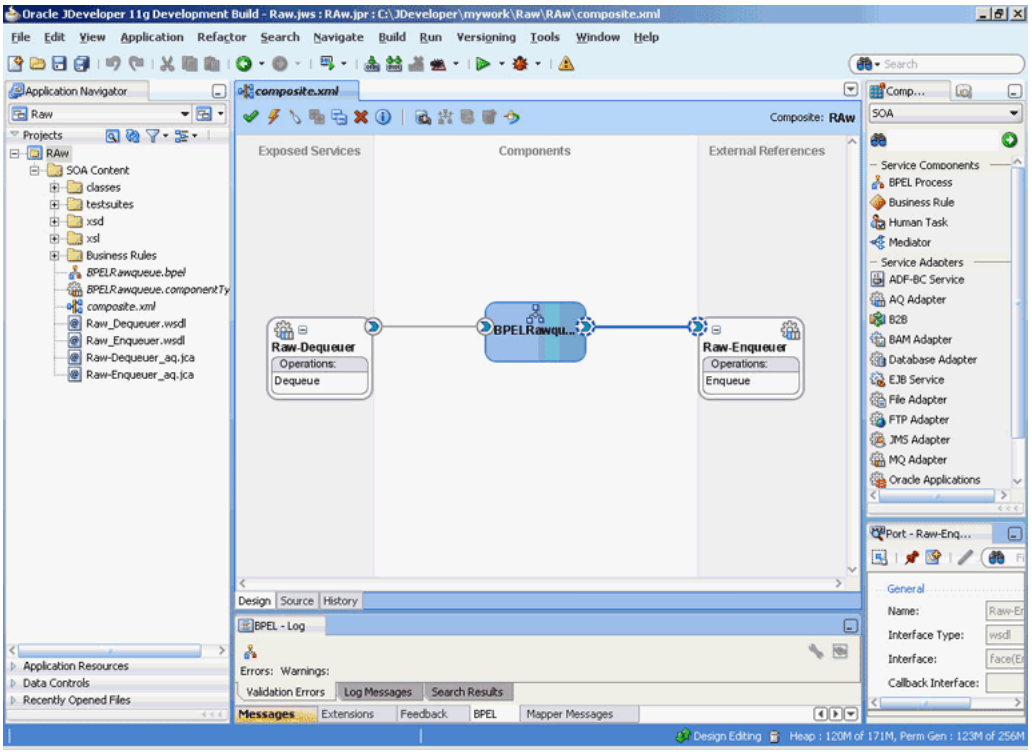
You must assemble or wire the three components that you have created: Inbound adapter service, BPEL process, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the Raw-Dequeuer in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in Raw-Enqueuer in the External References area.

   Similarly, drag the small triangle in the BPEL process in the Components area to the drop zone in OutboundService in the External References.

The JDeveloper `composite.xml` file is displayed, as shown in Figure 7–58.

Figure 7–58 The JDeveloper- Composite.xml

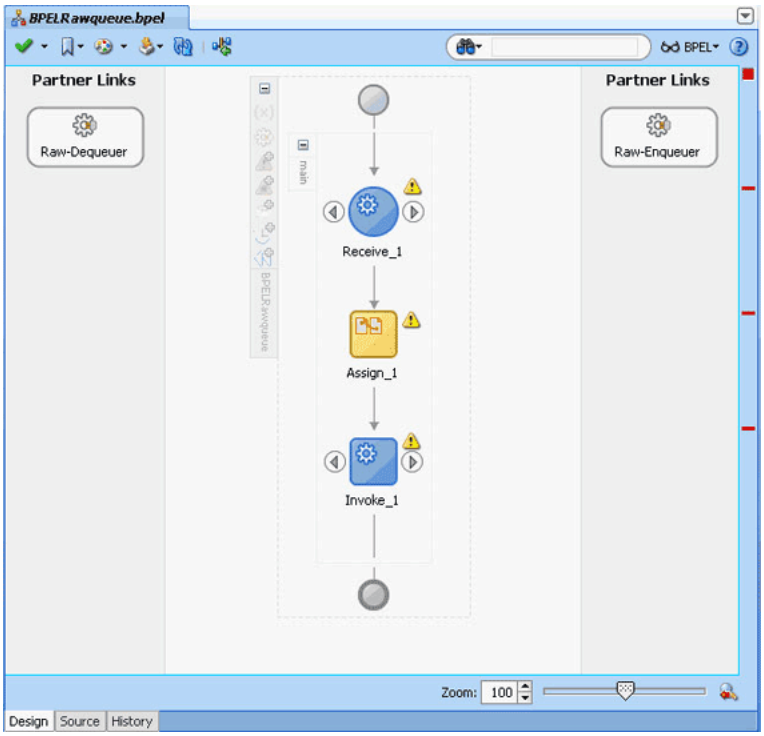


3. Click **File**, **Save All**.
4. Double-click **BPELRawqueue**.

   The BPELRawqueue.bpel page is displayed.
5. Drag and drop the **Receive**, **Assign**, and **Invoke** activities in the order mentioned, from the Component Palette to the Components area.

   The JDeveloper `BPELRawqueue.bpel` page is displayed, as shown in Figure 7–59.

Figure 7–59 The BPELRawqueue.bpel Page



6. Double-click the **Receive** activity.

   The Receive dialog is displayed.

7. Click the **Browse Partner Links** icon at the end of the Partner Link field.

   The Partner Link Chooser dialog is displayed.

8. Select **Raw-Dequeuer**, and then click **OK**.

   The Receive dialog is displayed with the Partner Link field populated with the value Raw-Dequeuer.

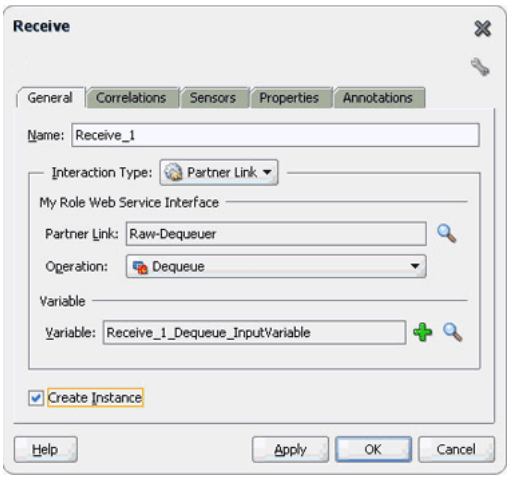
9. Click the **Auto-Create Variable** icon that is displayed at the end of the Variable field.

   The Create Variable dialog is displayed.

10. Accept the default values, and click **OK**.

11. Check the **Create Instance** box, as shown in Figure 7–60, and click **OK**.

Figure 7–60 The Receive Dialog



12. Double-click the **Invoke** activity.

    The Invoke dialog is displayed.

13. Click the **Browse Partner Links** icon at the end of the Partner Link field.

    The Partner Link Chooser dialog is displayed.

14. Select **Raw-Enqueuer**, and then click **OK**.

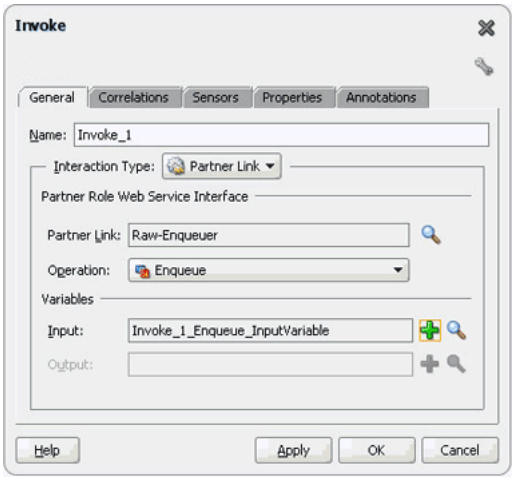
    The Invoke dialog is displayed with the Partner Link field populated with the value Raw-Enqueuer.

15. Click the **Automatically Create Input Variable** icon that is displayed at the end of the Input Variable field.

16. Accept the default values, and click **OK**.

    The Invoke dialog is displayed, as shown in Figure 7–61.

Figure 7–61 The Invoke Dialog



17. Click **OK**.

18. Double-click the **Assign** activity.

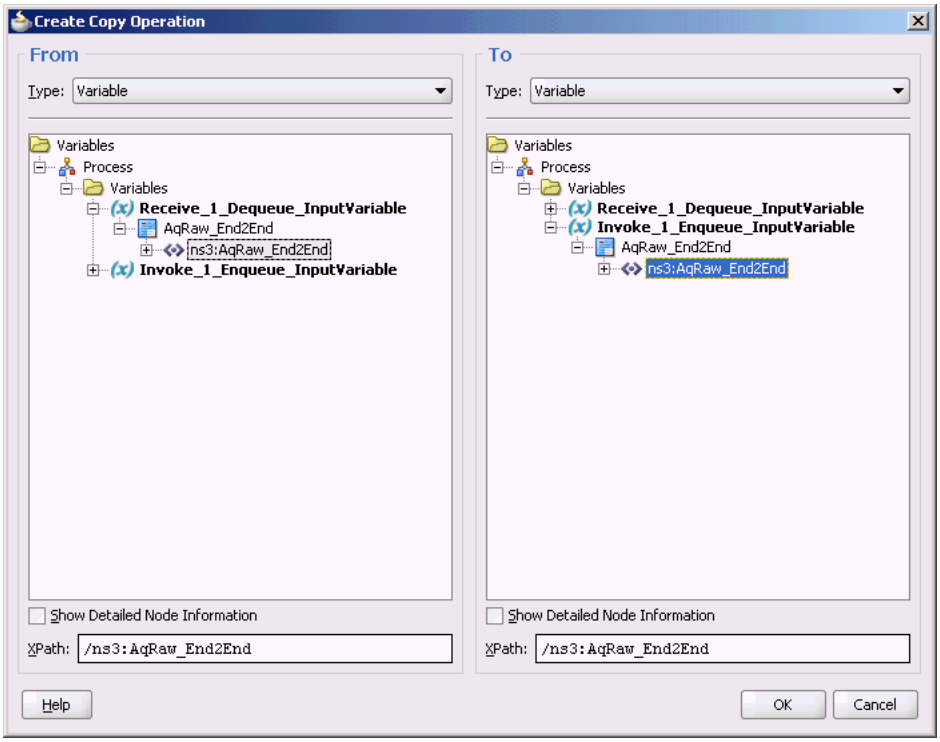
    The Assign dialog is displayed.

19. Click the plus icon, and select **Copy Operation**.

    The Create Copy Operation dialog is displayed.

20. Select the variables, as shown in Figure 7–62, and click **OK**.

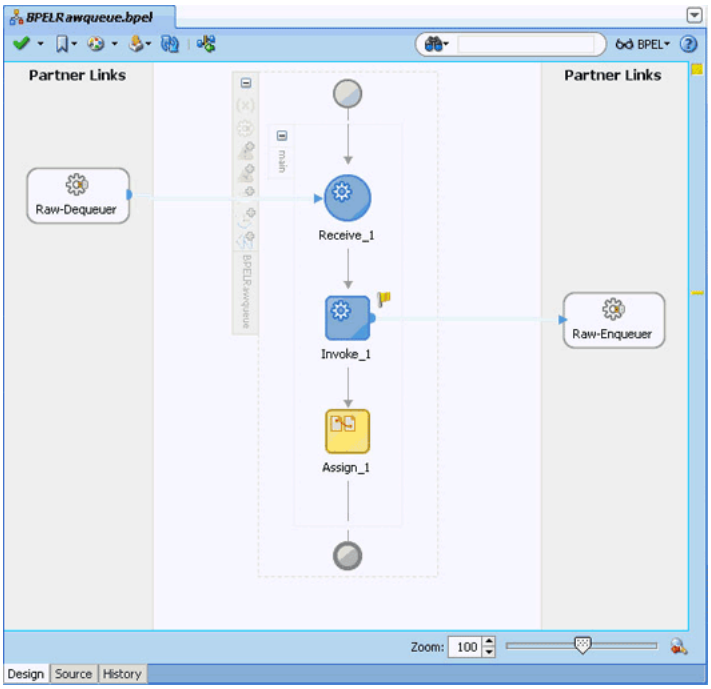
Figure 7–62 The Create Copy Operation Dialog



21. Click **OK** in the Assign dialog.

    The JDeveloper `BPELRawqueue.bpel` page is displayed, as shown in Figure 7–63.

Figure 7–63 The BPELRawqueue.bpel Page



22. Click **File**, **Save All**.

7.4.3.6 Configuring the Data Sources in the Oracle WebLogic Server Administration Console

1. Navigate to `http://servername:portnumber/console`.
2. Use the required credentials to open the Home page of the Oracle WebLogic Server Administration Console.
3. In the Home page, under Domain Structure, select **Services**, **JBDC**, and then click **DataSources**.

   The Summary of JDBC Data Sources page is displayed.
4. Click **New**. The Create a New JDBC Data Source page is displayed.
5. Enter the values for the properties to be used to identify your new JDBC data source.
6. Click **Next**. The Create a New JDBC Data Source Transaction Options page is displayed.
7. Click **Next**. The Create a New JDBC Data Source Connection Properties page is displayed.
8. Enter the connection properties in the Connection Properties page.
9. Click **Next**. The Create a New JDBC Data Source Test Database Connection page is displayed.
10. Click **Test Configuration** to test the database availability and the connection properties you provided. A message stating that the connection test succeeded is displayed at the top of the Create a New JDBC Data Source Test Database Connection page.
11. Click **Next**. The Create a New JDBC Data Source Select Targets page is displayed.

12. Select a target, and then click **Finish**.

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

13. Close the Oracle WebLogic Server Administration Console.

7.4.3.7 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps.

The following are the steps to deploy the application profile using JDeveloper:

1. Create an application server connection by using the procedure described in Chapter 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application by using the procedure described in Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

7.4.3.8 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed composite by using the Fusion Middleware Control Console. Perform the following steps:

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

   The composite you deployed is displayed in the Application Navigator.

2. In the Last 5 Instances pane, there is an entry of a new instance.

   This is the instance that triggered when you enqueued a message.

3. Click an instance.

   The Flow Trace page is displayed.

4. Click the **BPELRawqueue** component instance.

   The Audit page is displayed.

5. Click the **Flow-Debug** tab to debug the instance.

8

Oracle JCA Adapter for JMS

This chapter describes how to use the Oracle JCA Adapter for JMS (Oracle JMS Adapter), which enables an Oracle BPEL process or an Oracle Mediator component to interact with Java Messaging Service.

This chapter includes the following topics:

- Section 8.1, "Introduction to the Oracle JMS Adapter"
- Section 8.2, "Oracle JMS Adapter Features"
- Section 8.3, "Oracle JMS Adapter Concepts"
- Section 8.4, "Oracle JMS Adapter Use Cases"

8.1 Introduction to the Oracle JMS Adapter

The JMS architecture uses one client interface to many messaging servers. The JMS model has two messaging domains, point-to-point and publish-subscribe. In the point-to-point domain, messages are exchanged through a queue and each message is delivered to only one receiver. In the publish-subscribe model, messages are sent to a topic and can be read by many subscribed clients.

You can obtain JMS adapter sample files by accessing the Oracle SOA Sample Code site.

This section includes the following topics:

- Section 8.1.1, "Oracle JMS Adapter Integration with Oracle BPEL Process Manager"
- Section 8.1.2, "Oracle JMS Adapter Integration with Oracle Mediator"

8.1.1 Oracle JMS Adapter Integration with Oracle BPEL Process Manager

The JCA Binding Component is used for the bidirectional integration of the JCA 1.5 resource adapters with BPEL Process Manager. The JCA Binding Component is based on standards and employs the Web service Invocation Framework (WSIF) technology for exposing the underlying JCA interactions as Web services.

For information on Oracle JMS Adapter architecture, adapter integration with Oracle BPEL Process Manager (Oracle BPEL PM), and adapter deployments, see Chapter 3, "Adapter Integration with Oracle Application Server Components."

8.1.2 Oracle JMS Adapter Integration with Oracle Mediator

Mediator supports Oracle JCA Adapters and enables you to define inbound and outbound adapter services for each. An inbound adapter service receives data from an

external messaging system and transforms it into an XML message. An outbound adapter service sends data to a target application by transforming an XML message into the native format of the given adapter.

In the case of Oracle JMS Adapter service, by using Mediator, you can send or receive messages from a JMS queue or topic.

Oracle BPEL PM pre-dates Mediator, and most of this guide and the samples implicitly assume use with Oracle BPEL PM. However, the adapters work equally well with either Oracle BPEL PM or Mediator. For any mention of Oracle BPEL PM in this chapter, you may substitute Mediator, instead.

8.2 Oracle JMS Adapter Features

The Oracle JMS Adapter includes the following features:

- **Is based on JMS version 1.0.2b**
- **Is a generic Oracle JMS Adapter**

  Works with any JMS provider. It has been certified against AQ JMS (JMS providers OJMS 8.1.7, 9.0.1.4, and 9.2), TIBCO JMS, IBM Websphere MQSeries (IBM MQSeries JMS 6.0), Weblogic JMS, Apache, and Active MQ.
- **Supports JMS topics and queues**
- **Supports byte, text, and map message types**.

  Supports these data types only for this release. The Adapter Configuration Wizard provides the Native Format Builder wizard for consuming native data payloads at run time. The Native Format Builder wizard creates XSD definitions for the underlying native data.
- **Supports JMS headers and properties**
- **Supports jca.message.encoding property**

  The Oracle JMS Adapter supports the `jca.message.encoding` property for inbound and outbound payloads. If the `jca.message.encoding` property is used along with the `adapter.jms.encoding` property and the `nxsd:encoding` attribute, then the `jca.message.encoding` property takes precedence over the `adapter.jms.encoding` property, and the `nxsd:encoding` attribute is given the last preference. The `nxsd:encoding` value can be UTF, which-8 is typically recommended for interoperability and Unicode support. However, you can specify any legal encoding supported by the Java run-time environment. For a complete listing of supported encodings, visit `http://www.oracle.com/technetwork/java/index.html`. You can specify the encoding in the (N)XSD associated with the adapter proxy meta data. For example, you can specify the following attribute, `nxsd:encoding="iso-8859-1`

  The `jca.message.encoding` property is supported as an endpoint yu76y76 defined in `composite.xml` You can define this property using the Properties tab of the Adapter Configuration Wizard or by editing the `composite.xml` file. The `jca.message.encoding` property can be passed as a normalized message property for both inbound and outbound interactions.

  The following code snippet is an example of setting values in the composite.xml file for message encoding for an inbound service:

```
<service name="jms_inbound" ui:wsdlLocation="jms_inbound.wsdl">
   <interface.wsdl
```

```
interface="http://xmlns.oracle.com/pcbpel/adapter/jms/utf8/jcamessageencoding/
     jms_inbound#wsdl.interface(Consume_Message_ptt)" />
    <binding.jca config="jms_inbound_jms.jca">
    <property name="jca.message.encoding" type="xs:string" many="false"
                override="may">GBK</property>
  </binding.jca>
</service>
```

The following code snippet is an example of setting values in the composite.xml file for message encoding for an outbound reference:

```
<reference name="jms_outbound" ui:wsdlLocation="jms_outbound.wsdl">
    <interface.wsdl

interface="http://xmlns.oracle.com/pcbpel/adapter/jms/utf8/jcamessageencoding/
     jms_outbound#wsdl.interface(Produce_Message_ptt)"/>
    <binding.jca config="jms_outbound_jms.jca">
    <property name="jca.message.encoding" type="xs:string" many="false"
                override="may">GBK</property>
  </binding.jca>
</reference>
```

- **Supports the JMS message selector**

  Supports the JMS message selector for performing filtering while subscribing to JMS topics and queues. This parameter is based on the SQL 92 language for filtering messages based on fields present in the JMS header and properties section.

- **Is DOM2 compliant**

  The Oracle JMS Adapter can process and generate document objects that are compliant with DOM2 specification.

- **Supports normalized message.**

  Header manipulation and propagation is a key business integration messaging requirement. Oracle BPEL PM, Mediator, Oracle JCA, and Oracle B2B rely extensively on header support to solve customers' integration needs. For example, a user can preserve a file name from the source directory to the target directory by propagating it through message headers. Another example: the outbound Oracle JMS Adapter facilitates asynchronous request/response by propagating the `correlationId` and the `JMSReplyTo` address as JMS headers. In Oracle BPEL PM and Mediator, users can access, manipulate, and set headers with varying degrees of UI support.

  For more information, see Section 2.14, "Correlation Support Within Adapters".

  Propagating Headers in a Normalized Message:

  Normalized Message is simplified to have only two parts, properties and payload.Typically, properties are name-value pairs of scalar types. To fit the existing complex headers into properties, they are flattened into scalar types.

  Manipulating Headers in Design-Time:

  The user experience while manipulating headers in design time is simplified, because the complex properties are predetermined. In Mediator or Oracle BPEL designer, you can manipulate the headers with some reserved key words. For example, in Mediator designer, you can access an inbound Oracle File Adapter, `fileName` header by using the following expression:

```
$nmproperty.InboundFileHeaderType.fileName
```

However, this method does not address the properties that are dynamically generated based on your input. For example, in the Oracle AQ Adapter Wizard, you are allowed to propagate some of the fields from an AQ object as headers. Based on this user choice, the header definitions are generated. These definitions are not predetermined and hence cannot be accounted for in the list of predetermined property definitions. The user cannot design header manipulation of the dynamic properties before they are defined. To address this limitation, you must generate all the necessary services (composite entry points) and references. This restriction applies only to those services that are expected to generate dynamic properties. Once dynamic properties are generated, they must be captured in some given location for each composite. Only then can the user manipulate the dynamic properties in the Oracle Mediator or Oracle BPEL designer.

- **Supports specifying a durable JMS subscriber**
- **Supports persistent and nonpersistent modes of a JMS publisher**
- **Does not support connection retry functionality for MQ provider**
- **Does not support outbound retry functionality for AQJMS on Solaris**

  Note: When you use the Oracle JMS Adapter to connect to an AQ-JMS provider, and if the database that hosts the AQ destination is 10.1.0.4, then the adapter retry mechanism on the outbound direction fails to reconnect to the database server if the database server goes down. This is because of a client JDBC issue with `ojdbc14.jar`. To resolve this you must download the 10.1.0.4 JDBC drivers and use them in the mid tier by replacing the libraries, specifically ojdbc14.jar in `$MIDTIER_ORACLE_HOME/jdbc`. For a detailed explanation about how to resolve this issue, refer to Metalink Note 317385.1.

- The JMS API specifies three types of acknowledgments that can be sent by the JMS publisher:
  - `DUPS_OK_ACKNOWLEDGE`, for consumers that are not concerned about duplicate messages
  - `AUTO_ACKNOWLEDGE`, in which the session automatically acknowledges the receipt of a message
  - `CLIENT_ACKNOWLEDGE`, in which the client acknowledges the message by calling the message's acknowledge method
- **Supports tracking message size**

  The Oracle JMS Adapter is message size aware. The Oracle JMS Adapter calculates the message size and reports the size back to the JCA Binding Component. The API, related to size, exposed by the JCA Binding Component can be used for reporting purposes.
- **Supports MapMessage Data Type**

  A `MapMessage` is used to send a set of name-value pairs where names are strings and values are Java primitive types. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined. It inherits from a message and adds a map message body.

Oracle JMS adapter provides support for processing `MapMessage`. It now supports one new ActivationSpec and InteractionSpec property each namely `JmsMapMessageConsumeActivationSpec` and `JmsMapMessageProduceInteractionSpec`.

The `PayloadEntry` property specifies that the `MapMessage` entry is used as the payload. Users have the option to send payload as an attachment if the `AttachmentList` property is defined.

All other `MapMessage` entries are converted to adapter properties identified by `jca.jms.Map.xxxx`, where `xxxx` is name of the `MapMessage` entry.

If both `PayloadEntry` and `AttachmentList` properties are not defined, then the entire `MapMessage` is converted to XML and the XML file is transferred as the payload.

- **Supports Enterprise Information System (EIS) Credentials**

  The Oracle JMS Adapter supports securing of the Enterprise Information System (EIS) credentials such as the user name and password, whenever it establishes an outbound connection with EIS. You can secure the user name and password for Oracle JMS Adapter by using Oracle WebLogic Server container-managed sign-on.

  For more information about support for securing of the Enterprise Information System (EIS) credentials, see Section 4.2.22, "Securing Enterprise Information System Credentials."

- **Supports Streaming Large Payload**

  Oracle JMS Adapter provides support to stream payload. When you enable this feature, the payload is streamed to a database instead of getting manipulated in the SOA run time as in a memory DOM. This feature can be used while handling large payloads. To enable support to stream payload, ensure that you select the **Enable Streaming** check box while defining the consume operation parameters on the Consume Operation Parameters page in Oracle JDeveloper (JDeveloper). When the **Enable Streaming** check box is selected, a corresponding Boolean property `EnableStreaming` is appended to the `ActivationSpec` properties defined in the respective `.jca` file, as shown in the following example. If the `EnableStreaming` property does not exist, then the default value of false is assumed.

  ```
  <activation-spec
  className="oracle.tip.adapter.jms.inbound.JmsConsumeActivationSpec">
    <property name="DestinationName" value="jms/DemoInQueue"/>
    <property name="UseMessageListener" value="false"/>
    <property name="PayloadType" value="TextMessage"/>
    <property name="EnableStreaming" value="true"/>
  </activation-spec>
  ```

- **Supports Transactions**

  A transaction enables an application to coordinate a group of messages for production and consumption, treating messages sent or received as a single unit. When an application commits a transaction, all messages it received within the transaction are removed by the JMS provider. The messages it sent within the transaction are delivered as one unit to all JMS consumers. If the application rolls back the transaction, then the messages it received within the transaction are returned to the messaging system and the messages it sent are discarded. The Oracle JMS Adapter supports JMS transactions. A JMS-transacted session supports transactions that are located within the session. A JMS-transacted session's transaction does not have any effects outside of the session.

- **Supports Error Handling**

  For information about error handling, refer to Section 2.21, "Error Handling."

- **Supports Multiple Consumer Threads**

  The Oracle JMS Adapter supports an activation endpoint property, `"adapter.jms.receive.threads"`. Setting this property in composite.xml is a preferred way to spawn multiple poller threads for the inbound message flow between the adapter and the Enterprise Information System (EIS). This helps improve performance because multiple poller threads dequeue messages in a round robin fashion; this assists in Distributed scenarios as well.

- **Supports Performance Tuning**

  The Oracle JMS Adapter supports performance tuning.

  For more information, see "Oracle SOA JMS Adapter Tuning" in the Oracle Fusion Middleware Performance and Tuning Guide.

Note: Oracle JMS Adapter cannot be used programmatically inside an EJB or JMS client.

8.3 Oracle JMS Adapter Concepts

Messaging is any mechanism that enables communication between programs. Messages are structured data that one application sends to another. Message-oriented middleware (MOM) is an infrastructure that supports scalable enterprise messaging. MOM ensures fast, and reliable asynchronous communication, guaranteed message delivery, receipt notification, and transaction control. JMS is a Java interface developed by Sun Microsystems for producing, sending, and receiving messages of an enterprise messaging system. JMS is an API that JMS vendors implement. Oracle provides two implementations of JMS, WLS JMS and Oracle JMS based on Oracle advanced queues. A JMS producer creates JMS messages and a JMS consumer consumes JMS messages.

JMS supports two messaging paradigms, point-to-point (queues) and publish/subscribe (topics).

This section includes the following topics:

- Section 8.3.1, "Point-to-Point"
- Section 8.3.2, "Publish/Subscribe"
- Section 8.3.3, "Destination, Connection, Connection Factory, and Session"
- Section 8.3.4, "Structure of a JMS Message"
- Section 8.3.5, "Oracle JMS Adapter Header Properties"

8.3.1 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.

8.3.2 Publish/Subscribe

In publish/subscribe messaging, producers publish messages to a topic, and the consumer subscribes to a particular topic. Many publishers can publish to the same

topic, and many consumers can subscribe to the same topic. All messages published to the topic by the producers are received by all consumers subscribed to the topic. By default, subscribers receive messages only when the subscribers are active. However, JMS API supports durable subscriptions that ensure that consumers receive messages that were published even when the subscribers are not up and running. The durable subscription involves registering the consumer with a unique ID for retrieving messages that were sent when the consumer was inactive. These messages are persisted by the JMS provider and are either sent to the consumer when it becomes active again or purged from storage if the message expires. The JMS producer can be set either to persistent or nonpersistent mode. The messages are not persisted in the latter case and can be used only for nondurable subscriptions.

For scenarios that requires you to work with durable subscriptions on Oracle WebLogic Server, a connector factory with `ClientID` property defined is required, as shown in the following example:

```
<FactoryProperties>ClientID=uniquename</FactoryProperties>
```

When defining multiple durable subscriber it would entail you to define multiple connector factory each with a unique `ClientID` property specified. You must take care to not use the same connector factory for any other adapter interaction (such as outbound interaction if it is used for processing inbound messages) because Oracle WebLogic Server allows a `clientid` to be bound only once. For a scenario in which a connector factory with `ClientId` defined is used on the inbound to process incoming messages a different connector factory should be used for the outbound adapter interactions.

Note: You must manually remove durable subscribers that are not used by the BPEL partner link. Oracle JMS Adapter does not automatically remove these durable subscriptions.

The JMS API supports both synchronous and asynchronous communication for message consumption. In the synchronous case, the consumer explicitly calls the `receive()` method on the topic or queue. In the asynchronous case, the JMS client registers a message listener for the topic or queue and the message is delivered by calling the listener's `onMessage()` method.

8.3.3 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.

8.3.4 Structure of a JMS Message

The JMS message has a mandatory standard header element, an optional properties element, and an optional standard payload element. The payload can be a text message, byte message, map message, stream message, or object message. The properties element is JMS provider-specific and varies from one JMS provider to another.

8.3.5 Oracle JMS Adapter Header Properties

For information about the Oracle JMS Adapter header properties, see Appendix A.4, "Oracle JMS Adapter Properties."

8.4 Oracle JMS Adapter Use Cases

This section includes the following topics:

- Section 8.4.1, "Configuring Oracle JMS Adapter"
- Section 8.4.2, "Configuring Oracle JMS Adapter with TIBCO JMS"
- Section 8.4.3, "Configuring Oracle JMS Adapter with IBM WebSphere MQ JMS"
- Section 8.4.4, "WLS JMS Text Message"
- Section 8.4.5, "Accessing Queues and Topics from WLS JMS Server in a Remote Oracle WebLogic Server Domain"
- Section 8.4.6, "Synchronous/Asynchronous Request Reply Interaction Pattern"
- Section 8.4.7, "AQ JMS Text Message"
- Section 8.4.8, "Accessing Queues and Topics Created in 11g from the OC4J 10.1.3.4 Server"

8.4.1 Configuring Oracle JMS Adapter

The following use case demonstrates the procedure for configuring Oracle JMS Adapter and examines the resulting WSDL files and associated `weblogic-ra.xml` files.

This section includes the following topics:

- Section 8.4.1.1, "Creating an Application and a SOA Project"
- Section 8.4.1.2, "Using the Adapter Configuration Wizard to Configure Oracle JMS Adapter"
- Section 8.4.1.3, "Generated Files"
- Section 8.4.1.4, "weblogic-ra.xml file"
- Section 8.4.1.5, "Produce Message Procedure"

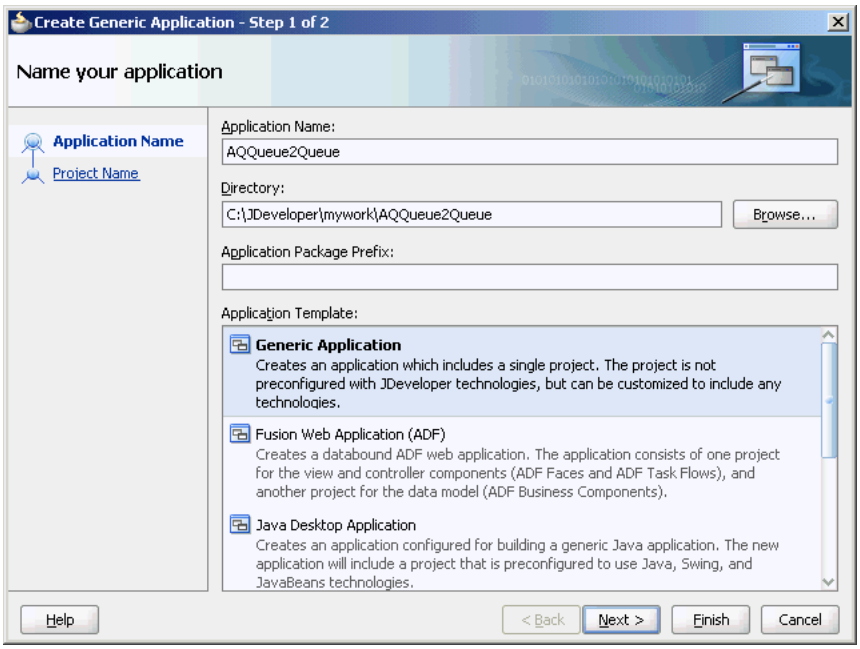
8.4.1.1 Creating an Application and a SOA Project

You must first create an JDeveloper application to contain the SOA composite. Use the following steps to create a new application and a SOA project:

1. Open JDeveloper.
2. In the Application Navigator, click **New Application**.

   The Create Generic Application - Name your Application page is displayed, as shown in Figure 8–1.
3. Enter a name for the application in the **Application Name** field. For example, `AQQueue2Queue`.
4. In the Application Template list, choose **Generic Application**.

Figure 8–1 The Create Generic Application - Name your application Page



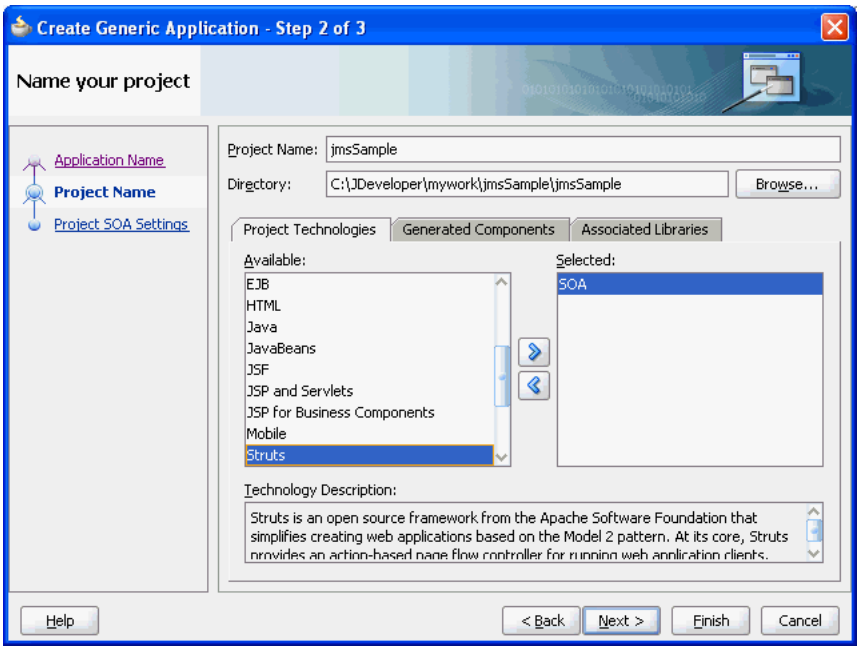
5. Click **Next**.

   The Name your project dialog is displayed, as shown in Figure 8–2.

6. In the **Project Name** field, enter a descriptive name. For example, `AQQueue2Queue`.

7. In the Available list in the **Project Technologies** tab, double-click **SOA** to move it to the Selected list.

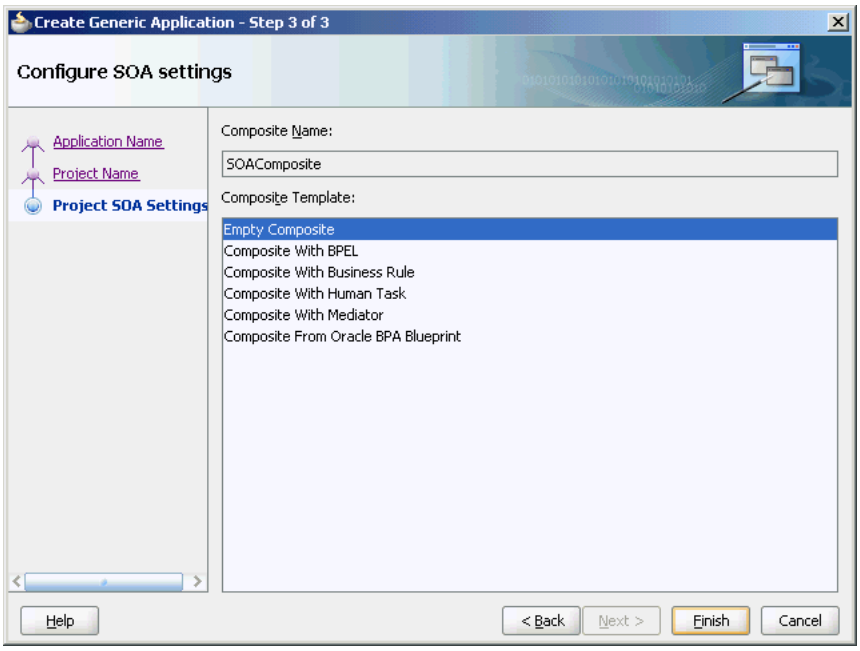
Figure 8–2 The Create Generic Application - Name your Generic project Page



8. Click **Next**.

The Create Generic Application - Configure SOA settings page is displayed, as shown in Figure 8–3.

Figure 8–3 The Create Generic Application - Configure SOA Settings Page

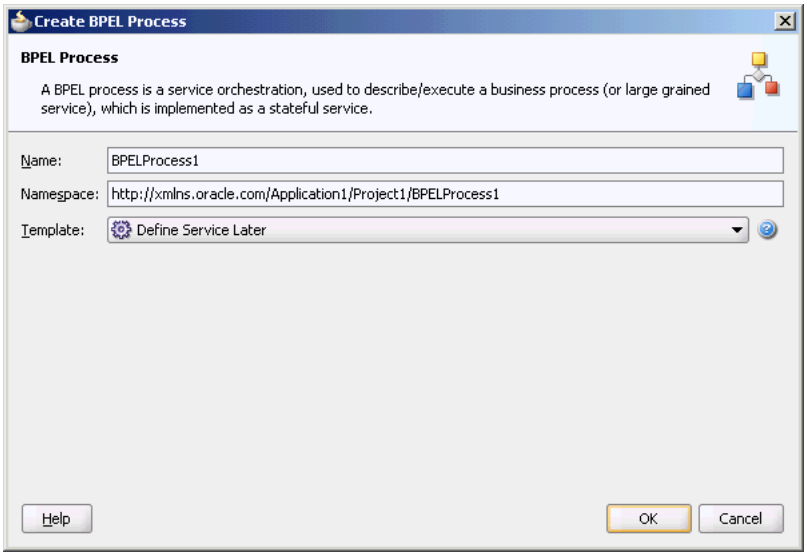


9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

   You have created a new application, and an SOA project. This automatically creates an SOA composite.

   The Create BPEL Process page is displayed, as shown in Figure 8–4.

Figure 8–4 The Create BPEL Process Page



10. Enter a name for the BPEL process in the **Name** field. In this example, use the default name.

11. Select **Define Service Later** in the Template list, and then click **OK**.

    You have created a BPEL process.

8.4.1.2 Using the Adapter Configuration Wizard to Configure Oracle JMS Adapter

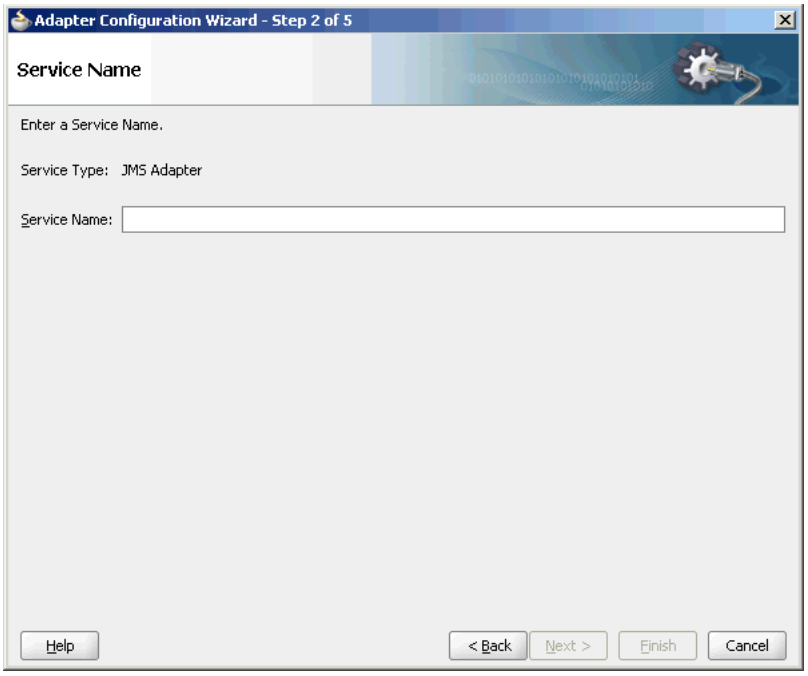
The following are the steps to configure an Oracle JMS Adapter by using the Adapter Configuration Wizard:

1. In the Component Palette, select **SOA**.
2. Drag and drop **JMS Adapter** from the Service Adapters list to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard is displayed.
3. Click **Next**.

   The Adapter Configuration Wizard - Service Name page is displayed, as shown in Figure 8–5.

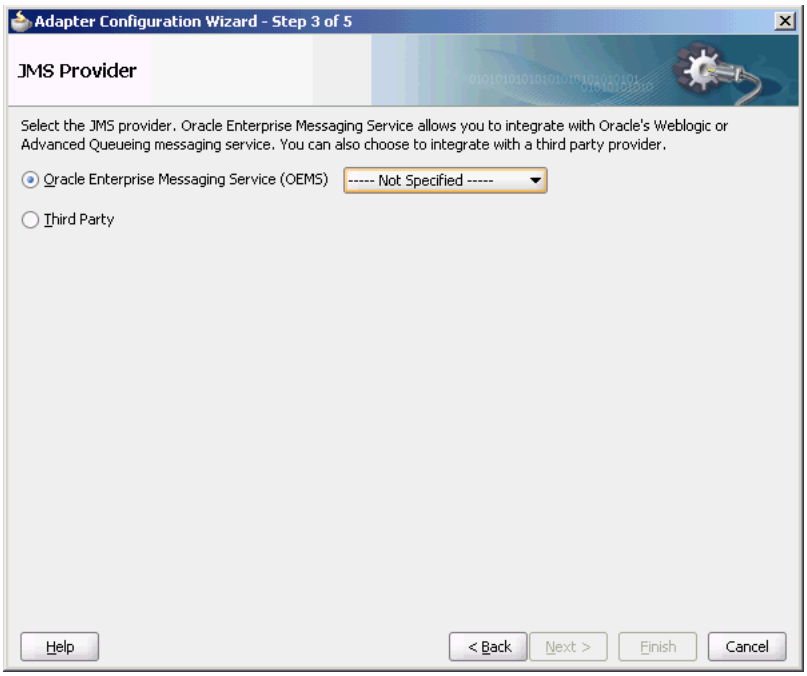
Figure 8–5 Service Name Page



4. Enter a name for the service, and then click **Next**.

   The Adapter Configuration Wizard - JMS Provider page is displayed, as shown in Figure 8–6.

Figure 8–6 The Adapter Configuration Wizard - JMS Provider Page

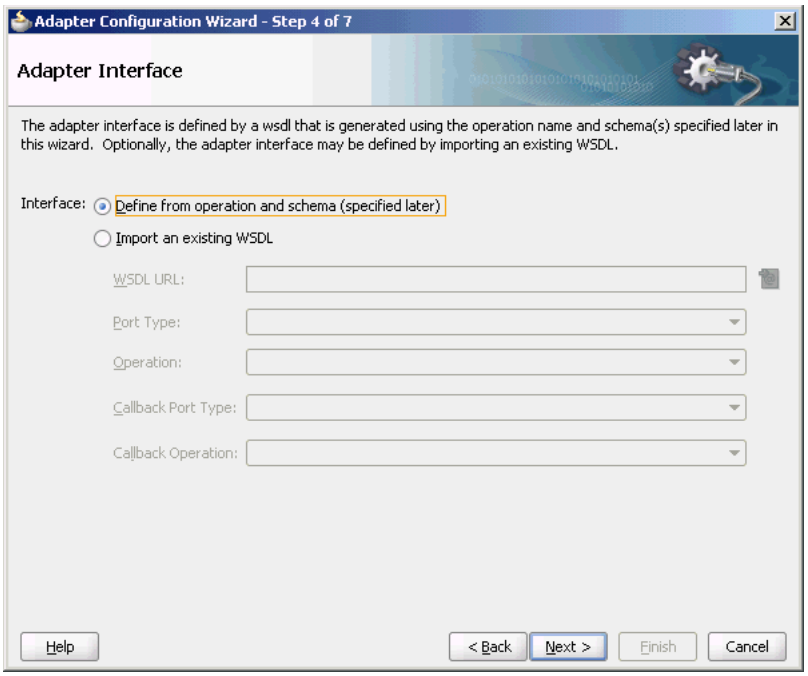


5. Select any one operation. In this example, select **Oracle Weblogic JMS**.
   - **Oracle Enterprise Messaging Service (OEMS)**: This enables you to integrate with the Weblogic service or Advanced Queueing messaging service.
   - **Third Party**: Select this option to integrate with a third party provider.
6. Click **Next**.

   The Adapter Configuration Wizard - Service Connection page is displayed.
7. You must establish connectivity between the design-time environment and the server you want to deploy it to.

   Perform the steps mentioned in Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters" to create an application server connection.
8. Click **Next**. The Adapter Interface page is displayed, as shown in Figure 8–7.
9. In the Adapter Interface page, select **Define from operation and schema (specified later)**.

Figure 8–7 The Adapter Configuration Wizard - Adapter Interface Page



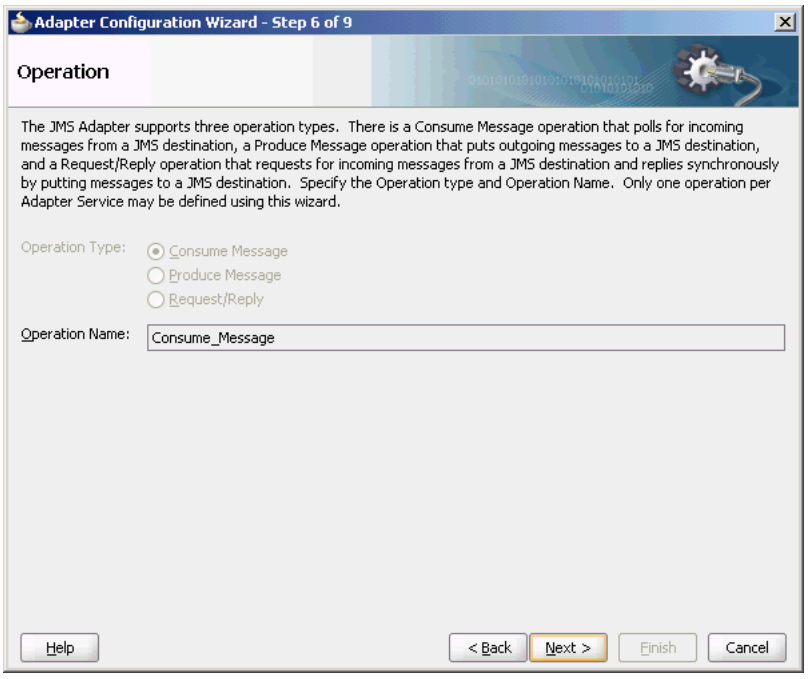
10. Click **Next**.

    The Adapter Configuration Wizard- Operation page is displayed.

11. Select **Consume Message**, **Produce Message**, or **Request/Reply**. In this example, select **Consume Message**.

    The operation name is filled in automatically, as shown in Figure 8–8.

Figure 8–8 The Adapter Configuration Wizard - Operation Page

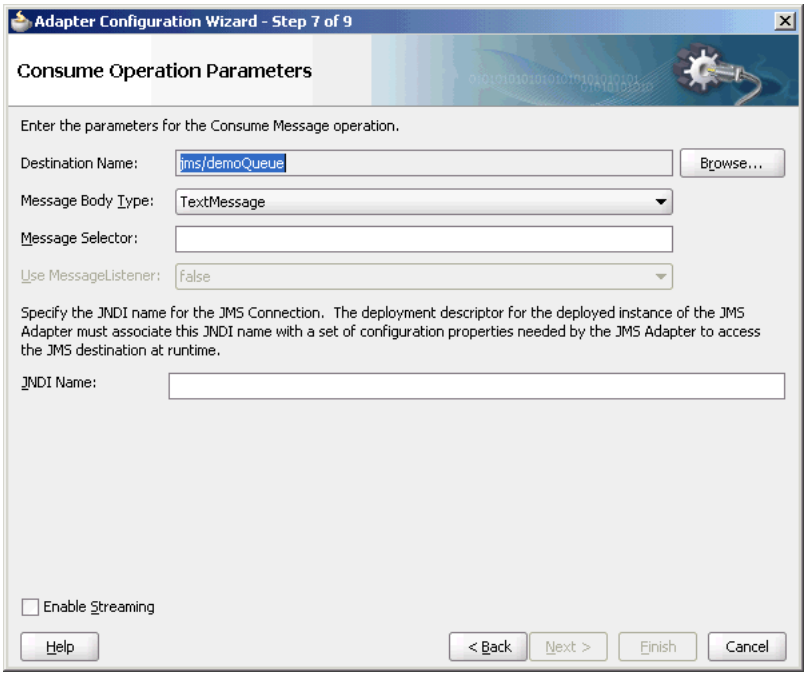


The **Consume Message** option enables the adapter to consume (receive) inbound messages from a JMS destination.

12. Click **Next**.

    The Adapter Configuration Wizard - Consume Operation Parameters page is displayed, as shown in Figure 8–9.

Figure 8–9 The Adapter Configuration Wizard - Consume Operation Parameters Page



13. Enter values for the following fields:

- **Destination Name**

  This is the JNDI name of the JMS queue or topic from which to receive the message. This is not an editable field. You must click **Browse** to browse for the queue or topic. The queue or topic to be chosen is based on the type of JMS provider you are using.

  For more information, see the following sections:

  – Section 8.4.2, "Configuring Oracle JMS Adapter with TIBCO JMS"

  – Section 8.4.3, "Configuring Oracle JMS Adapter with IBM WebSphere MQ JMS"

- **Message Body Type**

  The supported values are `TextMessage`, `BytesMessage`, `MapMessage`. The `StreamMessage` message type is not supported in this release.

- **Durable Subscriber ID**

  This field is optional. If you are setting up a durable subscriber, then the durable subscriber ID is required. Generally, a subscriber loses messages if the subscriber becomes disconnected, but a durable subscriber downloads stored messages when it reconnects.

Note: When the JMS provider is Oracle Weblogic JMS or Oracle Advanced queueing messaging service, then the Durable Subscriber option appears only when a topic is selected. However, the Durable Subscriber option always appears when the JMS provider is a third party.

- **Message Selector**

  This field is also optional. It filters messages based on header and property information. The message selector rule is a Boolean expression. If the expression is `true`, then the message is consumed. If the expression is `false`, then the message is rejected.

  For example, you can enter logic, such as:

  - **JMSPriority > 3**. Based on this, messages with a priority greater than 3 are consumed; all other messages are rejected.
  - **JMSType = 'car' AND color = 'blue' AND weight > 2500**
  - Country in ('UK', 'US', 'France')

- **Use MessageListener**

  This field is always set to `False` by default.

- **JNDI Name**

  The value specified in the JNDI name should exist in the Oracle JMS Adapter weblogic-ra.xml file to ensure that the adapter runs in managed mode.

Note: This example shows a consume message operation. For a produce message operation, this page is different. See Section 8.4.1.5, "Produce Message Procedure" to see how this part of the procedure differs.

After you enter the appropriate parameters, click **Next**.

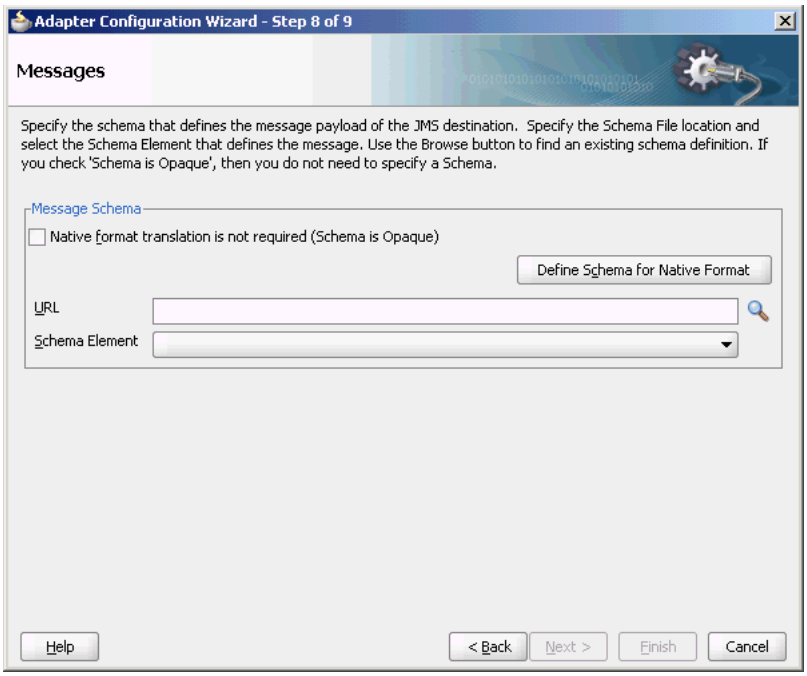
14. The Adapter Configuration Wizard - Messages page is displayed, as shown in Figure 8–10. The settings in this page define the correct schema for the message payload.

    You can perform one of the following:

    - Check **Native format translation is not required (Schema is Opaque)**, which disables the rest of the fields.
    - Click **Define Schema for Native Format** to start the Native Format Builder wizard, which guides you through the process of defining the native format.
    - Enter the path for the schema file URL (or browse for the path).

    The following steps demonstrate the last option: browsing for the schema file URL.

Figure 8–10 The Adapter Configuration Wizard - Messages Page

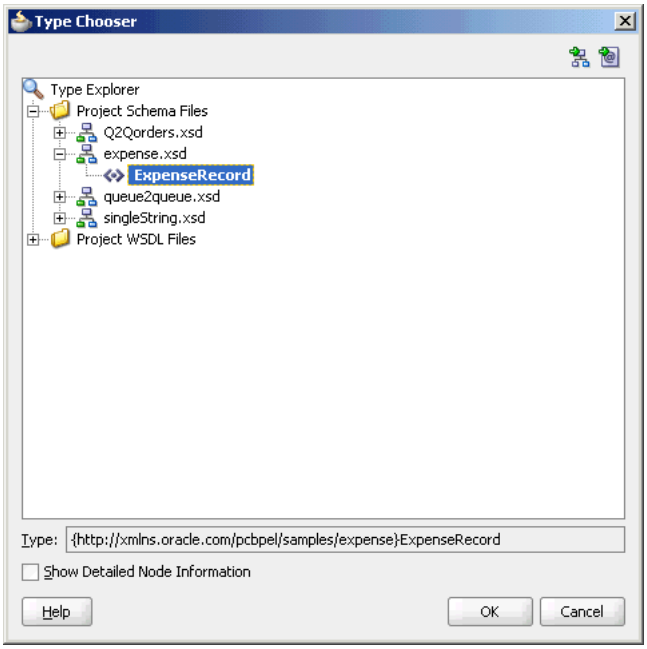


15. Click the **Browse** button.

    The **Type Chooser** dialog is displayed, with the **Type Explorer** navigation tree, as shown in Figure 8–11.

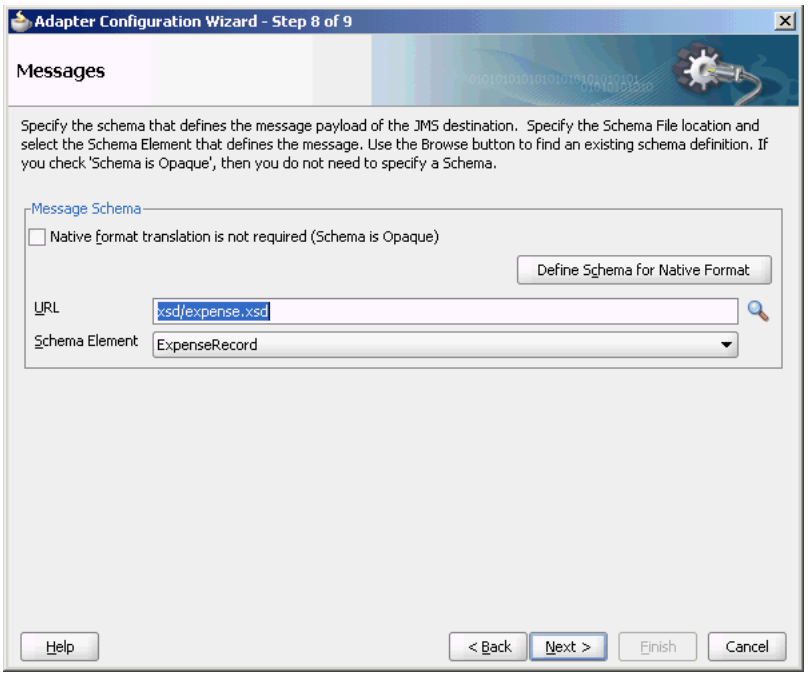
16. Browse the tree and select the appropriate schema type, and then click **OK**.

Figure 8–11 Selecting a Schema from the Type Chooser Dialog



The Messages page is displayed again, this time with the **Schema File URL** field and the **Schema Element** field filled up, as shown in Figure 8–12.

Figure 8–12 Completed Messages Dialog



17. Click **Next**.

    The **Finish** page is displayed. This box shows the path and name of the adapter file that the wizard creates.

18. Click **Finish**.

    The composite.xml page is displayed.

8.4.1.3 Generated Files

The following composite file is generated by the Adapter Configuration Wizard:

```
<composite name="AQQueue2Queue"           revision="1.0"
label="2007-09-04_11-58-50_914"         mode="active"          state="on"
xmlns="http://xmlns.oracle.com/sca/1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy">
  <import namespace="http://xmlns.oracle.com/pcbpel/adapter/jms/Inbound/"
location="Inbound.wsdl" importType="wsdl"/>
  <import namespace="http://xmlns.oracle.com/pcbpel/adapter/jms/Outbound/"
location="Outbound.wsdl" importType="wsdl"/>
  <service name="Inbound">
    <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/jms/Inbound/#wsdl.interface
(Consume_Message_ptt)"/>
    <binding.jca config="Inbound_jms.jca"/>
  </service>
  <component name="BPELProcess1">
    <implementation.bpel src="BPELProcess1.bpel"/>
  </component>
  <reference name="Outbound">
    <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/jms/Outbound/#wsdl.interface
(Produce_Message_ptt)"/>
```

```
    <binding.jca config="Outbound_jms.jca"/>
  </reference>
  <wire>
    <source.uri>Inbound</source.uri>
    <target.uri>BPELProcess1/Inbound</target.uri>
  </wire>
  <wire>
    <source.uri>BPELProcess1/Outbound</source.uri>
    <target.uri>Outbound</target.uri>
  </wire>
</composite>
```

The following code segment defines the name of the adapter and the locations of various necessary schemas and other definition files.

```
 <import namespace="http://xmlns.oracle.com/pcbpel/adapter/jms/Inbound/"
location="Inbound.wsdl" importType="wsdl"/>
  <import namespace="http://xmlns.oracle.com/pcbpel/adapter/jms/Outbound/"
location="Outbound.wsdl" importType="wsdl"/>
```

This code segment imports the necessary namespace.

```
<definitions name="Inbound"
targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/jms/Inbound/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/jms/Inbound/"
xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:imp1="http://xmlns.oracle.com/pcbpel/samples/expense">
    <types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://xmlns.oracle.com/pcbpel/samples/expense"
schemaLocation="xsd/expense.xsd"/>
        </schema>
    </types>
    <message name="ExpenseRecord_msg">
        <part name="ExpenseRecord" element="imp1:ExpenseRecord"/>
    </message>
    <portType name="Consume_Message_ptt">
        <operation name="Consume_Message">
            <input message="tns:ExpenseRecord_msg"/>
        </operation>
    </portType>
```

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

```
<adapter-config name="dequeue" adapter="Jms Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/wls/Queue" UIConnectionName="wls3"
UIJmsProvider="WLSJMS" adapterRef=""/>
  <endpoint-activation portType="Consume_Message_ptt" operation="Consume_
Message">
    <activation-spec
className="oracle.tip.adapter.jms.inbound.JmsConsumeActivationSpec">
      <property name="DestinationName" value="jms/DemoInQueue"/>
      <property name="UseMessageListener" value="false"/>
      <property name="PayloadType" value="TextMessage"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

8.4.1.4 weblogic-ra.xml file

The `weblogic-ra.xml` file defines the endpoints for the JMS connection factories. The connection factories include configuration properties for each endpoint. Endpoints are added to accommodate different types of connections, as demonstrated in the following sections. The following example is from the generic `weblogic-ra.xml` file:

```
<connection-instance>
  <jndi-name>eis/wls/Queue</jndi-name>
    <connection-properties>
       <properties>
           <property>
               <name>ConnectionFactoryLocation</name>
               <value>weblogic.jms.XAConnectionFactory</value>
           </property>
           <property>
               <name>FactoryProperties</name>
               <value></value>
           </property>
           <property>
               <name>AcknowledgeMode</name>
               <value>AUTO_ACKNOWLEDGE</value>
           </property>
           <property>
              <name>IsTopic</name>
              <value>false</value>
           </property>
           <property>
               <name>IsTransacted</name>
               <value>false</value>
           </property>
           <property>
               <name>Username</name>
               <value></value>
           </property>
           <property>
               <name>Password</name>
               <value></value>
           </property>
           </properties>
      </connection-properties>
</connection-instance>
```

8.4.1.4.1 Creating a New Connection by Using the Oracle WebLogic Server Administration Console

You can also create a new connection by using the Oracle WebLogic Server Administration Console. The following are the steps for creating a new connection by using the Oracle WebLogic Server Administration Console:

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

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

The Oracle WebLogic Server Administration Console - Summary of Deployments page is displayed.

4. Under **Deployments**, click any JMS adapter that you have deployed. For example, click **JmsAdapter**.

   The Oracle WebLogic Server Administration Console - Settings for JmsAdapter page is displayed.

5. Click the **Configuration** tab, and then click the **Outbound Connection Pools** tab.

   The Outbound Connection Pool Configuration Table is displayed.

6. Click **Next**.

   The Create a New Outbound Connection page is displayed.

7. Select the default outbound connection group, and then click **Next**.

8. Click **Next**.

9. In the **JNDI Name** field, enter the JNDI name that you want to use to obtain the new connection instance. For example, `eis/wls/Queue`.

   You can specify any name for the JNDI field. However, you must ensure that you use the same JNDI name while defining the consume or produce operation parameters in JDeveloper.

10. Click **Finish**.

    The Save Deployment Plan Assistant page is displayed.

    The configuration changes that you made must be stored in a new deployment plan.

11. In the **Path** field, select or enter the path of a deployment plan file. The path must end with '.xml'.

12. Click **OK**.

    You have created a new connection. After you have done this, you must verify whether the properties you have created are correct.

13. In the Settings for JmsAdapter page, click the **Configuration** tab, and then click the **Properties** tab.

    The connection that you created is listed in this page. Verify whether this value is correct. For example, if you are connecting to a third-party JMS server, then ensure that the Connection Factory Location field has the correct value applicable for a third-party JMS server.

    Note: In this example, you created a new connection for Oracle JMS Adapter by using the Oracle WebLogic Server Administration Console. To create connection for other adapters, you must follow the same steps. However, ensure that you select the appropriate adapter for which you want to create a connection in Step 4.

14. Click **Save**.

8.4.1.4.2 Adding a Third-Party JMS Provider

You can specify that the adapter uses a third-party JMS Provider for non-Web Logic Server JMS and non-AQJMS connection instances, by supplying a value to the

FactoryProperties parameter in the weblogic-ra.xml file. Specifically, you can provide the `ThirdPartyJMSProvider` value to the FactoryProperties parameter. This property is required only when you deploy an adapter to the WebLogic Server.

When this value is set to true, the JMS Adapter does not use `DestinationAvailabilityListener` for creating consumers for processing of JMS messages. The default is false. You must ensure you employ code similar to the following snippet:

```
<property>
    <name>FactoryProperties</name>
    <value>ThirdPartyJMSProvider=true</value>
</property>
```

> **Note:** All pre-populated connection instances on a WebLogic Server reflect the change and consequently, no further tuning is required for those instances. Only when a new non WLS JMS or AQJMS provider access is required do you must add new connection instance and therefore the ThirdPartyJMSProvider property.

8.4.1.5 Produce Message Procedure

A produce message operation has certain differences in the definition procedure, particularly in Step 13 of Section 8.4.1.2, "Using the Adapter Configuration Wizard to Configure Oracle 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 page is shown in Figure 8–13.

- **Destination Name:**

  The JNDI name of the JMS queue or topic to which the message must be delivered. The name to enter is based on the type of JMS provider you use.

  For more information about destination name, see the following:

  - Section 8.4.2, "Configuring Oracle JMS Adapter with TIBCO JMS"
  - Section 8.4.3, "Configuring Oracle JMS Adapter with IBM WebSphere MQ JMS"

- **Message Body Type:**

  The supported values are `TextMessage`, `BytesMessage`, and `MapMessage`. `StreamMessage` is not supported in this release.

- **Delivery Mode:**

  The values are `Persistent` or `Nonpersistent`. A persistent delivery mode specifies a persistent JMS publisher; that is, a publisher that stores messages for later use by a durable subscriber. A durable subscriber is a consume message with a durable subscriber ID in the corresponding field in Step 15 of Section 8.4.1.2, "Using the Adapter Configuration Wizard to Configure Oracle 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 time to receive all the messages.

- **Priority:**

Select a priority value, with `9` representing the highest priority and `0` representing the lowest priority. The default is `4`.

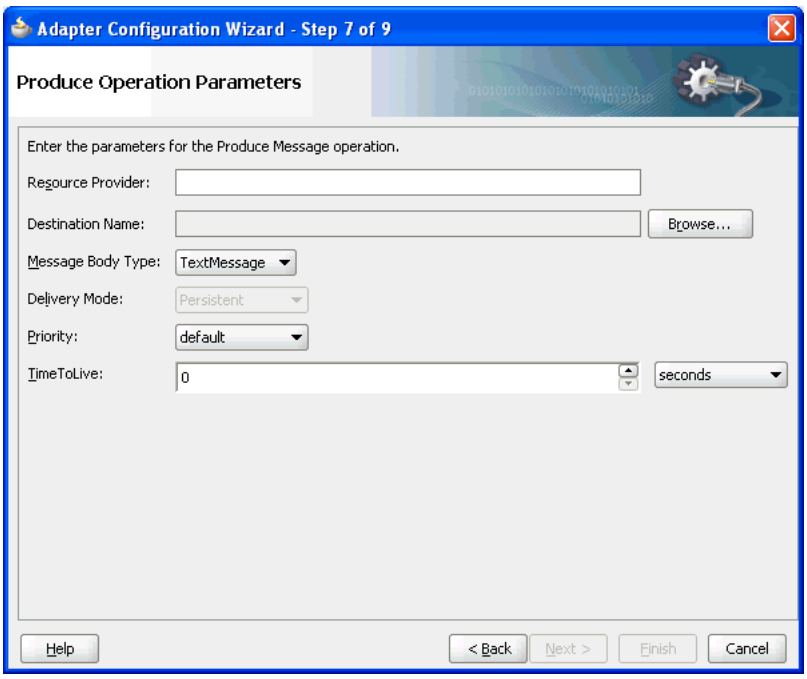
- **TimeToLive:**

  The amount of time before the message expires and is no longer available to be consumed.

- **Unit of order**

  Enables a message producer or group of message producers acting as one, to group messages into a single unit that is processed sequentially in the order the messages were created. The message processing of a single message is complete when a message is acknowldged, committed, recovered, or rolled back. Until message processing for a message is complete, the remaining unprocessed messages for that Unit of Order are blocked. This unit of order property enables you to set the unit-of-order value for the MessageProducer when producing a message.

Figure 8–13 Produce Operation Parameters Page



8.4.2 Configuring Oracle JMS Adapter with TIBCO JMS

This section describes how to configure Oracle JMS Adapter with Tibco JMS for direct connection and nondirect connection.

8.4.2.1 NonDirect Connection

Perform the following steps:

1. Copy the following file to the `<SOAInstall_DIR>/user_projects/domains/<DOMAIN_NAME>/lib` folder:

   - `/<YOUR-TIBCO-INSTALL-LOCATION>/clients/java/tibjms.jar`

2. Configure the connector factory by modifying the `weblogic-ra.xml` file in `AS11gR1SOA/soa/connectors/JmsAdapter.rar`, as shown in the following example:

```
<connection-instance>
        <jndi-name>eis/tibjms/Topic</jndi-name>
         <connection-properties>
                                                <properties>
                                                <property>
                                          <name>ConnectionFactoryLocation</name>
                                          <value>TopicConnectionFactory</value>
                                 </property>
                                 <property>
                                       <name>FactoryProperties</name>
                                       <value>java.naming.factory.initial=
com.tibco.tibjms.naming.TibjmsInitialContextFactory;java.naming.provider.url=ti
bjmsnaming://<HOST>:<PORT>;java.naming.security.principal=<USERNAME>;java.namin
g.security.credentials=;ThirdPartyJMSProvider=true</value>
                                 </property>
                                 <property>
       <name>AcknowledgeMode</name>
       <value>AUTO_ACKNOWLEDGE</value>
                                 </property>
                                 <property>
                                       <name>IsTopic</name>
                                       <value>true</value>
                                 </property>
                                 <property>
                                          <name>IsTransacted</name>
                                          <value>true</value>
                                 </property>
                                 <property>
                                          <name>Username</name>
                                          <value><USERNAME></value>
                                 </property>
                                 <property>
                                          <name>Password</name>
                                          <value><PASSWORD></value>
                                </property>
                     </properties>
          </connection-properties>
</connection-instance>
<connection-instance>
          <jndi-name>eis/tibjms/Queue</jndi-name>
          <connection-properties>
                     <properties>
                                 <property>
                                          <name>ConnectionFactoryLocation</name>
                                          <value>QueueConnectionFactory</value>
                                 </property>
                                 <property>
                                          <name>FactoryProperties</name>
<value>java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContext
Factory;java.naming.provider.url=tibjmsnaming://<HOST>:<PORT>;java.naming.secur
ity.principal=<USERNAME>;java.naming.security.credentials=;ThirdPartyJMSProvide
r=true</value>
                                 </property>
                                 <property>
                                          <name>AcknowledgeMode</name>
                                          <value>AUTO_ACKNOWLEDGE</value>
```

```
                                        </property>
                                        <property>
                                                <name>IsTopic</name>
                                                <value>false</value>
                                        </property>
                                        <property>
                                                <name>IsTransacted</name>
                                                <value>true</value>
                                        </property>
                                        <property>
                                                <name>Username</name>
                                                <value><USERNAME></value>
                                        </property>
                                        <property>
                                                <name>Password</name>
                                                <value><PASSWORD></value>
                                        </property>
                                </properties>
                        </connection-properties>
</connection-instance>
```

Note that the default <USERNAME> and <PASSWORD> are `admin` and `password`, respectively.

Alternatively, to configure a new connection factory by using the Oracle WebLogic Server Administration Console, use the steps mentioned in Section 2.18, "Adding an Adapter Connection Factory."

8.4.2.2 Direct Connection

Perform the following steps:

1. Copy the following file to the `<SOAInstall_DIR>/user_projects/domains/<DOMAIN_NAME>/lib` folder:
   - /<YOUR-TIBCO-INSTALL-LOCATION>/clients/java/tibjms.jar
2. Configure the connector factory by modifying the `weblogic-ra.xml` file in `AS11gR1SOA/soa/connectors/JmsAdapter.rar`, as shown in the following example:

```
<connection-instance>
          <jndi-name>eis/tibjmsDirect/Queue</jndi-name>
          <connection-properties>
                    <properties>
                                        <property>
                                                <name>ConnectionFactoryLocation</name>

<value>com.tibco.tibjms.TibjmsQueueConnectionFactory</value>
                                        </property>
                                        <property>
                                                <name>FactoryProperties</name>

<value>ServerUrl=tcp://<HOST>:<PORT>;UserName=<USERNAME>;UserPassword=<PASSWORD
></value>
                                        </property>
                                        <property>
                                                <name>AcknowledgeMode</name>
                                                <value>AUTO_ACKNOWLEDGE</value>
                                        </property>
                                        <property>
                                                <name>IsTopic</name>
```

```
                                        <value>false</value>
                                    </property>
                                    <property>
                                        <name>IsTransacted</name>
                                        <value>true</value>
                                    </property>
                                    <property>
                                        <name>Username</name>
                                        <value><USERNAME></value>
                                    </property>
                                    <property>
                                        <name>Password</name>
                                        <value><PASSWORD></value>
                                    </property>
                            </properties>
                    </connection-properties>
</connection-instance>
<connection-instance>
                    <jndi-name>eis/tibjmsDirect/Topic</jndi-name>
                    <connection-properties>
                            <properties>
                                     <property>

<name>ConnectionFactoryLocation</name>

<value>com.tibco.tibjms.TibjmsTopicConnectionFactory</value>
                                    </property>
                                    <property>
                                        <name>FactoryProperties</name>

<value>ServerUrl=tcp://<HOST>:<PORT>;UserName=<USERNAME>;UserPassword=<PASSWORD
></value>
                                    </property>
                                    <property>
                                        <name>AcknowledgeMode</name>
                                        <value>AUTO_ACKNOWLEDGE</value>
                                    </property>
                                    <property>
                                        <name>IsTopic</name>
                                        <value>true</value>
                                    </property>
                                    <property>
                                        <name>IsTransacted</name>
                                        <value>true</value>
                                    </property>
                                    <property>
                                        <name>Username</name>
                                        <value><USERNAME></value>
                                    </property>
                                    <property>
                                        <name>Password</name>
                                        <value><PASSWORD></value>
                                    </property>
                             </properties>
                    </connection-properties>
</connection-instance>
```

Note that the default <USERNAME> and <PASSWORD> are `admin` and `password`, respectively.

Alternatively, to configure a new connection factory by using the Oracle WebLogic Server Administration Console, use the steps mentioned in Section 2.18, "Adding an Adapter Connection Factory."

8.4.3 Configuring Oracle JMS Adapter with IBM WebSphere MQ JMS

This section describes how to configure Oracle JMS Adapter with IBM WebSphere MQ JMS for non-XA and XA data sources.

8.4.3.1 Non-XA Data Sources

Perform the following steps:

1. Copy the following files to the `<SOAInstall_DIR>/user_projects/domains/<DOMAIN_NAME>/lib` folder:
   - `/<YOUR-MQSERIES-INSTALL-LOCATION>/java/lib/com.ibm.mq.jar`
   - `/<YOUR-MQSERIES-INSTALL-LOCATION>/java/lib/com.ibm.mqjms.jar`
   - `/<YOUR-MQSERIES-INSTALL-LOCATION>/java/lib/dhbcore.jar`
2. Configure the connector factory by modifying the `weblogic-ra.xml` file in `AS11gR1SOA/soa/connectors/JmsAdapter.rar`, as shown in the following example:

```
<connection-instance>
<jndi-name>eis/webspheremq/Queue</jndi-name>
   <connection-properties>
      <properties>
         <property>
           <name>ConnectionFactoryLocation</name>
           <value>com.ibm.mq.jms.MQQueueConnectionFactory</value>
         </property>
         <property>
            <name>FactoryProperties</name>
            <value>QueueManager=<QUEUEMANAGER>;TransportType=1;HostName=
<YOUR-HOST>;Port=<YOUR-PORT>;Channel=<CHANNEL>;ThirdPartyJMSProvider=true
</value>
          </property>
          <property>
              <name>AcknowledgeMode</name>
              <value>AUTO_ACKNOWLEDGE</value>
          </property>
          <property>
             <name>IsTopic</name>
             <value>false</value>
          </property>
          <property>
             <name>IsTransacted</name>
             <value>true</value>
         </property>
         <property>
             <name>Username</name>
             <value><USERNAME></value>
         </property>
         <property>
            <name>Password</name>
            <value><PASSWORD></value>
        </property>
        </properties>
```

```
    </connection-properties>
</connection-instance>
```

Note that the default <USERNAME> and <PASSWORD> are `MUSR_MQADMIN` and `password`, respectively.

Alternatively, to configure a new connection factory by using the Oracle WebLogic Server Administration Console, use the steps mentioned in Section 2.18, "Adding an Adapter Connection Factory."

8.4.3.2 XA Data Sources

Perform the following steps:

1. Copy the following files to `the <SOAInstall_DIR>/user_projects/domains/<DOMAIN_NAME>/lib` folder:
   - `/<YOUR-MQSERIES-INSTALL-LOCATION>/java/lib/com.ibm.mq.jar`
   - `/<YOUR-MQSERIES-INSTALL-LOCATION>/java/lib/com.ibm.mqjms.jar`
   - `/<YOUR-MQSERIES-INSTALL-LOCATION>/java/lib/dhbcore.jar`
   - `com.ibm.mqetclient.jar`

     This is an IBM-extended transactional client, which is an optional component that requires separate licensing.
2. Configure the connector factory by modifying the `weblogic-ra.xml` file in `AS11gR1SOA/soa/connectors/JmsAdapter.rar`, as shown in the following example:

```
<connection-instance>
<jndi-name>eis/webspheremq/Queue</jndi-name>
   <connection-properties>
        <properties>
           <property>
              <name>ConnectionFactoryLocation</name>
              <value>com.ibm.mq.jms.MQXAQueueConnectionFactory</value>
           </property>
           <property>
              <name>FactoryProperties</name>
              <value>QueueManager=<QUEUEMANAGER>;TransportType=1;
HostName=<YOUR-HOST>;Port=<YOUR-PORT>;Channel=<CHANNEL>;
ThirdPartyJMSProvider=true</value>
           </property
           <property>
             <name>AcknowledgeMode</name>
             <value>AUTO_ACKNOWLEDGE</value>
           </property>
           <property>
             <name>IsTopic</name>
             <value>false</value>
           </property
           <property>
              <name>IsTransacted</name
              <value>false</value>
           </property
           <property>
              <name>Username</name>
              <value><USERNAME></value>
           </property>
```

```
            <property>
               <name>Password</name>
               <value><PASSWORD></value>
             </property>
          </properties>
      </connection-properties>
</connection-instance>
```

Note that the default <USERNAME> and <PASSWORD> are `MUSR_MQADMIN` and `password`, respectively.

Alternatively, to configure a new connection factory by using the Oracle WebLogic Server Administration Console, use the steps mentioned in Section 2.18, "Adding an Adapter Connection Factory."

8.4.4 WLS JMS Text Message

This WLS JMS text message use case for Oracle BPEL PM demonstrates how the Oracle JMS Adapter dequeues from and enqueues to the WLS JMS Queue.

In the case of a WLS JMS text message scenario for a Mediator business process, you need the following files from the `artifacts.zip` file contained in the `adapters-jms-101-wlsjms-textmessageusingqueues` sample:

- `artifacts/schemas/expense.xsd`

You can obtain the `adapters-jms-101-wlsjms-textmessageusingqueues` sample by accessing the Oracle SOA Sample Code site.

This section includes the following topics:

- Section 8.4.4.1, "Meeting Prerequisites"
- Section 8.4.4.2, "Creating an Application Server Connection"
- Section 8.4.4.3, "Creating an Application and an SOA Project"
- Section 8.4.4.4, "Creating an Inbound Adapter Service"
- Section 8.4.4.5, "Creating an Outbound Adapter Service"
- Section 8.4.4.6, "Wiring Services and Activities"
- Section 8.4.4.7, "Deploying with JDeveloper"
- Section 8.4.4.8, "Monitoring Using the Fusion Middleware Control Console"

8.4.4.1 Meeting Prerequisites

You must perform the following prerequisite for the WLS JMS text message use case for Oracle BPEL PM:

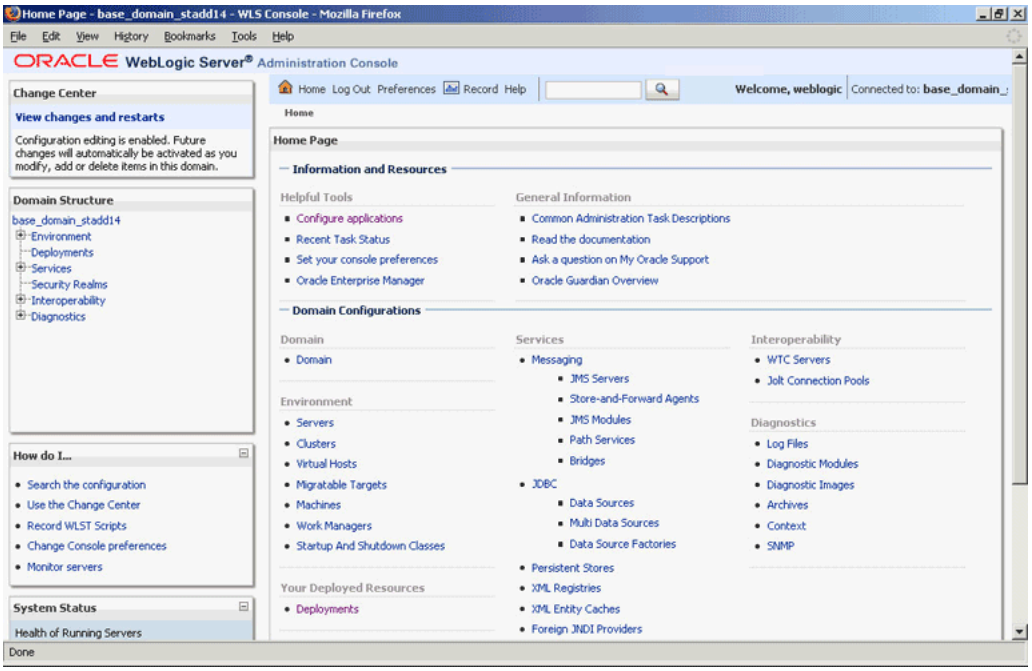
8.4.4.1.1 Creating Queues in the Oracle WebLogic Server Administration Console

Perform the following steps to create queues required for this use case:

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

   The Home page of the Oracle WebLogic Server Administration Console is displayed, as shown in Figure 8–14.

Figure 8–14 The Oracle WebLogic Server Administration Console Home Page



3. Navigate to **Services**, **Messaging**, **JMS Modules** in the Domain Structure pane.

   The Oracle WebLogic Server Administration Console - JMS Modules page is displayed.

4. Click one of the existing modules. In this example, click **SOAJMSModule**.

   The Oracle WebLogic Server Administration Console - Settings for SOAJMSModule page is displayed.

5. Under the Summary of Resources section, click **New**.

   The Oracle WebLogic Server Administration Console - Create a New JMS System Module Resource page is displayed.

6. Select **Queue**, and then click **Next**.

7. Enter the following queue details:

   - Name
   - JNDI Name
   - Template

8. Click **Next**.

9. Select the subdeployment you want to use from the Subdeployments list.

10. Click **Finish**.

    You have created the queue, `ReceiveQueue`.

11. Repeat steps 1 through 10, and create a queue named `SendQueue`.

8.4.4.1.2 Creating the Q2Qorders.xsd file

Create the Q2Qorders.xsd file by using the following code:

```
<?xml version="1.0" ?>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"

targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/extensions/FileInbound"
            xmlns:tns="http://xmlns.oracle.com/pcbpel/nxsd/extensions/FileInbound"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified" nxsd:encoding="US-ASCII"
nxsd:stream="chars" nxsd:version="NXSD">
  <xsd:element name="Items">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="item" minOccurs="1" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Name" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Type" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Quantity" type="xsd:string"
nxsd:style="terminated" nxsd:terminatedBy="," nxsd:quotedBy="&quot;">
              </xsd:element>
              <xsd:element name="Rate" type="xsd:string" nxsd:style="terminated"
nxsd:terminatedBy="${eol}" nxsd:quotedBy="&quot;">
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
<!--NXSDWIZ:C:\errors\inputFiles\orders.txt:-->
```

8.4.4.2 Creating an Application Server Connection

You must establish connectivity between the design-time environment and the server you want to deploy to. Perform the steps mentioned in Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters" to create an application server connection.

8.4.4.3 Creating an Application and an SOA Project

You must create an JDeveloper application to contain the SOA composite. Use the following steps to create an application and an SOA project:

1. Open JDeveloper.
2. In the **Application Navigator**, click **New Application**. The Create Generic Application - Name your Application dialog is displayed.
3. Enter a name for the application in the **Application Name** field. For example, `queue2queue`.
4. In the Application Template list, choose **Generic Application**.
5. Click **Next**.

   The Name your project page is displayed.

6. In the **Project Name** field, enter a descriptive name. For example, `queue2queue`.
7. In the **Available list** in the **Project Technologies** tab, double-click **SOA** to move it to the Selected list.
8. Click **Next**. The Create Generic Application - Configure SOA settings page is displayed.
9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

   You have created a new application, and an SOA project.

   The Create BPEL Process page is displayed.
10. Enter a name for the BPEL process in the **Name** field. For example, `queue2queue`.
11. Select **Define Interface Later** in the Template list, and then click **OK**.

    You have created a BPEL process.

    The `queue2queue` application, `queue2queue` project, and the SOA composite appear in the design area.
12. Copy the `Q2Qorders.xsd` file to the XSD folder in your project.

8.4.4.4 Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message to a queue:

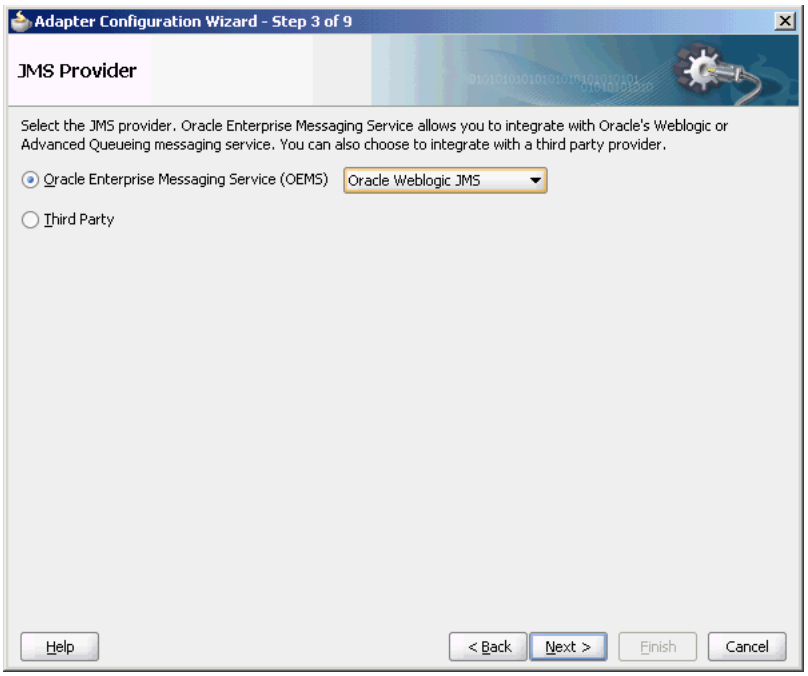
1. Drag and drop **JMS Adapter** from the Service Adapters list to the Exposed Services swim lane in the composite.xml page.

   The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**.

   The Service Name page is displayed.
3. Enter `Inbound` in the **Service Name** field, and click **OK**.

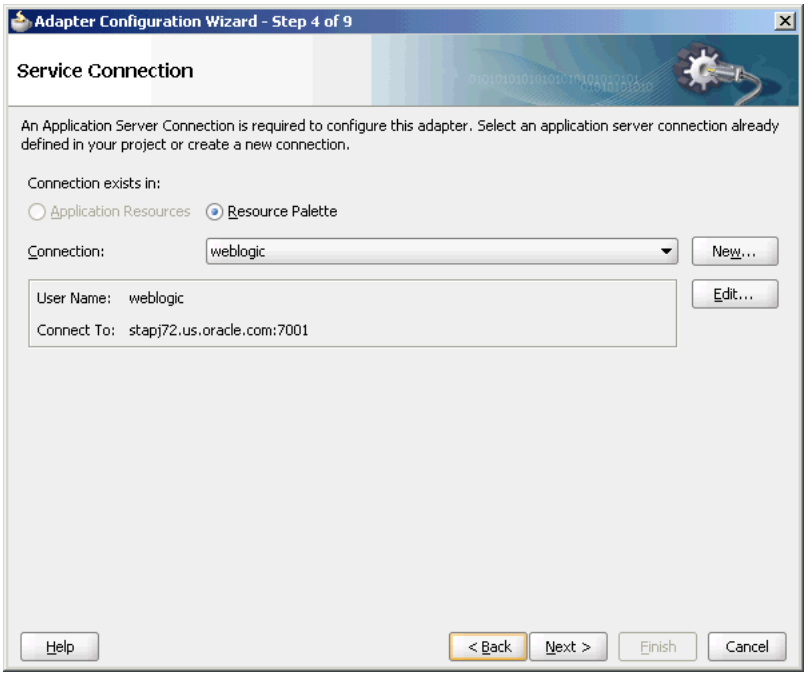
   The JMS Provider page is displayed.
4. Select **Oracle Weblogic JMS** in the Oracle Enterprise Messaging Service (OEMS) box, as shown in Figure 8–15, and click **Next**. The Service Connection page is displayed.

Figure 8–15 The Adapter Configuration Wizard JMS Provider Page



5. Select the connection created in Section 8.4.4.2, "Creating an Application Server Connection," as shown in Figure 8–16.

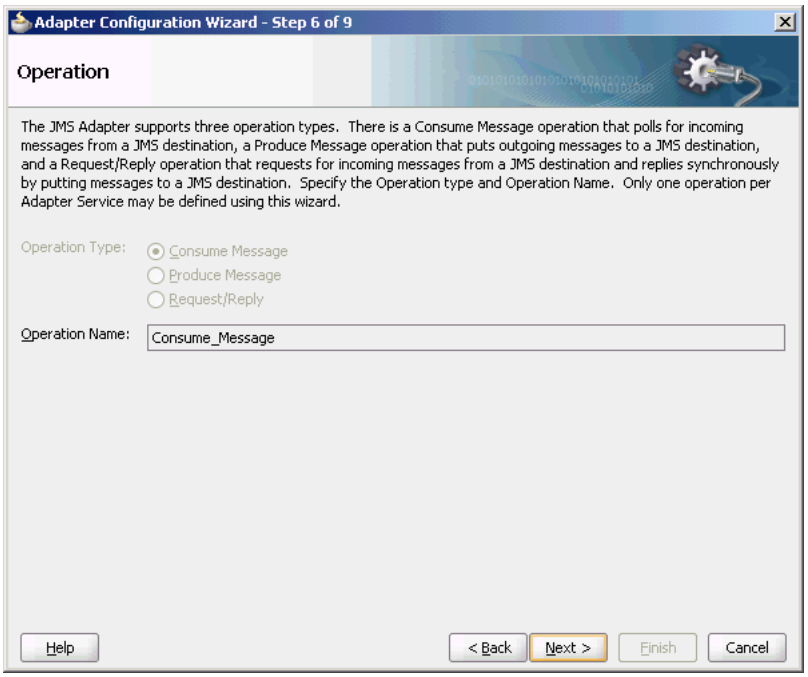
Figure 8–16 The Adapter Configuration Wizard Service Connection Page



6. Click **Next**. The Adapter Interface page is displayed.
7. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
8. Select **Consume Message**, as shown in Figure 8–17, and click **Next**.

The Consume Operation Parameters page is displayed.

Figure 8–17 The Adapter Configuration Wizard Operation Page



9. Click **Browse** and select **ReceiveQueue** in the Destination field.

   The Consume Operation Parameters page is displayed.

10. Enter the parameters for the consume operation, and then click **Next**.

    The Messages page is displayed.

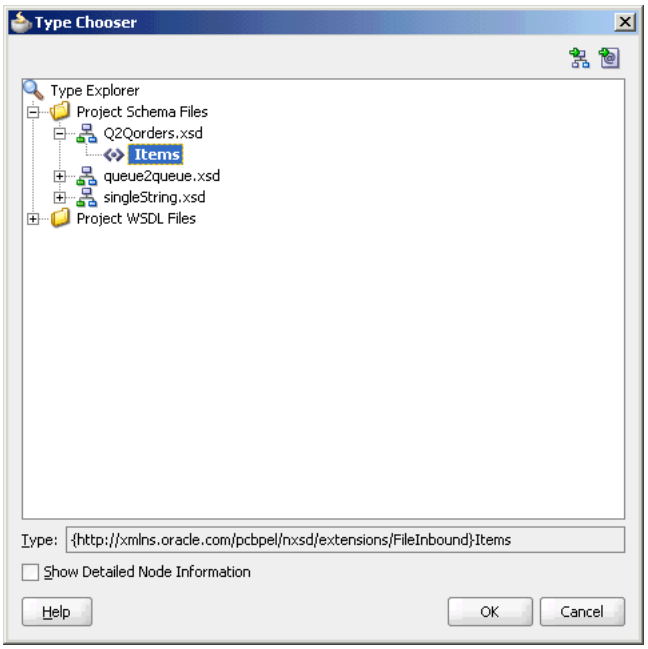
> **Note:** The value specified in the JNDI name should exist in the Oracle JMS Adapter `weblogic-ra.xml` file to ensure that the adapter runs in managed mode.

11. Click **Browse** at the end of the URL field.

    The Type Chooser dialog is displayed.

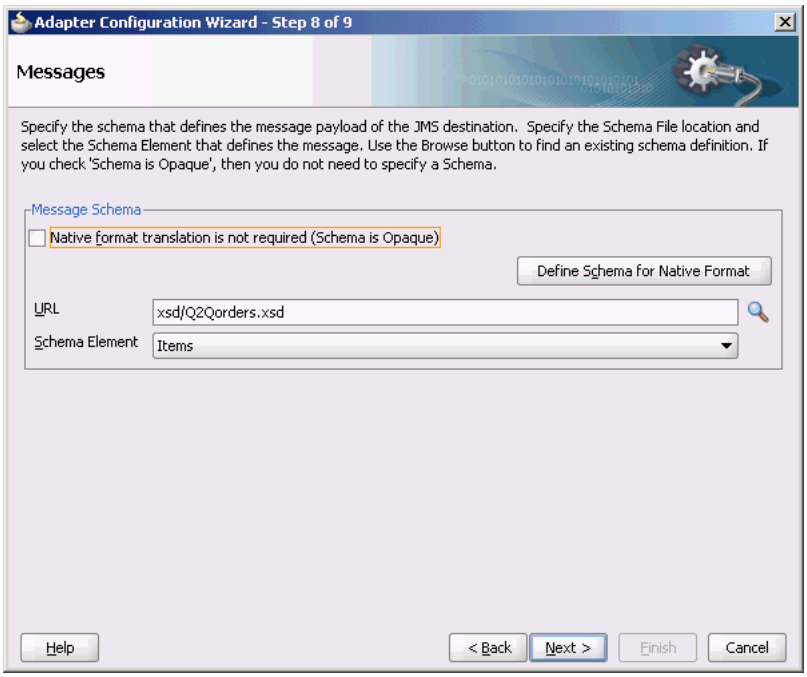
12. Select **Project Schema Files**, **Q2Qorders.xsd**, and **Items**, as shown in Figure 8–18.

Figure 8–18 The Type Chooser Dialog



13. Click **Next**. The Q2Qorders.xsd schema file is displayed in the URL in the Messages page, as shown in Figure 8–19.

Figure 8–19 The Adapter Configuration Wizard - Message Page



14. Click **Next**. The Finish page is displayed.
15. Click **Finish**. You have configured a JMS inbound adapter service.

8.4.4.5 Creating an Outbound Adapter Service

Perform the following steps to create an adapter service to enqueue the request messages and dequeue the corresponding response messages (report) from a queue:

1. Drag and drop **JMS Adapter** from the Component Palette into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter **Outbound** in the **Service Name** field, and click **OK**. The JMS Provider page is displayed.
4. Select **Oracle Weblogic JMS** in the Oracle Enterprise Messaging Service (OEMS) box, and click **Next**. The Service Connection page is displayed.
5. Select the connection created in Section 8.4.4.2, "Creating an Application Server Connection," and click **Next**. The Adapter Interface page is displayed.
6. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
7. Select **Produce Message**, and click **Next**. The Produce Operation Parameters page is displayed.
8. Click **Browse** and select **SendQueue** in the Destination field. The Produce Operation Parameters page is displayed.
9. Click **Next**. The Messages page is displayed.
10. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
11. Select **Project Schema Files**, **Q2Qorders.xsd**, and **Items**.
12. Click **Next**. The Q2Qorders.xsd schema file is displayed in the URL in the Message dialog.
13. Click **Next**. The Finish page is displayed.
14. Click **Finish**. You have configured the JMS adapter service, and the composite.xml page is displayed.

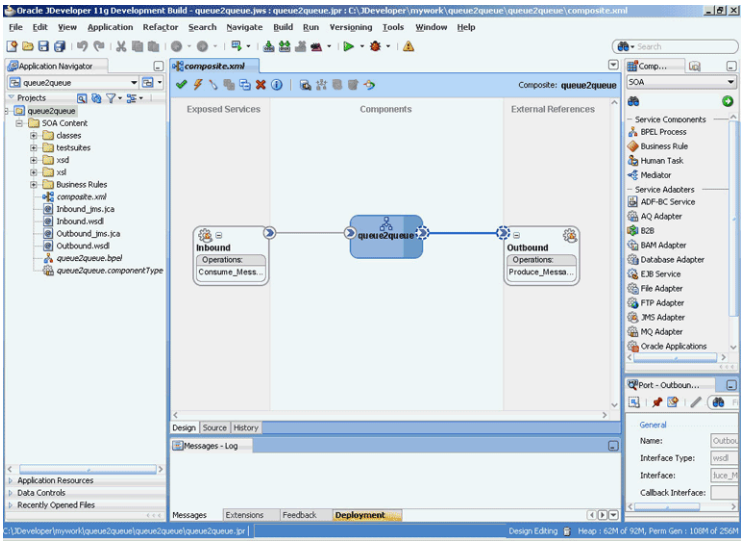
8.4.4.6 Wiring Services and Activities

You must wire the three components that you have created, Inbound adapter service, BPEL process, and Outbound adapter reference. Perform the following steps to wire components:

1. Drag the small triangle in the inbound Oracle JMS Adapter component in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the outbound Oracle JMS Adapter in the External References area.

   The JDeveloper Composite.xml is displayed, as shown in Figure 8–20.

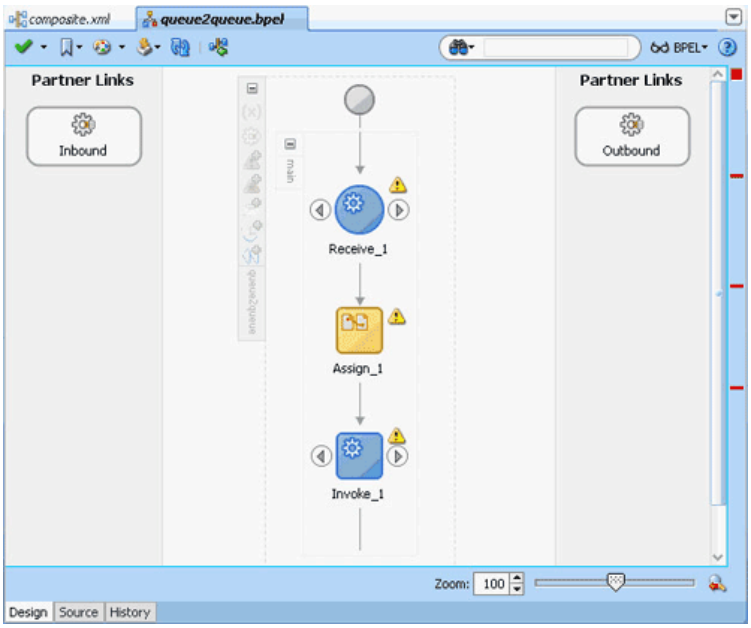
Figure 8–20 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.
4. Double-click **queue2queue**.

   The queue2queue.bpel page is displayed.
5. Drag and drop the **Receive**, **Assign**, and **Invoke** activities in the order mentioned from the Component Palette to the Components area, as shown in Figure 8–21.

Figure 8–21 The queue2queue.bpel Page



6. Double-click **Receive**.

   The Receive dialog is displayed.
7. Click the **Browse Partner Links** icon at the end of the Partner Link field.

   The Partner Link Chooser dialog is displayed.

8. Select **Inbound**, and then click **OK**.

   The Receive dialog is displayed with the Partner Link field populated with the value Inbound.

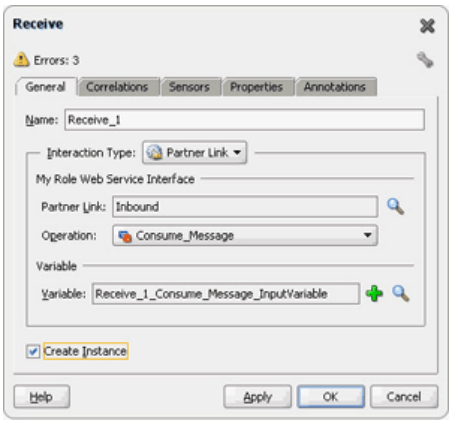
9. Click the **Auto-Create Variable** icon that is displayed at the end of the Variable field.

   The Create Variable dialog is displayed.

10. Accept the defaults, and click **OK**.

11. Select the **Create Instance** box, as shown in Figure 8–22, and click **OK**.

Figure 8–22 The Receive Dialog



12. Double-click the **Invoke** activity.

    The Invoke dialog is displayed.

13. Click the **Browse Partner Links** icon at the end of the Partner Link field.

    The Partner Link Chooser dialog is displayed.

14. Select **Outbound**, and then click **OK**.

    The Invoke dialog is displayed with the Partner Link field populated with the value Outbound.

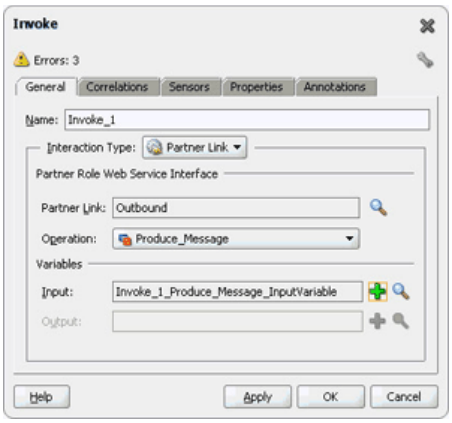
15. Click the **Automatically Create Input Variable** icon that is displayed at the end of the Input Variable field.

    The Create Variable dialog is displayed.

16. Accept the defaults, and click **OK**.

    The Invoke dialog is displayed, as shown in Figure 8–23.

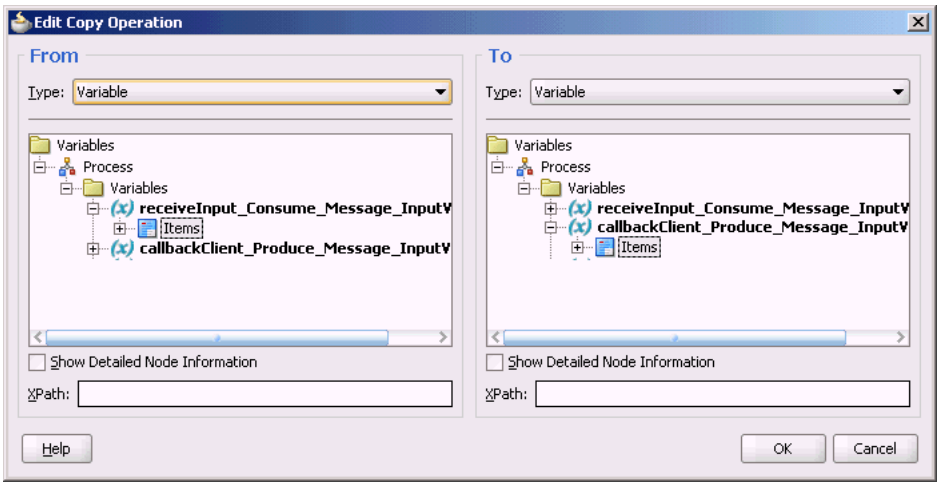
Figure 8–23 The Invoke Dialog



17. Click **OK**.
18. Double-click the **Assign** activity.

    The Assign dialog is displayed.
19. Click the plus icon, and select **Copy Operation**. The Create Copy Operation dialog is displayed.
20. Select the variables, as shown in Figure 8–24, and click **OK**.

Figure 8–24 The Create Copy Operation Dialog



21. Click **OK** in the Assign dialog.
22. Click **File**, **Save All**.

8.4.4.7 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, use the following steps:

1. Create an application server connection by using the procedure described in Chapter 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."

2. Deploy the application by using the procedure described in Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

8.4.4.8 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed composite by using the Fusion Middleware Control Console. Perform the following steps:

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

   The composite you deployed is displayed in the Application Navigator.

2. In the Last 5 Instances pane, there is an entry of a new instance. This is the instance that was triggered when you enqueued a message using queue2queue`.java`.

3. Click one of the instances. The Flow Trace page is displayed.

4. Click the **TextMessage** component instance. The Audit page is displayed.

5. Click the **Flow-Debug** tab to debug the instance.

8.4.5 Accessing Queues and Topics from WLS JMS Server in a Remote Oracle WebLogic Server Domain

You can use the Oracle JMS Adapter to access remote WLS JMS destinations. Remote destinations refer to queues or topics that are defined in a WLS JMS server, which is part of a remote Oracle WebLogic Server domain.

To do so, ensure that you use the connector factory configured to interact to the remote WLS JMS server. You can achieve this by setting the <FactoryProperties> property of the connector factory defined in weblogic-ra.xml to remote server configuration, as shown in the following example:

```
<property>
<name>FactoryProperties</name>
<value>java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory;
java.naming.provider.url= t3://<HOST>:<PORT>;java.naming.security.principal=
<USERNAME>;java.naming.security.credentials=<PASSWORD>
</value>
</property>
```

To enable Oracle JMS Adapter to read from a remote queue that is present in a remote WLS JMS server, you must configure the following:

1. You must have a unique domain name and JMS server name in both the servers.

2. You must enable global trust between the two servers.

   Refer to the following link for information about how to enable global trust between servers:

   http://download.oracle.com/docs/cd/E13222_01/wls/docs100/ConsoleHelp/taskhelp/security/EnableGlobalTrustBetweenDomains.html

   This configuration is appropriate when you connect to queues or topics present in WLS9.2 server.

8.4.5.1 JMS Adapter Limitations When a Remote Server is Used

The JMS Adapter enables you to interact with WebLogic Server JMS destination locations in a domain that are remote to the WebLogicServer domain where SOA is installed.

Two options are supported that enable you to access remote destinations via the JMS adapter:

- Direct access via specification of the `FactoryProperties` property in the weblogic-ra.xml file, with access parameters indicating the remote domain.
- Configuring the foreign server to access the remote domain.

For inbound use cases, both options are supported. For outbound use cases only, direct access is supported, but configuring the foreign server is not supported.

8.4.6 Synchronous/Asynchronous Request Reply Interaction Pattern

Oracle JMS Adapter supports both synchronous and asynchronous request reply interaction pattern.

8.4.6.1 Synchronous Request Reply Pattern

You can use the Adapter Configuration Wizard to model a process that enables Oracle JMS Adapter to be used in a synchronous request reply interaction pattern. In this case, the Oracle JMS Adapter sends a request to the request queue and waits for a response from the reply queue before further execution continues. Underneath, the Oracle JMS Adapter uses a new interaction pattern `JmsRequestReplyInteractionSpec`. This interaction spec allows for a request and reply destination name to be configured.

A variation, new to 11g Release 1 (11.1.1.4.0), allows usage of temporary destination as part of the reply queue. Basically, this pattern allows an Oracle JMS Adapter to send a message to a JMS destination. In turn, the adapter sets the `JMSReplyTo` header to the reply destination. This value is then used by a third party client to send the message to the reply destination which is then dequeued by the Oracle JMS Adapter.

When using the Oracle JMS Adapter in a synchronous pattern ensure that you use a non-XA connection factory and set the connector factory `isTransacted` property to `true` in `weblogic-ra.xml`.

When you use the Oracle JMS Adapter in a synchronous pattern with Oracle WebLogic Server JMS, the connection factory must be `weblogic.jms.ConnectionFactory` or any other non-XA connection factory. Also, if Oracle WebLogic Server JMS is running in the local JVM (the same JVM as the adapter), then you must ensure that the connector factory `isTransacted` property is set to `false` in `weblogic-ra.xml`. You can obtain the following samples by accessing the Oracle sample code site:

- `adapters-jms-106-wlsjms-syncrequestreply`
- `adapters-jms-107-wlsjms-syncrequestreplywithtemporaryreplydestination`

8.4.6.2 Asynchronous Request Reply Pattern

You can use the Adapter Configuration Wizard to model a process that allows Oracle JMS Adapter to be used in an asynchronous request reply interaction pattern.

Basically this pattern allows an Oracle JMS Adapter to send a message to a JMS destination. When a message is received on the reply queue, the Oracle JMS Adapter can route messages to the correct composite or the component instance. The correlation is done based on the `JMSMessageID` of the request message, which becomes the `JMSCorrelationID` of the reply message, and the conversation ID of the underlying component.

For more information, you can obtain the following samples by accessing the Oracle SOA Sample Code site.

- `adapters-jms-105-wlsjms-nativecorrelation`

8.4.7 AQ JMS Text Message

This use case demonstrates how the Oracle JMS Adapter dequeues from and enqueues to the AQ JMS Queue.

You can obtain the `adapters-jms-108-aqjms-textmessageusingqueues` sample by accessing the Oracle SOA Sample Code site.

This section includes the following topics:

- Section 8.4.7.1, "Meeting Prerequisites"
- Section 8.4.7.2, "Create an Application Server Connection"
- Section 8.4.7.3, "Creating an Application and an SOA Project"
- Section 8.4.7.4, "Creating an Inbound Adapter Service"
- Section 8.4.7.5, "Creating an Outbound Adapter Service"
- Section 8.4.7.6, "Wiring Services and Activities"
- Section 8.4.7.7, "Deploying with JDeveloper"
- Section 8.4.7.8, "Monitoring Using the Fusion Middleware Control Console"

8.4.7.1 Meeting Prerequisites

You must perform the following prerequisites to complete this use case:

- Section 8.4.7.1.1, "Configuring AQ JMS in Oracle WebLogic Server Administration Console"
- Section 8.4.7.1.2, "Creating Queues in Oracle Database"

8.4.7.1.1 Configuring AQ JMS in Oracle WebLogic Server Administration Console

To configure AQ JMS in Oracle WebLogic Server Administration Console, you must perform the following steps:

- Adding an Oracle WebLogic JMS Module
- Adding an AQJMS Foreign Server to the JMS Module
- Configuring the AQJMS Foreign Server
- Adding Connection Factories to the AQ JMS Foreign Server
- Adding Destinations to the AQJMS Foreign Server

Adding an Oracle WebLogic JMS Module

Note that adding an Oracle WebLogic JMS module is optional. You can also create an AQJMS foreign server in a preexisting JMS module.

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

The Home page of the Oracle WebLogic Server Administration Console is displayed.

3. Navigate to **Services**, **Messaging**, **JMS Modules** in the Domain Structure pane.

   The Oracle WebLogic Server Administration Console - JMS Modules page is displayed.

4. Click **New** to create a new WebLogic JMS module.

   The Oracle WebLogic Server Administration Console - Create JMS System Module page is displayed.

5. Enter a name for the JMS module, and then click **Next**.

   The Oracle WebLogic Server Administration Console - Create JMS System Module page is displayed.

6. Select a target server where your SOA component is running, and then click **Next**.

   The Oracle WebLogic Server Administration Console - Create JMS System Module page is displayed.

7. Click **Finish**.

   You have created a JMS module.

Adding an AQJMS Foreign Server to the JMS Module

The next step is to add an AQ JMS foreign server to the JMS module by performing the following:

1. Click the JMS module that you created.

   The Oracle WebLogic Server Administration Console - Settings for AQJMSModule page is displayed.

2. Click **New** in the Summary of Resources table to create a new JMS system module resource.

   The Oracle WebLogic Server Administration Console - Create a New JMS System Module Resource page is displayed.

3. Under Choose the type of resource you want to create, select **Foreign Server**, and then click **Next**.

   The Oracle WebLogic Server Administration Console - Create a New JMS System Module Resource page is displayed.

4. In the **Name** field, enter a name for the foreign server, and then click **Finish**.

   The Oracle WebLogic Server Administration Console - Settings for <JMS Module Name> page is displayed.

Configuring the AQJMS Foreign Server

The next step is to configure the AQJMS foreign server that you created:

1. Click the AQ JMS Foreign Server listed under the Summary of Resources table.

   The Oracle WebLogic Server Administration Console - Settings for TestAQJMS_ ForeignServer page is displayed.

2. Enter the following values:

   - **JNDI Initial Context Factory**: `oracle.jms.AQjmsInitialContextFactory`

If the AQJMS Foreign Server is used by the WebLogic server side components, then you must configure a data source with this AQ JMS Foreign Server, by specifying the following values:

In the **JNDI Properties** field, enter **datasource=<datasource jndi location>**. Replace the place holder with the JNDI location of your data source.

However, if the AQJMS Foreign Server is used by WebLogic application client, then you must configure the JDBC URL with the AQ JMS foreign server you created.

- **JNDI Connection URL**: Specify the URL that WebLogic Server uses to contact the JNDI provider.

  This value is required only if the AQJMS foreign server is used by the WebLogic application client.

- **JNDI Properties Credential:** Specify any Credentials that must be set for the JNDI provider.

  This value is required only if the AQJMS foreign server is used by the Weblogic application client.

Note: If you want to use an Oracle RAC database as adapter endpoint, then the datasource pointed by the JNDI property, mentioned in the preceding step, must point to a multi data source.

Individual data sources and multi data sources used for such endpoints must use the recommended setting listed in Section 2.20, "Recommended Setting for Data Sources Used by Oracle JCA Adapters."

Adding Connection Factories to the AQ JMS Foreign Server

To add connection factories to the AQJMS foreign server:

1. In the Connection Factories tab in the Settings for <Foreign Server Name> page, click the AQJMS foreign server that you created.
2. Click **New**.

   The Oracle WebLogic Server Administration Console - Create a New Foreign JMS Connection Factory page is displayed.
3. In the **Name** field, enter a name for this connection factory. This is a logical name that would be referenced by Oracle WebLogic Server.
4. In the **Local JNDI Name** field, enter the local JNDI name that you would use in your application to look up this connection factory.

Note: Ensure that you specify `aqjms/XAQueueConnectionFactory` for local JNDI name if you are connecting to a queue with JNDI name eis/aqjms/Queue that is provided with the sample use case, `AQQueuetoQueue`.

Else, specify `aqjms/XATopicConnectionFactory` if you are connecting to a topic with JNDI name `eis/aqjms/Topic`.

---

5. In the **Remote JNDI Name** field, enter one of the following values depending on your requirement. If you use this connection factory in a global transaction, then use an XA-based connection factory, else use non-XA based connection factory.
   - `QueueConnectionFactory`
   - `TopicConnectionFactory`
   - `ConnectionFactory`
   - `XAQueueConnectionFactory`
   - `XATopicConnectionFactory`
   - `XAConnectionFactory`
6. Click **OK**.

Adding Destinations to the AQJMS Foreign Server

To add destinations to the AQJMS foreign server:

1. Click the **Destinations** tab in the Settings for <Foreign Server Name> page.
2. Click **New** and specify a name for this destination. This is a logical name that is referenced by the Oracle WebLogic Server and has nothing to do with the destination name.
3. In the **Local JNDI Name** field, enter the local JNDI name you would use in your application to look up this destination.
4. In the **Remote JNDI Name** field, enter `Queues/<queue name>`if the destination is a queue, or enter `Topics/<topic name>` if the destination is a topic.
5. Click **OK**.
6. Restart the Oracle WebLogic Server Administration Console.

You have configured AQJMS in an Oracle WebLogic Server.

8.4.7.1.2 Creating Queues in Oracle Database

To create queues:

1. Run the `setup_user.sql` script.
2. Run the `create_start_queues.sql` script.

These scripts are located in the `adapters-jms-108-aqjms-textmessageusingqueues` sample `artifacts/sql` directory. You can obtain the `adapters-jms-108-aqjms-textmessageusingqueues` sample by accessing the Oracle SOA Sample Code site.

8.4.7.2 Create an Application Server Connection

You must establish connectivity between the design-time environment and the server you want to deploy to. Perform the steps mentioned in Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters" to create an application server connection.

8.4.7.3 Creating an Application and an SOA Project

You must create an JDeveloper application to contain the SOA composite. Use the following steps to create a new application and an SOA project:

1. Open JDeveloper.
2. In the Application Navigator, click **New Application**. The Create Generic Application - Name your Application dialog is displayed.
3. Enter a name for the application in the **Application Name** field. For example, `AQQueue2Queue`.
4. In the Application Template list, choose **Generic Application**.
5. Click **Next**.

   The Name your project page is displayed.
6. In the **Project Name** field, enter a descriptive name. For example, `AQQueue2Queue`.
7. In the Available list in the **Project Technologies** tab, double-click **SOA** to move it to the Selected list.
8. Click **Next**. The Create Generic Application - Configure SOA Settings page is displayed.
9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

   You have created a new application and an SOA project.

   The Create BPEL Process page is displayed.
10. Enter a name for the BPEL process in the **Name** field.
11. Select **Define Interface Later** in the Template list, and then click **OK**.

    You have created a BPEL process.

    The `AQQueue2Queue` application, the `AQQueue2Queue` project, and the SOA composite appear in the design area.
12. Copy the `expense.xsd` file to the XSD folder in your project.

    This file is located in the `adapters-jms-108-aqjms-textmessageusingqueues` sample `artifacts/schemas` directory. You can obtain the `adapters-jms-108-aqjms-textmessageusingqueues` sample by accesing the Oracle SOA Sample Code site, and selecting the Adapters tab.

8.4.7.4 Creating an Inbound Adapter Service

Perform the following steps to create an adapter service to dequeue the message to a queue:

1. Drag and drop **JMS Adapter** from the Service Adapters list to the Exposed Services swim lane in the composite.xml page. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter **Inbound** in the **Service Name** field, and click **OK**. The JMS Provider page is displayed.

4. Select **Oracle Advanced Queueing** in the Oracle Enterprise Messaging Service (OEMS) box, and click **Next**. The Service Connection page is displayed.
5. Select the connection created in Section 8.4.4.2, "Creating an Application Server Connection."
6. Click **Next**. The Adapter Interface page is displayed.
7. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
8. Select **Consume Message**, and click **Next**. The Consume Operation Parameters page is displayed.
9. Click **Browse** and select **testInQueue** in the Destination field.
10. Click **Next**. The Messages page is displayed.
11. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
12. Select **Project Schema Files**, **expense.xsd**.
13. Click **Next**. The expenses.xsd schema file is displayed in the URL field in the Messages page.
14. Click **Next**. The Finish page is displayed.
15. Click **Finish**. You have configured a JMS inbound adapter service.

8.4.7.5 Creating an Outbound Adapter Service

Perform the following steps to create an adapter service that enqueues the request messages and dequeue the corresponding response messages (report) from a queue:

1. Drag and drop **JMS Adapter** from the Component Palette into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter **Outbound** in the **Service Name** field, and click **OK**. The JMS Provider page is displayed.
4. Select **Oracle Advanced Queueing** in the Oracle Enterprise Messaging Service (OEMS) box, and click **Next**. The Service Connection page is displayed.
5. Select the connection created in Section 8.4.4.2, "Creating an Application Server Connection," and click **Next**. The Adapter Interface page is displayed.
6. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation page is displayed.
7. Select **Produce Message**, and click **Next**. The Produce Operation Parameters page is displayed.
8. Click **Browse** and select **testOutQueue** in the Destination field. The Produce Operation Parameters page is displayed.
9. Click **Next**. The Messages page is displayed.
10. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
11. Select **Project Schema Files**, **expense.xsd**.
12. Click **Next**. The expense.xsd schema file is displayed in the URL field in the Message dialog.
13. Click **Next**. The Finish page is displayed.

14. Click **Finish**. You have configured the JMS adapter service, and the composite.xml page is displayed.

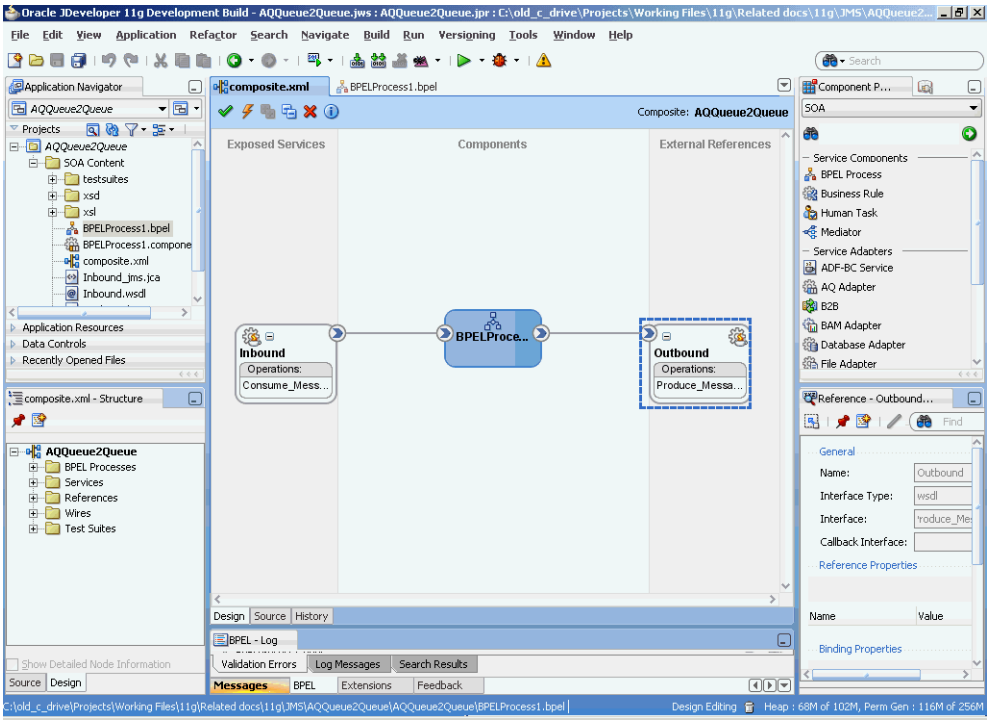
8.4.7.6 Wiring Services and Activities

You must wire the three components that you have created: Inbound adapter service, BPEL process, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the inbound Oracle JMS Adapter component in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in the outbound Oracle JMS Adapter in the External References area.

   The JDeveloper Composite.xml is displayed, as shown in Figure 8–25.

Figure 8–25 JDeveloper - Composite.xml



3. Click **File**, **Save All**.
4. Double-click the BPEL process. The BPELProcess1.bpel page is displayed.
5. Drag and drop the **Receive**, **Assign**, and **Invoke** activities, in the order mentioned, from the Component Palette to the Components area.
6. Double-click the **Receive** activity.

   The Receive dialog is displayed.
7. Click the **Browse Partner Links** icon at the end of the Partner Link field.

   The Partner Link Chooser dialog is displayed.
8. Select **Inbound**, and then click **OK**.

The Receive dialog is displayed with the Partner Link field populated with the value Outbound.

9. Click the **Auto-Create Variable** icon that is displayed at the end of the Variable field. The Create Variable dialog is displayed.
10. Accept the defaults, and click **OK**.
11. Check the **Create Instance** box.
12. Double-click the **Invoke** activity to Outbound.

    The Invoke dialog is displayed.
13. Click the **Automatically Create Input Variable** icon that is displayed at the end of the Input Variable field.
14. Click the **Browse Partner Links** icon at the end of the Partner Link field.

    The Partner Link Chooser dialog is displayed.
15. Select **Outbound**, and then click **OK**.

    The Invoke dialog is displayed with the Partner Link field populated with the value Outbound.
16. Accept the defaults, and click **OK**.
17. Click **OK**.
18. Double-click the **Assign** activity.

    The Assign dialog is displayed.
19. Click the plus icon, and select **Copy Operation**.

    The Create Copy Operation dialog is displayed.
20. Select the variables, and click **OK**.
21. Click **OK** in the Assign dialog.
22. Click **File**, **Save All**.

Note: When using Oracle JMS Adapter to dequeue from AQ JMS Topics with durable subscriptions, if you notice that the dequeue operation exhibits slow performance, then you can speed up the entire performance by using multiple inbound threads for each adapter service.

Oracle JMS Adapter allows multiple inbound threads if you specify an endpoint property `adapter.jms.receive.threads`.

However, note that this workaround is not applicable when using non-durable subscriptions because doing so results in duplicate messages.

8.4.7.7 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile by using JDeveloper, perform the following steps:

1. Create an application server connection by using the procedure described in Chapter 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."

2. Deploy the application by using the procedure described in Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

8.4.7.8 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed composite by using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed is displayed in the Application Navigator.
2. In the Last 5 Instances pane, there is an entry of a new instance. This is the instance that was triggered when you enqueued a message by using AQQueue2Queue`.java`.
3. Click one of the instances. The Flow Trace page is displayed.
4. Click the TextMessage component instance. The Audit page is displayed.
5. Click the **Flow-Debug** tab to debug the instance.

8.4.8 Accessing Queues and Topics Created in 11*g* from the OC4J 10.1.3.4 Server

This section describes the procedure for accessing queues and topics you created in Oracle Application Server 11*g* from OC4J 10.1.3.4. To do this, you must configure Oracle BPEL PM JMS adapter with Oracle WebLogic Server.

The following are the steps to configure Oracle BPEL PM JMS adapter with Oracle WebLogic Server:

1. Create the `wlfullclient.jar` file using the following steps:
   a. Change to the `server/lib` directory, as shown in the following example:

      ```
      cd WL_HOME/server/lib
      ```

   b. Use the following command to create the `wlfullclient.jar` file in the `server/lib` directory:

      ```
      java -jar ../../../modules/com.bea.core.jarbuilder_X.X.X.X.jar
      ```

      where `X.X.X.X` is the version number of the jarbuilder module in the *WL_HOME*`/server/lib` directory. For example:

      ```
      java -jar ../../../modules/com.bea.core.jarbuilder_1.0.1.0.jar
      ```

2. Copy the `wlfullclient.jar` file to the 10.1.3.4. server at the following location:

   ```
   <ORACLEAS_HOME>/j2ee/<OC4J_INSTANCE>/connectors/JmsAdapter/JmsAdapter
   ```

3. Configure the connector factory setting in the `oc4j-ra.xml` file, as shown in the following example:

   ```
   <connector-factory location="eis/wlsjms/Queue"
   connector-name="Jms Adapter">
           <config-property name="connectionFactoryLocation"
   value="weblogic.jms.ConnectionFactory"/>
           <config-property name="factoryProperties"
   value="java.naming.factory.initial=weblogic.jndi.
   WLInitialContextFactory;java.naming.provider.url=t3://w3.us.oracle.com:7001;
   java.naming.security.principal=weblogic;java.naming.security.credentials=
   weblogic"/>
          <config-property name="acknowledgeMode" value="AUTO_ACKNOWLEDGE"/>
          <config-property name="isTopic" value="false"/>
   ```

```
        <config-property name="isTransacted" value="false"/>
        <config-property name="username" value=""/>
        <config-property name="password" value=""/>
    <connection-pooling use="none">
    </connection-pooling>
      <security-config use="none">
      </security-config>
</connector-factory>
```

Note: The `isTransacted` configuration property value must typically be set to `FALSE`. Currently, XA integration with WebLogic JMS is not supported unless the adapter is deployed on Oracle WebLogic Server.

4. Modify the `server.xml` file of the 10.1.3.4 server to include the `environment-naming-url-factory-enabled="true"` property, as shown in the following example:

```
<application-server
...
...
environment-naming-url-factory-enabled="true"
...
>
```

5. Restart the 10.1.3.4 server to make the changes come into effect.

8.4.9 Configuring the 11G Server to Access Queues Present in 10.1.3.X OC4J

You can configure your 11G server to access queues present in 10.1.3.x OC4J with the following steps.

8.4.9.1 Copy Jar Files into the domains Folder of the Web Logic Server

Copy the following jar files under the `domains/<DOMAIN_NAME>/lib` folder of the WebLogic Server:

- `$J2EE_HOME/lib/jms.jar`
- `$J2EE_HOME/lib/jta.jar`
- `$J2EE_HOME/oc4jclient.jar`
- `$AS_HOME/opmn/lib/optic.jar`

8.4.9.2 Add Connector factory in the weblogic-ra.xml File

The next step is to add the Connector Factory in the `weblogic-ra.xml` file:

```
<connection-instance>
   <jndi-name>eis/oc4jjms/Queue</jndi-name>
    <connection-properties>
       <properties>
        <property>
          <name>ConnectionFactoryLocation</name>
          <value>jms/XAQueueConnectionFactory</value>
        </property>
        <property>
          <name>FactoryProperties</name>
```

```
<value>java.naming.factory.initial=com.evermind.server.rmi.RMIInitialContextFactor
y;java.naming.provider.url= <PROVIDER_
URL>;java.naming.security.principal=oc4jadmin;
java.naming.security.credentials=welcome1</value>
        </property>
        <property>
          <name>AcknowledgeMode</name>
          <value>AUTO_ACKNOWLEDGE</value>
       </property>
       <property>
         <name>IsTopic</name>
         <value>false</value>
       </property>
       <property>
         <name>IsTransacted</name>
         <value>false</value>
       </property>
       <property>
         <name>Username</name>
         <value>oc4jadmin</value>
       </property>
       </property>
       <property>
         <name>Password</name>
        <value>welcome1</value>
     </property>
    </properties>
   </connection-properties>
  </connection-instance>
```

where `<PROVIDER_URL>=opmn://localhost:6003` or, `ormi://localhost:12401` to use against a specific node or, `opmn:ormi://localhost:6003:oc4j_soa` to use against the `oc4j_soa` instance.

8.4.10 Accessing Distributed Destinations (Queues and Topics) on the WebLogic Server JMS

A distributed destination is a set of destinations (queues, set of physical JMS queue members, or topics, set of physical JMS topic members) that are accessible as a single, logical destination to a client.

> **Note:** For more information on distributed destinations, and a definition of terms used in this context, visit the Using Distributed Destinations pages at http://download.oracle.com/docs/cd/E13222_01/wls/docs103//jms/dds.html

The JMS Adapter can process messages addressed to a distributed destination member after receiving available notification; it can process available, unavailable, and failure notifications related to a distributed destination member.To have the JMS Adapter process such messages, you must provide additional properties to provide access to distributed destinations (queues or topics) with the JMS Adapter.

When you provide additional queues or topics, you can separate multiple FactoryProperty values with a semicolon. See the following example.

```
<property>
```

```
  <name>FactoryProperties</name>
  <value>ClientID=SOACLient2; TopicMessageDistributionAll=true</value>
</property>
```

8.4.10.1 Providing JMS Adapter Access to Distributed Destinations

You use three FactoryProperty parameter values to provide adapter access to distributed destinations, to specifically enable the Client ID to be shared by multiple connections, to enable the sharing of Durable subscriptions among multiple subscribers, and to specify whether you want one copy of a message per application or per endpoint. The properties include:

- `ClientIDPolicy`

  Use the FactoryProperties parameter `ClientIDPolicy` property with a value of `UNRESTRICTED` to enable the Client ID to be shared by multiple connections. The default, if no value is specified, is `UNRESTRICTED`. The non-default value is `RESTRICTED`. The default is used in almost all uses, so typically you do not have to set it. See the following example:

  ```
  </property>
     <name>FactoryProperties</name>
  <value>ClientIDPolicy=UNRESTRICTED</value>

   </property>
  ```

- `SubscriptionSharingPolicy`

  Use the FactoryProperties parameter with a value of `SHARABLE` to enable the sharing of Durable Subscriptions among multiple subscribers.

  A value of SubscriptionSharingPolicy `EXCLUSIVE` means you cannot share Durable Subscriptions among multiple subscribers. If you do not specify a value, the default is `SHARABLE`; in most cases, you do not have to change the value.

  ```
   <property>
      <name>FactoryProperties</name>
      <value>SubscriptionSharingPolicy=SHARABLE</value>
   </property>
  ```

- `TopicMessageDistributionAll`

  See the section on Distirbuted Topics for more information on the `TopicMessageDistributionAll` FactoryProperties parameter. You can set it as in the following example:

  ```
  <property>
      <name>FactoryProperties</name>
      <value>TopicMessageDistributionAll=true</value>
  </property>
  ```

8.4.10.2 The JMS Adapter with Distributed Queues and Distributed Topics

Specific inbound and outbound queue and error handling behaviors apply to the JMS Adapter with JMS Distributed Queues and Distributed Topics.

For inbound queues, the JMS Adapter creates an inbound poller thread and registers a notification listener with the WebLogic Server JMS on endpoint activation; it unregisters notification listener upon endpoint deactivation.

The JMS Adapter handles errors in the Distributed environment in the same fashion as such errors are handled in a non-Distributed environment: retriable exceptions lead to message retry; non-retriable exceptions lead to message rejection.

There is no change from the behavior of other Adapters to JMS adapter behavior when the Adapter produces a message to a Distributed Queue.

JMS messages for Distributed Destinations are produced by creating a MessageProducer for the Distributed Destination and not for a specific member.

Outbound errors are processed based on fault-policies previously defined for the outbound reference.

For inbound adapters with distributed topics, the JMS Adapter registers a notification listener with the WebLogic Server JMS on endpoint activation. The JMS Adapter creates an inbound poller thread for each available notification received from WebLogic Server JMS for a Distributed Topic member.

The inbound poller thread stops working and necessary cleanup is performed if an unavailable notification is received for the member for which the poller thread was created. The durable subscription is maintained in a similar fashion as in a non-Distributed topic scenario.

The Adapter unregisters the notification listener upon endpoint deactivation. Any message arriving at a Distributed Topic is processed based on the various settings used and the type of Distributed Destination in use: either one copy of a message per application, or one copy of a message per adapter endpoint.

The behaviors for each of these types of Distributed Destination are provided below.

8.4.10.3 One Copy of a Message Per Application (Default Behavior)

The default behavior for WebLogic Server Partitioned Distributed Topics when used with the JMS Adapter is to provide one copy of a message per application. Each message must be processed exactly once (that is, there is no duplicate processing.). In this scenario, where there is one copy of a message per application, the client id and subscription name are the same for every Distributed Destination and each adapter instance creates subscriptions on every member. The name is unique and immutable across server restarts.

When using Partitioned Distributed Topics you must configure the JMS adapter to use unrestricted clientid and shared subscription policy. These two are the default settings for Distributed Destinations.

When using Replicated Distributed Topics, you must configure JMS adapter to use the unrestricted clientid and shared subscription policy, which are the default settings. To achieve better performance you should use Partitioned Distributed Topics.

In addition, you must specify the following message selector, `NOT JMS_WL_DDForwarded` when defining an activation spec.

Refer to the following example, consisting of a snippet of a connection instance from the `weblogic-ra.xml` file for a local cluster:

```
<property>
    <name>FactoryProperties</name>
    <value>ClientID=SOAClient1;</value>
</property>
```

8.4.10.4 One Copy Of a Message Per Adapter Endpoint

The second type of scenario you can employ with Distributed Topics is to have one copy of message per adapter endpoint. In this case, either the client id or the subscription name is unique for each adapter instance. The unique part of the member name is immutable across server restarts.

When using Partitioned Distributed Topics you have to configure the JMS adapter to use unrestricted clientid and shared subscription policy, which are the default settings. At the same time, to achieve subscription name uniqueness, JMS adapter requires that the property `TopicMessageDistributionAll` (default value of false) is set to true. You can define this property by setting the `FactoryProperties` property of the connection instance in the `weblogic-ra.xml` file. An example usage (a snippet of connection instance from a `weblogic-ra.xml` file for a local cluster) is shown below.

```
<name>FactoryProperties</name>
<value>ClientID=SOAClient2;TopicMessageDistributionAll=true</value>
</property>
```

In addition, specify the message selector `(NOT_JMS_WL_DDForwarded)` when defining an activation spec.

To achieve better performance, you should use Partitioned Distributed Topics.

When using Replicated Distributed Topics, configure the JMS adapter to use unrestricted clientid and shared subscription policy, which are the default settings. At the same time, to achieve subscription name uniqueness, the JMS adapter requires that the property `TopicMessageDistributionAll` (default value of false) is set to true. You can define this property by setting the `FactoryProperties` property of the connection instance in `weblogic-ra.xml`. An example usage (snippet of connection instance from `weblogic-ra.xml` for a local cluster) is as shown below:

```
<name>FactoryProperties</name>
<value> ClientID=SOAClient2;TopicMessageDistributionAll=true</value>
</property>
```

8.4.10.4.1 Specifying the Message Selector when Defining an Activation Spec

Specify a message selector when defining an activation spec. The message selector is required when you create one copy of message per adapter Endpoint.

To specify the selector, use the Adapter Wizard when modeling a composite application that reads from Replicated Distributed Topic The metadata for the message selector you specify are captured in the `.jca` file.

Below is an example of a message selector defined in an activation spec.This message selector filters out the copy of the forwarded message when sending a message to a destination subscriber. This message selector is only applicable for when using Replicated Distributed Topics.

```
<activation-spec
className="oracle.tip.adapter.jms.inbound.JmsConsumeActivationSpec">
     <property name="DestinationName" value="jms/DemoInTopic"/>
     <property name="UseMessageListener" value="false"/>
     <property name="DurableSubscriber" value="dsub1"/>
     <property name="MessageSelector" value="NOT JMS_WL_DDForwarded"/>
     <property name="PayloadType" value="TextMessage"/>
</activation-spec>
```

With Distributed Topics, retriable exceptions lead to message retry, while non-retriable exceptions lead to message rejection.

Available/Unavailable/Failure notification does not impact the working of the outbound adapter reference. The message is produced by creating a MessageProducer for the Distributed Destination and not for a specific member.

In the Distributed Topics environment, as elsewhere, an error is processed based on the fault policies defined for the outbound reference.

8.4.10.4.2 Compatibility and Migration

Remote Distributed Queue support is feasible all the way back to WLS JMS version 9.0 using the new DestinationAvailabilityListener API. A remote Distributed Topic cannot be supported if it is older than WebLogic 10.3.4, as “shared subscriptions”, “unrestricted client ids”, the “not forwarded” selector, and even “partitioned” Distributed Topics are not supported. Users will need to instead directly reference a DT member JNDI name, and deal with the single subscriber per subscription limitation.

8.4.10.5 Configuring Request-Reply in JMS Adapter

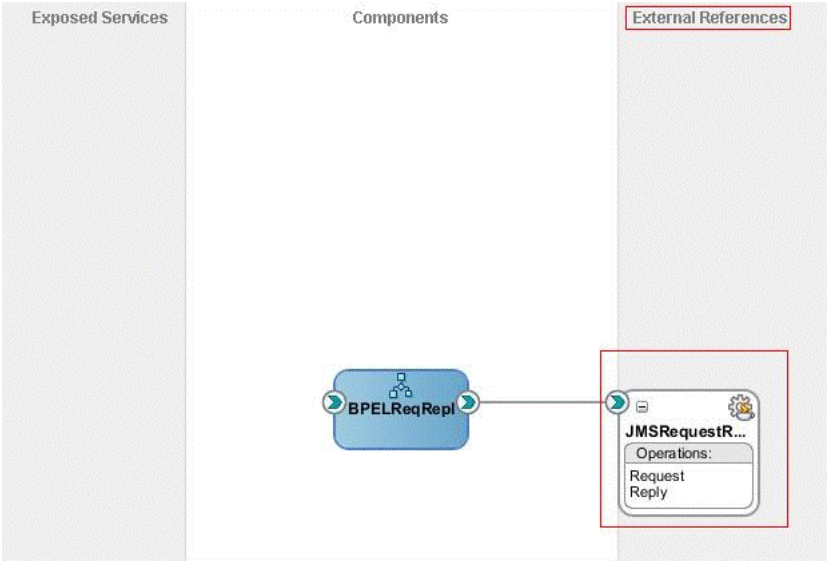
The Request-Reply configuration feature enables you to perform the following:

- Combine Request and Reply in a single configuration step. In the prior releases of the Oracle SOA Suite, you would require to configure two distinct adapters.
- Automatic correlation without your needing to configure BPEL correlation set. This works seamlessly in Mediator and BPMN as well.

To configure the JMSAdapter Request-Reply feature:

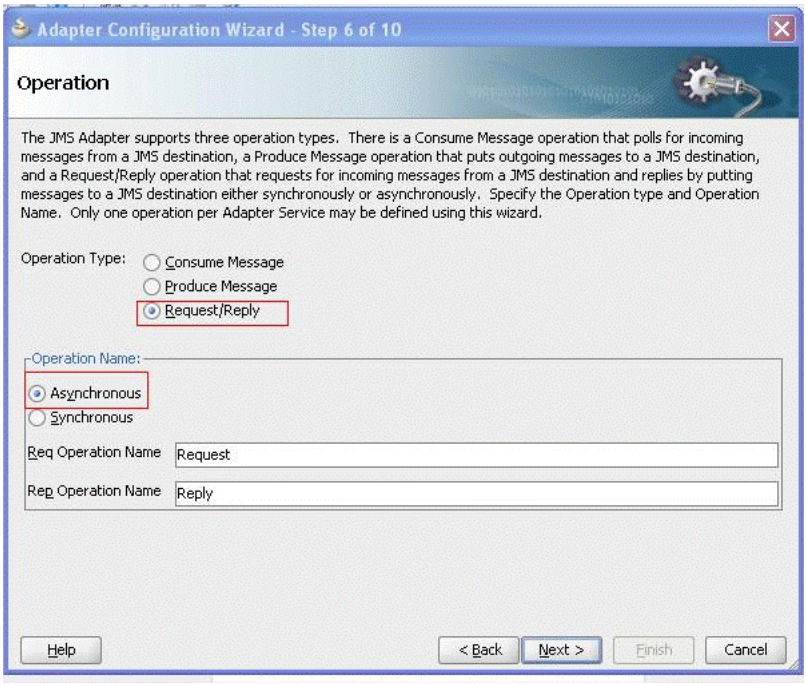
1. Drag and drop a JMSAdapter onto the External References swim lane in the JDev composite editor..

Figure 8–26 Dragging and Dropping a JMS Adapter into External References Swimlane



2. Enter default values for the first few screens in the JMS Configuration Adapter wizard until you reach the screen where the JMS Configuration Adapter wizard prompts you to enter the operation name. Select Request-Reply as the "Operation Type" and Asynchronous as "Operation Name".

Figure 8–27 Operations Screen for Request/Reply



3. Select the Request and Reply queues in the following screens of the wizard. The message is enqueued in the "Request" queue and the reply is returned in the "Reply" queue.

Figure 8–28 The Request Operation Parameters Screen

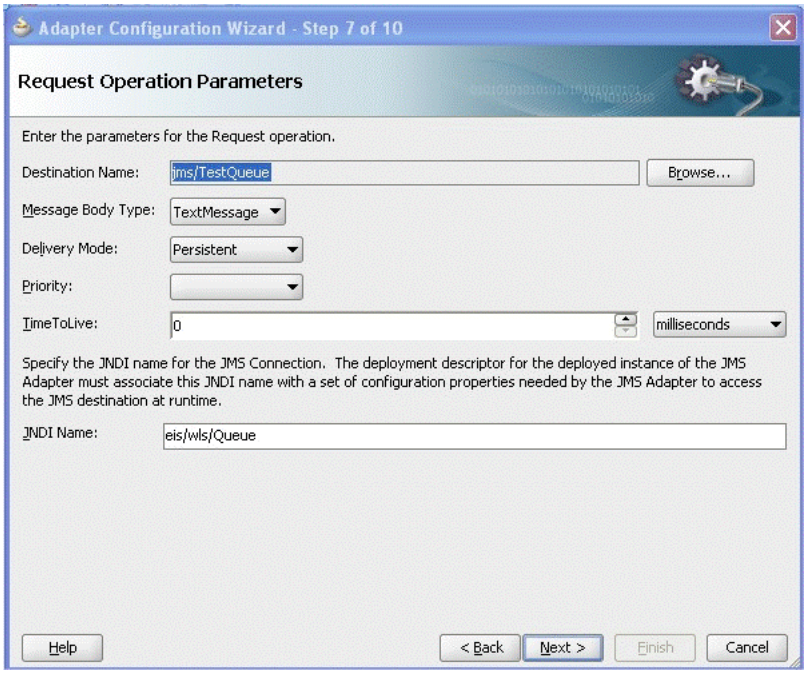
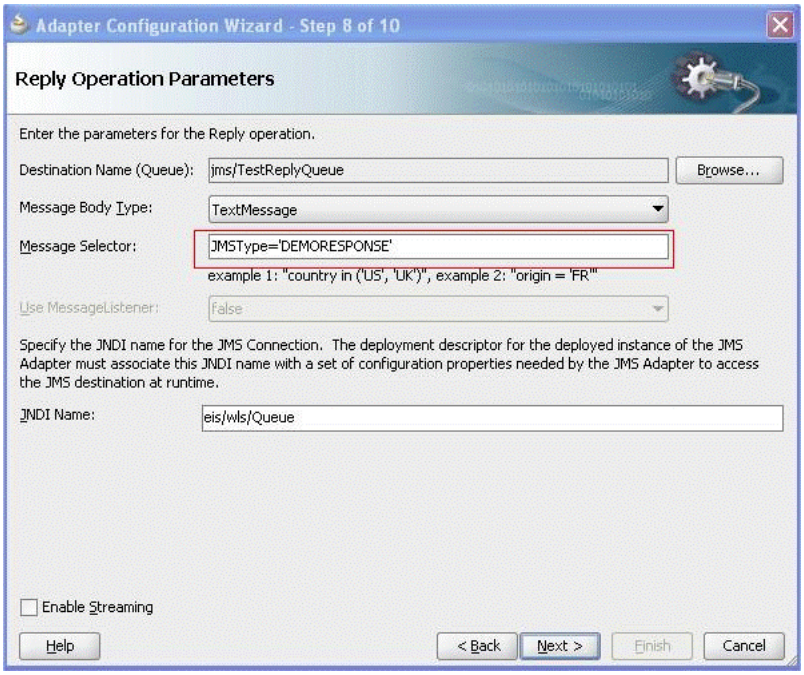


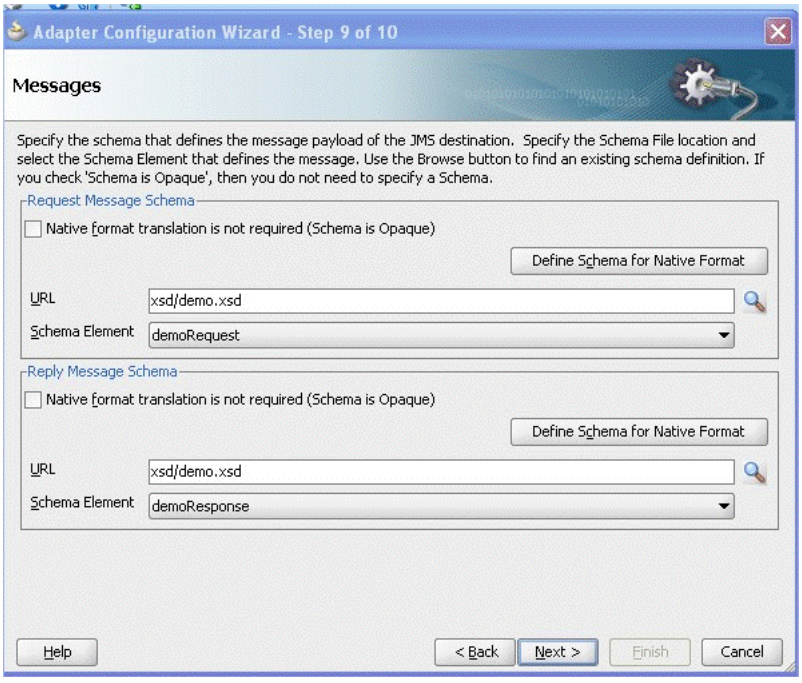
Figure 8–29 Reply Operation Parameters Screen



The reason we have used such a selector is that the back-end system that reads from the request queue and generates the response in the response queue actually generates more than one response and hence we must use a filter to exclude the unwanted responses.

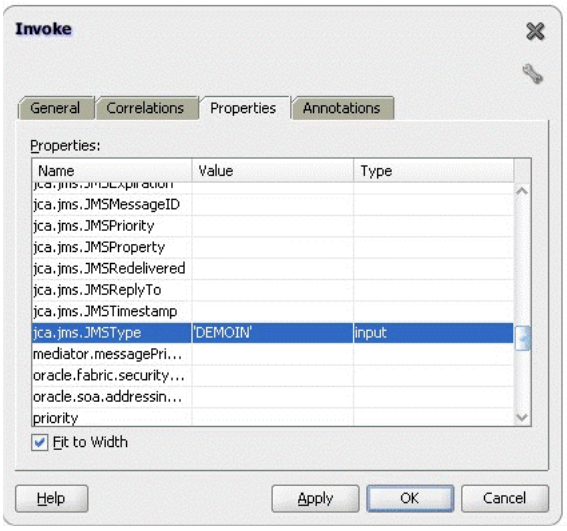
4. Select the message schema for and a response.

Figure 8–30 Selecting Message Schema for Request and for Response



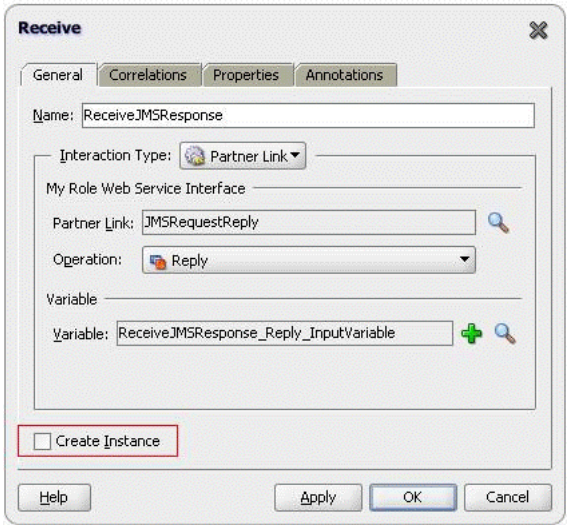
5. Add an <invoke> activity in BPEL corresponding to the JMS Adapter partner link. An additional header is set as the third-party application we use requires this.

Figure 8–31 Invoke BPEL Properties Dialog Corresponding to the JMS Adapter Link



Add a <receive> activity just after the <invoke> activity, and select the Reply operation. Please ensure that the Create Instance option is unchecked.

Figure 8–32 Receive Dialog for Reply Operation



9 Oracle JCA Adapter for Database

This chapter describes the Oracle JCA Adapter for Database (Oracle Database Adapter), which works with Oracle BPEL Process Manager and Oracle Mediator (Mediator). This chapter also includes support for stored procedures and functions (for Oracle databases only). In addition, it contains references to use cases for the Oracle Database Adapter and for stored procedures.

This chapter includes the following topics:

- Section 9.1, "Introduction to the Oracle Database Adapter"
- Section 9.2, "Complete Walkthrough of the Adapter Configuration Wizard"
- Section 9.3, "Oracle Database Adapter Features"
- Section 9.4, "Oracle Database Adapter Concepts"
- Section 9.5, "Deployment"
- Section 9.6, "JDBC Driver and Database Connection Configuration"
- Section 9.7, "Stored Procedure and Function Support"
- Section 9.8, "Oracle Database Adapter Use Cases"

9.1 Introduction to the Oracle Database Adapter

The Oracle Database Adapter enables a BPEL process to communicate with Oracle databases or third party databases through JDBC. The Oracle Database Adapter service is defined within a BPEL process partner link by using the Adapter Configuration Wizard of Oracle BPEL Process Manager (Oracle BPEL PM).

This section includes the following topics:

- Section 9.1.1, "Functional Overview"
- Section 9.1.2, "Design Overview"

9.1.1 Functional Overview

This section provides a functional overview of the Oracle Database Adapter. The Oracle Database Adapter enables Oracle SOA Suite and Oracle Fusion Middleware to communicate with database end points. These include Oracle database servers and any relational databases that follow the ANSI SQL standard and which provide JDBC drivers.

The principle of the tables and views in the Oracle Database Adapter is to expose to SOA tables and SQL as transparently and non-intrusively as possible. From an integration standpoint, tables and SQL are what relational database products have in

common, so a generic solution focused on what is standard has the greatest reach. In exposing databases to SOA, it is also about combining the technologies of SQL and XML, the former an ideal language for querying information, the latter an ideal format for transporting and representing information. While stored procedure support is less standard across databases, Oracle Database Adapter provides support for stored procedures as the guide describes.

The Oracle Database Adapter is a JCA 1.5 connector, which runs on the Oracle Application Server. It relies on an underlying JDBC connector/driver to enact the database communication. In contrast to JDBC, it is non-programmatic. The interaction (series of `SELECT`, `UPDATE`, `INSERT`) is loosely modeled using the Adapter Configuration Wizard. The inputs/outputs are XML, most easily seen as input parameters and result sets converted to XML. These XML inputs and outputs allow the Oracle Database Adapter services to be plugged into Oracle Fusion Middleware.

To access an existing relational schema, you must create an application and an SOA project to use the Adapter Configuration Wizard to perform the following:

- Import a relational schema (one or more related tables) and map it as an XML schema (XSD)

  For more information, see Section 9.4.1, "Relational-to-XML Mapping."

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

  For more information, see Section 9.4.2, "SQL Operations as Web Services."

- Have database events initiate an Oracle Fusion Middleware process.

The Oracle Database Adapter can currently be used only within the context of an SOA process as Section 9.1.1.1, "Oracle Database Adapter Integration with Oracle BPEL PM" describes.

Although Oracle Streams Advanced Queuing (Oracle AQ) is an Oracle Database feature, you use the separate, specialized Oracle JCA Adapter for AQ to integrate with Oracle AQ. For more information, see Chapter 7, "Oracle JCA Adapter for AQ".

For non-relational and legacy systems (with a few exceptions such as DB2 on AS/400), application and mainframe adapters are available. For more information about application and mainframe adapters, see:

- Section 1.2.2, "Legacy Adapters"
- Section 1.2.3, "Packaged-Application Adapters"
- Section 1.2.4, "Oracle Adapter for Oracle Applications"

For more information on the Oracle Database Adapter, see:

- Section 9.3, "Oracle Database Adapter Features"
- Section 9.4, "Oracle Database Adapter Concepts"
- Section 9.2, "Complete Walkthrough of the Adapter Configuration Wizard"

9.1.1.1 Oracle Database Adapter Integration with Oracle BPEL PM

When the Oracle Database Adapter is used to poll for database events (usually an `INSERT` operation on an input table) and initiate a process, in a Mediator component or an SOA composite it is called an exposed service. In Oracle BPEL process it is a partner link tied to a Receive activity. The expression `inbound` (from database into SOA) is commonly used.

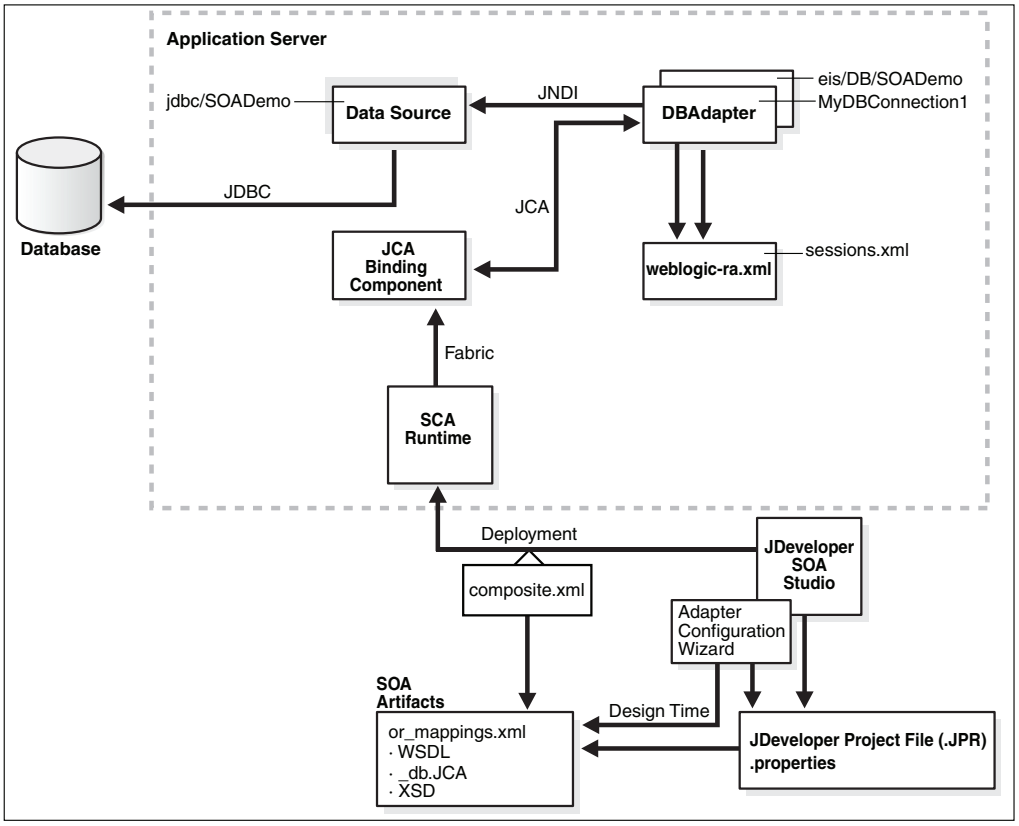
When the Oracle Database Adapter is used to invoke a one-time DML statement such as `INSERT` or `SELECT`, in a Mediator component or an SOA composite, it is called a

service reference. In Oracle BPEL process, it is a partner link tied to an Invoke activity. The expression `outbound` (from SOA out to the database) is used.

9.1.2 Design Overview

This section provides an overview of the design of the Oracle Database Adapter. Figure 9–1 shows how the Oracle Database Adapter interacts with the various design-time and deployment artifacts.

Figure 9–1 How the Oracle Database Adapter Works



The Oracle Database Adapter is a JCA 1.5 connector, which is deployed to the application server during installation.

The Oracle Database Adapter consists of multiple instances; each instance represents a connection to a database end point. Different SOA processes may point to the same adapter instance (database), while different service endpoints in a SOA process may point to different adapter instances (databases).

Because each adapter instance points to a single database, there is a one-to-one correspondence from adapter instances to application server data sources. Out of the box there is a single Oracle Database Adapter instance named `eis/DB/SOADemo`, which points to the data source `jdbc/SOADataSource`.

The list of adapter instances is stored in a deployment descriptor file, `weblogic-ra.xml` on Oracle WebLogic Server. (It is inside of `DbAdapter.rar`, which contains also the Java class files in `DBAdapter.jar`). Configuring an Oracle Database Adapter instance is more about creating the underlying data source: getting the correct JDBC driver and connection URL.

For more information, see Section 9.6, "JDBC Driver and Database Connection Configuration."

However `weblogic-ra.xml` entries occasionally have more than simply the name of the underlying data source. These properties are detailed further under Section 9.5, "Deployment".

While at run time you have Oracle Database Adapter instances, at design time you have the Adapter Configuration Wizard (link). You can run it once to generate a single adapter service end point, and then multiple times in edit mode to make incremental changes to each. It generates all the adapter related artifacts needed when deploying a SOA composite as Table 9–1 lists.

Table 9–1 Adapter Configuration Wizard Generated SOA Composite Adapter Artifacts

File	Description
`<serviceName>.wsdl`	This is an abstract WSDL, which defines the service end point in terms of the name of the operations and the input and output XML elements.
`<serviceName>_table.xsd`	This contains the XML file schema for these input and output XML elements. Both these files form the interface to the rest of the SOA project.
`<serviceName>_or-mappings.xml`	This is an internal file. It is a TopLink specific file, which is used to describe the mapping between a relational schema and the XML schema. It is used at run time.
`<serviceName>_db.jca`	This contains the internal implementation details of the abstract WSDL. It has two main sections, location and operations. Location is the JNDI name of an adapter instance, that is, `eis/DB/SOADemo`. Operations describe the action to take against that end point, such as `INSERT`, `UPDATE`, `SELECT`, and `POLL`. The contents of the `db.jca` file are wholly determined by choices made while running the Adapter Configuration Wizard.
`<serviceName>.properties`	This is also an internal file. It is created when tables are imported, and information about them is saved. It is used only at design time. At run time, the location is used to look up the adapter instance which executes the service. Based on the properties in the `db.jca` file and the linked `or-mappings.xml` file, `<seviceName>.properties` file generates the correct SQL to execute, parses the input XML, and builds an output XML file matching the XSD file. To execute the SQL, it obtains a pooled SQL connection from the underlying data source.

9.2 Complete Walkthrough of the Adapter Configuration Wizard

This section describes the Adapter Configuration Wizard and how you can define an Oracle Database Adapter by using the Adapter Configuration Wizard.

This section describes the various Oracle Database Adapter concepts through a use case, which is, a complete walkthrough of the Adapter Configuration Wizard. In addition, this use case also describes how by using the Adapter Configuration Wizard, you can import tables from the database, specify relationships spanning multiple tables, generate corresponding XML schema definitions, and create services to expose the necessary SQL or database operations. These services are consumed to define partner links that are used in the BPEL process. You use the Adapter Configuration Wizard to both create and edit adapter services.

- Section 9.2.1, "Creating an Application and an SOA Project"
- Section 9.2.2, "Defining an Oracle Database Adapter"
- Section 9.2.3, "Connecting to a Database"

- Section 9.2.4, "Selecting the Operation Type"
- Section 9.2.5, "Selecting and Importing Tables"
- Section 9.2.6, "Defining Primary Keys"
- Section 9.2.7, "Creating Relationships"
- Section 9.2.8, "Creating the Attribute Filter"
- Section 9.2.9, "Defining a WHERE Clause"
- Section 9.2.10, "Choosing an After-Read Strategy"
- Section 9.2.11, "Specifying Polling Options"
- Section 9.2.12, "Specifying Advanced Options"
- Section 9.2.13, "Entering the SQL String for the Pure SQL Operation"

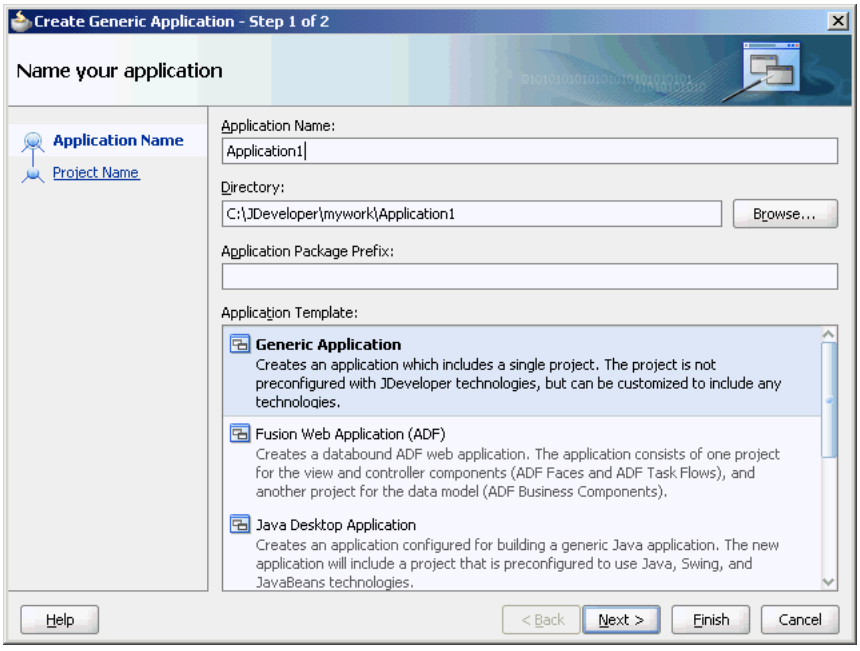
9.2.1 Creating an Application and an SOA Project

You must create an Oracle JDeveloper (JDeveloper) application to contain the SOA composite. Perform the following steps to create an application, and an SOA project:

1. Open JDeveloper.
2. In the Application Navigator, click **New Application**.

   The Create Generic Application - Name your application page is displayed, as shown in Figure 9–2.
3. Enter a name for the application in the **Application Name** field.
4. In the Application Template list, choose **Generic Application**.

Figure 9–2 The Create Generic Application - Name your application Page

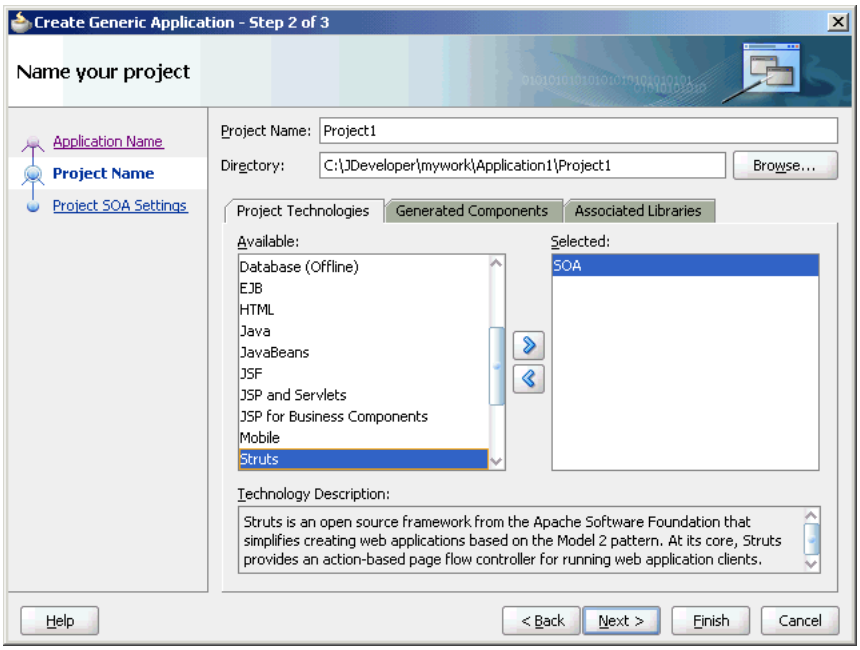


5. Click **Next**.

   The Create Generic Application - Name your project page is displayed, as shown in Figure 9–3.

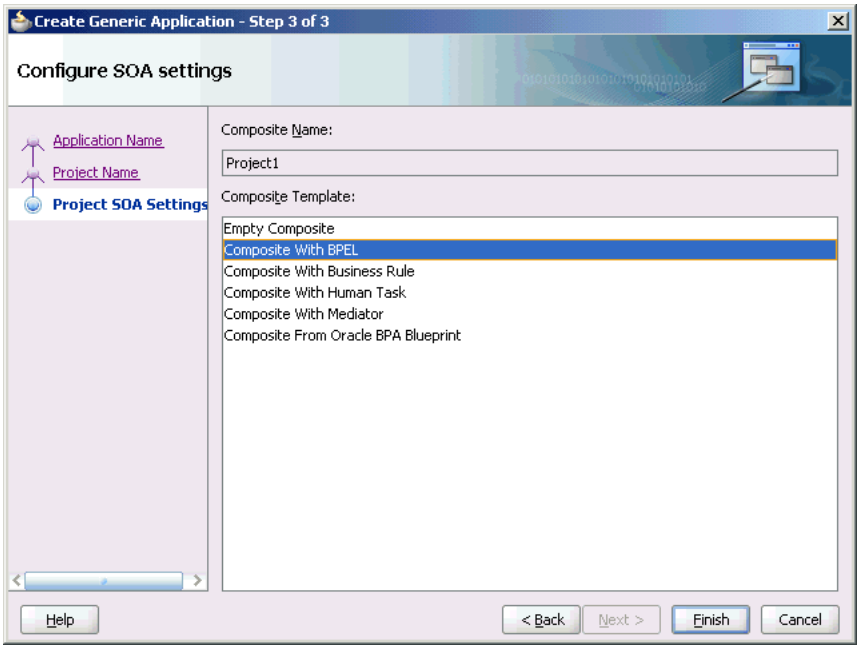
6. In the **Project Name** field, enter a descriptive name.
7. In the **Available** list in the **Project Technologies** tab, double-click **SOA** to move it to the Selected list.

Figure 9–3 The Create Generic Application - Name your Generic project Page



8. Click **Next**. The Create Generic Application - Configure SOA settings page is displayed, as shown in Figure 9–4.

Figure 9–4 The Create Generic Application - Configure SOA settings Page

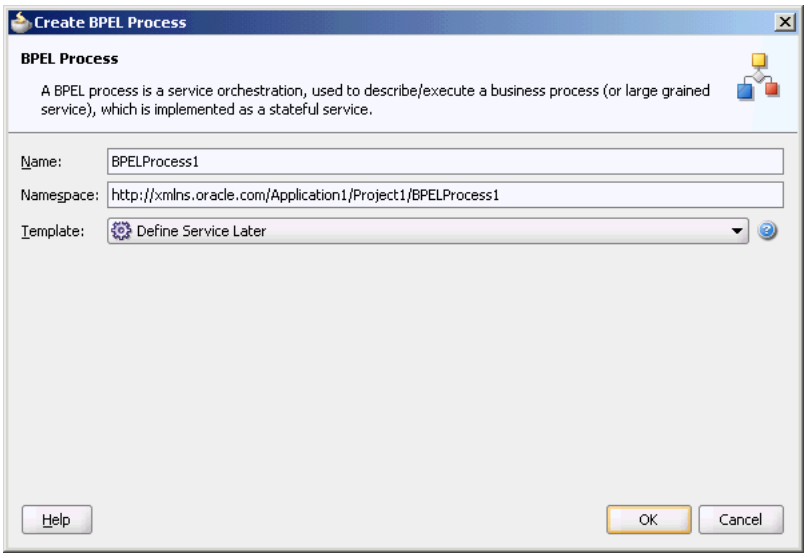


9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

You have created a new application and an SOA project. This automatically creates an SOA composite.

The Create BPEL Process page is displayed, as shown in Figure 9–5.

Figure 9–5 The Create BPEL Process Page



10. Enter a name for the BPEL process in the **Name** field.
11. Select **Define Service Later** in the Template list, and then click **OK**.

    You have created a BPEL process.

9.2.2 Defining an Oracle Database Adapter

The next step is to define an Oracle Database Adapter service. Perform the following steps to create an Oracle Database Adapter service:

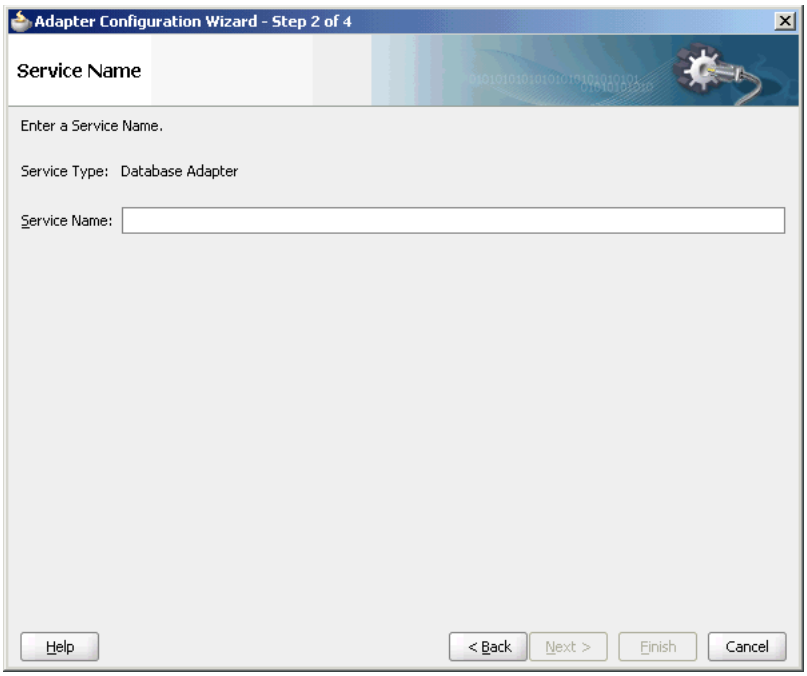
1. In the Component Palette, select **SOA**.
2. Drag and drop **Database Adapter** from the Service Adapters list to the Exposed components swim lane in the composite.xml page.

   The Adapter Configuration Wizard is displayed.

   > **Note:** To create an Oracle Database Adapter service as part of a BPEL process, drag and drop a BPEL process from Service Components onto Components. Double-click it. Then, in the BPEL Component Palette, drag and drop Database Adapter from BPEL Services onto a Partner Links swim lanes.

3. Click **Next**. The Service Name page is displayed, as shown in Figure 9–6. Enter the following information:

Figure 9–6 Specifying the Service Name



4. In the **Service Name** field, enter a service name, and then click **Next**. The Service Connection page is displayed.

See Section 9.2.3, "Connecting to a Database" to continue using the Adapter Configuration Wizard.

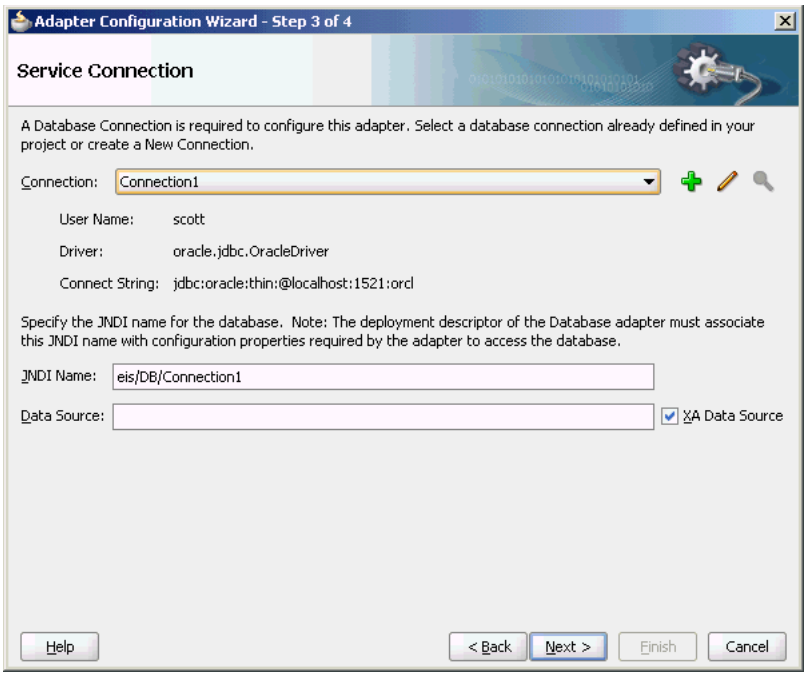
9.2.3 Connecting to a Database

Figure 9–7 shows where you select the database connection that you are using with the service. This is the database from which you import tables to configure the service. This is the database from which you import tables to configure the service. You can re-create it here in each new JDeveloper application you create.

You can provide a Java Naming and Directory Interface (JNDI) name to identify the database connection, as the default name that is provided is `eis/DB/<ConnectionNameInJDev>`.

For more information, see Section 9.5, "Deployment."

Figure 9–7 The Adapter Configuration Wizard: Service Connection Page



Note the following:

- In production environments, it is recommended that you add the JNDI entry to the adapter deployment descriptor (`weblogic-ra.xml`). This way, the Oracle Database Adapter is more performant by working in a managed mode.

  For information about creating a data source and an outbound connection pool, see Section 2.18, "Adding an Adapter Connection Factory."

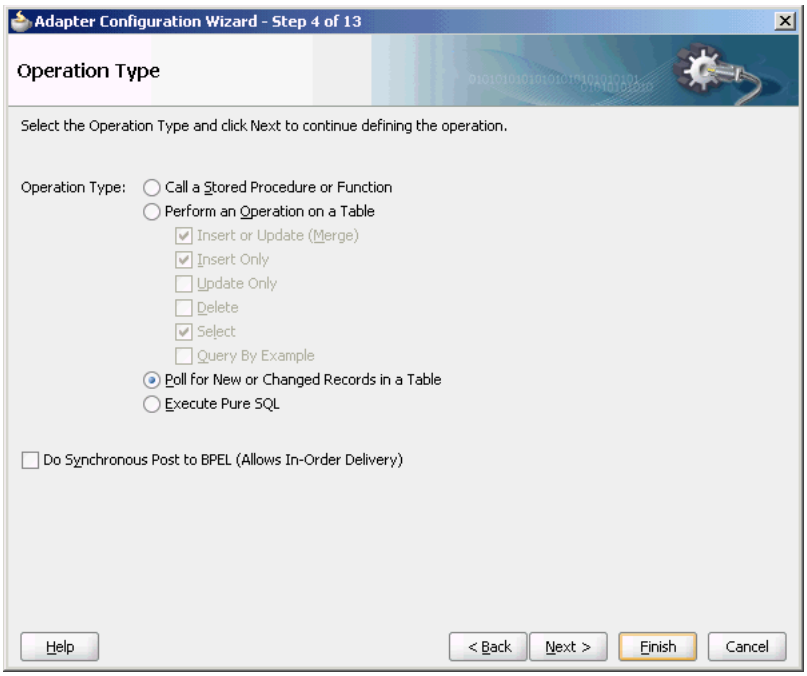
- When you click **Next**, a connection to the database is attempted. If a connection cannot be made, you cannot proceed to the next window, even if you are editing an existing partner link.

See Section 9.2.4, "Selecting the Operation Type" to continue using the Adapter Configuration Wizard.

9.2.4 Selecting the Operation Type

Figure 9–8 shows where you indicate the type of operation you want to configure for this service.

Figure 9–8 The Adapter Configuration Wizard: Operation Type Page



The following operation types are available:

- **Call a Stored Procedure or Function**

  Select this option if you want the service to execute a stored procedure or function. For more information, see Section 9.7, "Stored Procedure and Function Support."

- **Perform an Operation on a Table**

  Select this option for outbound operations. You can select **Insert or Update**, **Insert Only**, **Update Only**, **Delete**, **Select**, or any combination of the six. These operations loosely translate to SQL `MERGE`, `INSERT`, `UPDATE`, `DELETE`, and `SELECT` operations.

  For more information, see Section 9.4.2.1, "DML Operations."

  > **Note:** The operation `Update Only` sometimes performs inserts/deletes for child records. That is, an update to Master could involve a new or deleted detail. So if the input to update contains only one detail record, then the other detail records in the table are deleted.

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

  Select this option for an inbound operation (that is, an operation that is associated with a Receive activity). This operation type polls a specified table and returns for processing any new rows that are added. You can also specify the polling frequency.

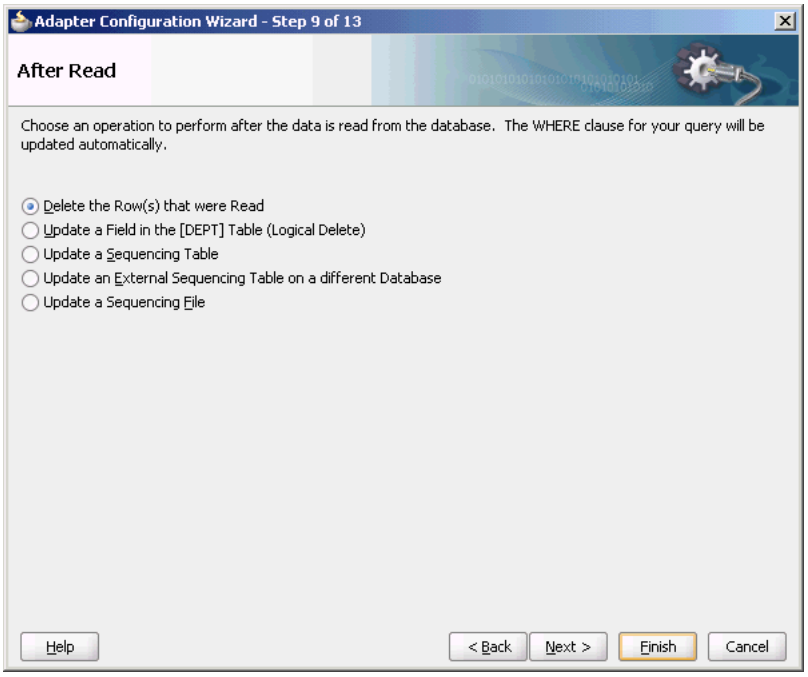
  For more information, see Section 9.4.2.2, "Polling Strategies."

  The following is a list of polling operations that you can perform after the data is read from the database, as shown in Figure 9–9:

  – Delete the Row(s) that were Read

- Update a Field in the [Table_Name] Table (Logical Delete)
- Update a Sequencing Table
- Update an External Sequencing Table on a Different Database
- Control Table Strategy

Figure 9–9 Polling Operations



- **Execute Pure SQL**

  Useful when dealing with arbitrarily complex statements, aggregate queries (result is not row-based), and `XMLType` columns. See Section 9.3.2, "Pure SQL - XML Type Support" to follow this usage of the Adapter Configuration Wizard.

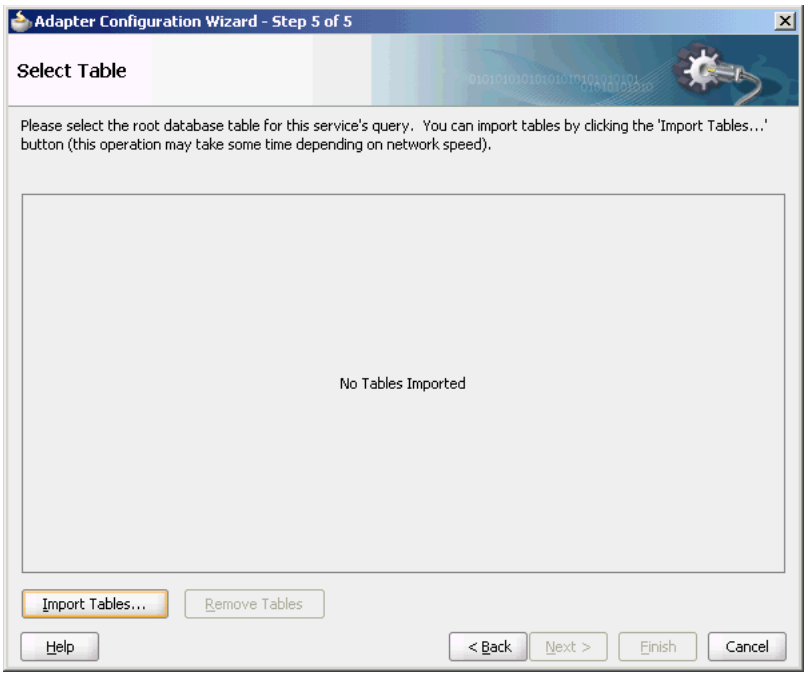
  Note: Schema Bound XML tables are not supported.

Otherwise, see Section 9.2.5, "Selecting and Importing Tables" to continue using the Adapter Configuration Wizard.

9.2.5 Selecting and Importing Tables

Figure 9–10 shows where you select the root database table for your operation. If you are using multiple related tables, then this is the highest-level table (or highest parent table) in the relationship tree.

Figure 9–10 The Adapter Configuration Wizard: Select Table



Selecting Import Tables launches a sub-wizard, which lets you search for and select multiple tables to import from the database. Removing a table removes (or undoes) any relationships on related tables that remain. If any underlying tables have changed when running this wizard in edit mode, you get a warning showing you what changes have occurred. To reconcile, import the tables again. If you click **Import Tables** and select multiple tables, then relationships between these tables are inferred based on the foreign key constraints. However if you launch **Import Tables** once for each table imported, then no relationships are inferred.

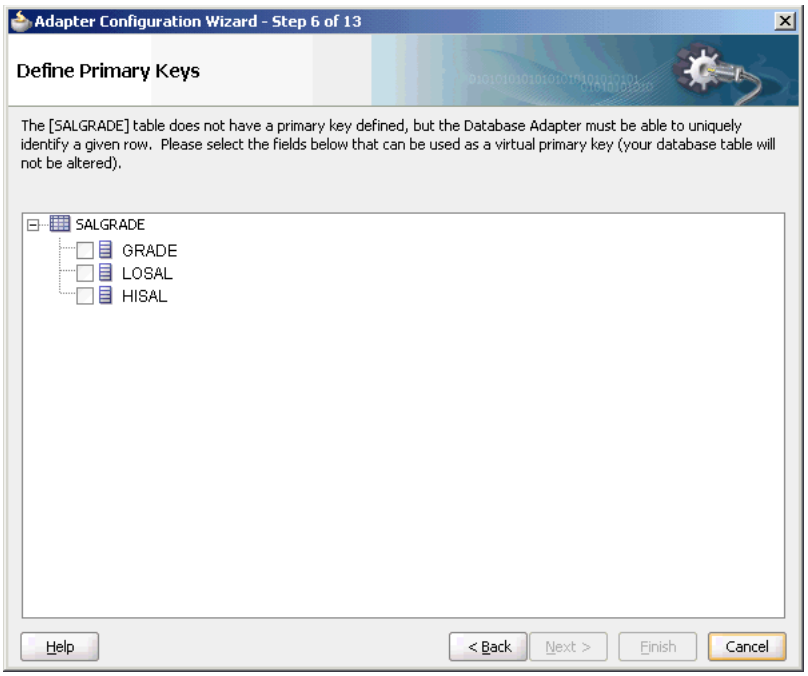
Note: If you reimport a table, you lose any custom relationships you may have defined on that table and any custom `WHERE` clauses (if the table being imported was the root table).

See Section 9.2.6, "Defining Primary Keys" to continue using the Adapter Configuration Wizard.

9.2.6 Defining Primary Keys

If any of the tables you have imported do not have primary keys defined on the database, you are prompted to provide a primary key for each one, as shown in Figure 9–11. You must specify a primary key for all imported tables. You can select multiple fields to specify a multipart primary key.

Figure 9–11 The Adapter Configuration Wizard: Define Primary Keys Page



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

See Section 9.2.7, "Creating Relationships" to continue using the Adapter Configuration Wizard.

Note: Note that the Oracle Database Adapter only supports tables where there is a primary key defined. If primary key constraints have not been defined on a table explicitly, then you must provide one at design time while defining the Oracle Database Adapter by using the Adapter Configuration Wizard. If you do not provide a valid primary key, then the unique constraint is not guaranteed, and this could result in possible loss of messages at run time. That is, rows with duplicate primary key values are likely to be lost.

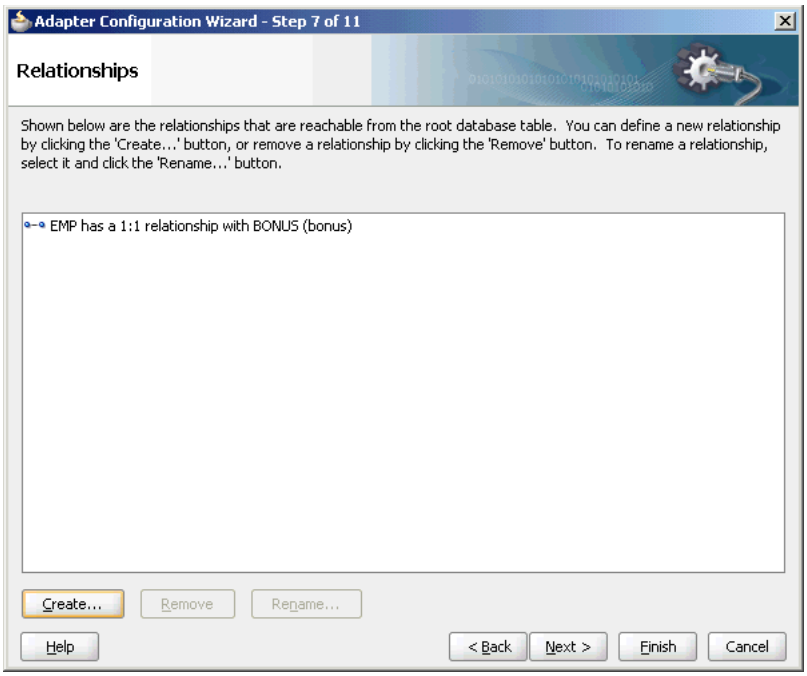
To obtain a sample that describes how to use the `row id` field as primary key, access the Oracle SOA Sample Code site.

Note: Oracle recommends that you use `varchar` instead of `char` for primary key columns, otherwise you must set the `weblogic-ra.xml` property `shouldTrimStrings` to `false`. The truncation of trailing spaces could cause the primary key to be read incorrectly, making it impossible to update read rows as processed.

9.2.7 Creating Relationships

Figure 9–12 shows the relationships defined on the root database table and any other related tables. You can click **Create Relationships…** to create a relationship between two tables, or click **Remove Relationship** to remove it. To rename a relationship, click **Rename Relationship**.

Figure 9–12 The Adapter Configuration Wizard: Relationships Page



Note the following regarding creating relationships:

- If foreign key constraints between tables exist on the database, then two relationships are created automatically when you import the tables, a one-to-one (1:1) from the source table (the table containing the foreign key constraints) to the target table, and a one-to-many (1:M) from the target table to the source table.
- As Figure 9–12 shows, you see only the relationships that are reachable from the root database table. If, after removing a relationship, other relationships are no longer reachable from the root table, then they are not shown in the Relationships window. Consider the following set of relationships:

  ```
  A --1:1--> B --1:1--> C --1:M--> D --1:1--> E --1:M--> F

  (1) (2) (3) (4) (5)
  ```

  If you remove relationship 3, then you see only:

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

  B --1:1--> C
  ```

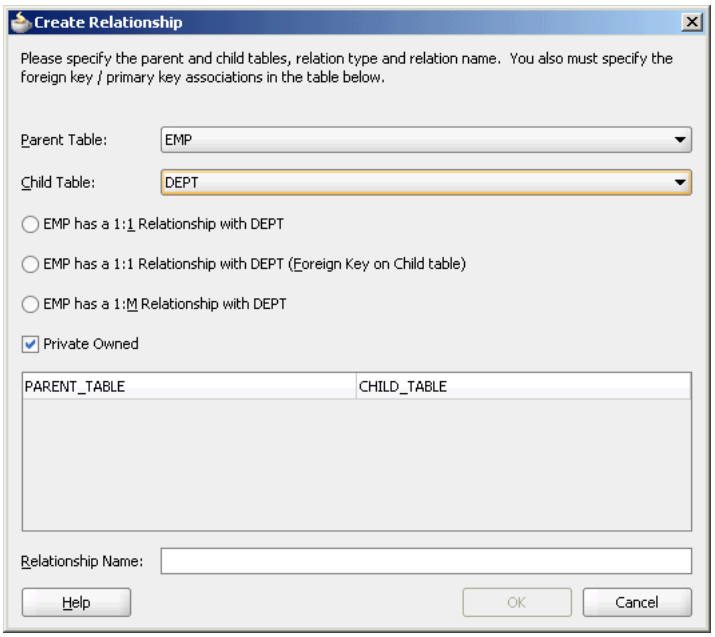
  If you remove relationship 2, then you see only:

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

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

Figure 9–13 shows where you can create a relationship.

Figure 9–13 The Create Relationship Dialog



To create a relationship:

1. Select the parent and child tables.
2. Select the mapping type (one-to-many, one-to-one, or one-to-one with the foreign key on the child table).
3. Associate the foreign key fields to the primary key fields.
4. Optionally name the relationship (a default name is generated).

> **Note:** Only tables that are reachable from the root table can be selected as a parent.

9.2.7.1 What Happens When Relationships Are Created or Removed

When tables are initially imported into the Adapter Configuration Wizard, a TopLink direct-to-field mapping corresponding to each field in the database is created. Consider the schemas shown in Figure 9–14 and Figure 9–15:

Figure 9–14 EMPLOYEE Schema

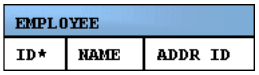


Figure 9–15 ADDRESS Schema

Immediately after importing these two tables, the following mappings in the `Employee` descriptor are created:

```
Employee:
```

- `id` (direct mapping to the `ID` field, for example, *151*)
- `name` (direct mapping to the `NAME` field, for example, *Stephen King*)
- `addrId` (direct mapping to the `ADDR_ID` field, for example, *345*)

When creating a relationship mapping, the direct-to-field mappings to the foreign key fields are removed and replaced with a single relationship (one-to-one, one-to-many) mapping. Therefore, after creating a one-to-one relationship between `Employee` and `Address` called `homeAddress`, the `Employee` descriptor appears, as shown in the following example:

`Employee:`

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

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

9.2.7.2 Different Types of One-to-One Mappings

When relationships are auto created, the one-to-many relationship is from the table without the foreign key. However, you can declare this mapping, which is technically 1-many, as a 1-1. For that, choose 1-1 (foreign key on target).

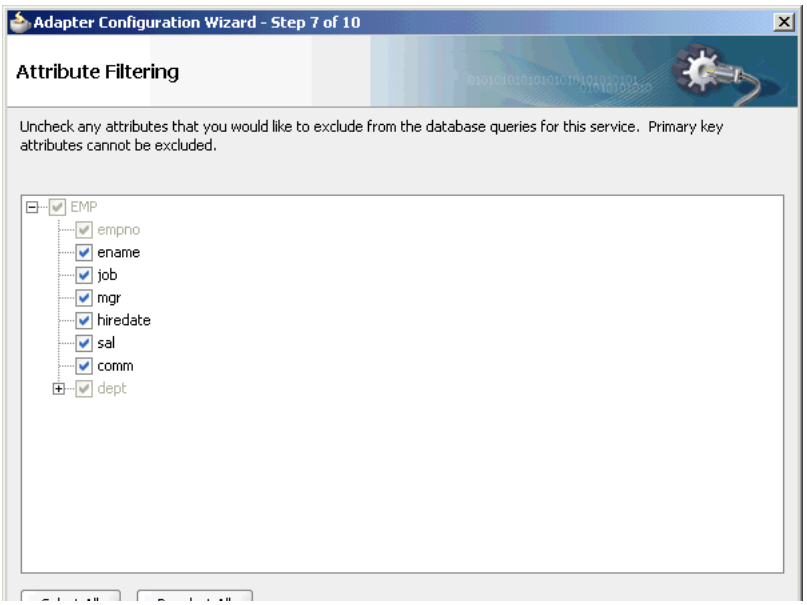
9.2.7.3 When Foreign Keys Are Primary Keys

Not all tables imported are in the third normal form (3NF). In rare cases, you may have two or more tables which share the same primary key but no separate foreign key columns exist. It is recommended to create 1-1 (foreign key on target) relationships from the root table to all related tables. The reason is two fold. First, if you were to declare the primary key on the root as a foreign key (1-1, foreign key on source), then that mapping would be removed, so you would not see the primary key in the root record, only in the child record. Second, a foreign key can only point to a single table. Once you declare a column to be part of a foreign key, it is removed, so it cannot be used again in a new relationship. Creating a 1-1 (foreign key on source) on the root table not only makes the primary key column disappear but prevents you from joining the root table to the remaining tables.

9.2.8 Creating the Attribute Filter

Figure 9–16 shows the attribute filter that is created from the imported table definitions, including any relationships that you may have defined.

Figure 9–16 The Adapter Configuration Wizard: Attribute Filtering Page



If your object filter contains self-relationships (for example, the employee-to-employee manager relationship), then you see these as loops in the tree. These loops are not present in the XSD file. This is the descriptor object model, not the XSD file.

In this page, you select those columns that appear in the XML file, whether for input (`MERGE`, `INSERT`) or output (`SELECT`). Columns you are not interested in or which are to be read-only (should not be modified) can be deselected here.

See Section 9.2.9, "Defining a WHERE Clause" to continue using the Adapter Configuration Wizard.

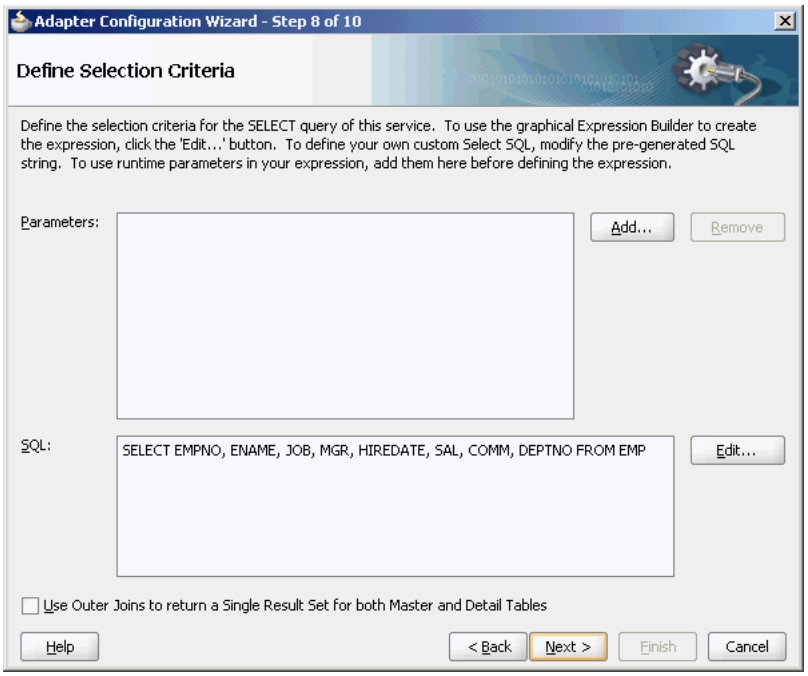
9.2.9 Defining a WHERE Clause

If your service contains a `SELECT` query (that is, inbound polling services, or outbound services that contain a `SELECT`), then you can customize the `WHERE` clause of the `SELECT` statement.

> **Note:** When using polling with `Sequencing Table/Update an External Sequencing Table`, ensure that the name of the table in the `SELECT` query matches the case of the data in the sequencing table.

Figure 9–17 shows where you define a `WHERE` clause for an outbound service.

Figure 9–17 The Adapter Configuration Wizard: Define Selection Criteria Page



Note: The `WHERE` clause applies to `SELECT` operations only (that is, polling for new or changed records or performing a `SELECT` operation on a table). It does not apply to `INSERT`, `UPDATE`, and `DELETE` operations.

The most basic expression in a `WHERE` clause can be one of the following three cases, depending on what the right-hand side (RHS) is:

1. `EMP.ID = 123`

   In this case, the RHS is a literal value. This RHS is the **Literal** option shown in Figure 9–18.

2. `EMP.ADDR_ID = ADDR.ID`

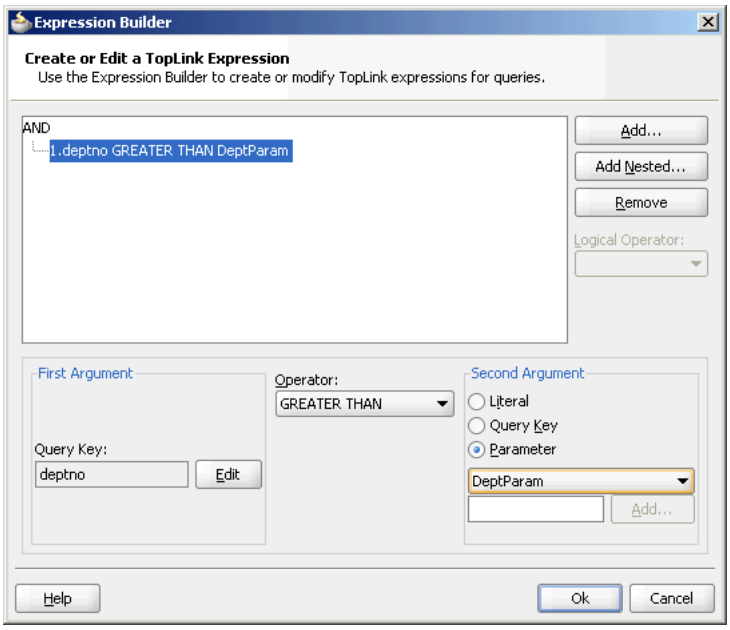
   In this case, the RHS is another database field. This RHS is the Query Key option shown in Figure 9–18.

3. `EMP.ID = ?`

   In this case, the RHS value must be specified at run time. This is the Parameter option shown in Figure 9–18.

You can 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 9–18.

Figure 9–18 Expression Builder



To model more complex `WHERE` clauses (sub selects and functions), and to add `ORDER BY` clauses, you can edit the SQL procedure manually and click **Next**. However, this creates maintenance overhead later on, due to hard-coded `SQL`, and you may lose platform independence.

You can change the columns listed in the `FROM` clause when the number of columns and the types of each remain unchanged. For more complex changes consider using the Execute Pure SQL option directly where you can type any `SQL`.

Return Single Result Set

You must select **Use Outer Joins to return a Single Result Set for both Master and Detail Tables** in the Define Selection Criteria page to use an advanced feature that influences how many total statements `TopLink` uses when querying against multiple related tables. The safest method is to use the default (1 per table), and this feature attempts 1 total, by outer joining all related tables into a single result set.

See Section 9.2.10, "Choosing an After-Read Strategy" to continue using the Adapter Configuration Wizard.

9.2.10 Choosing an After-Read Strategy

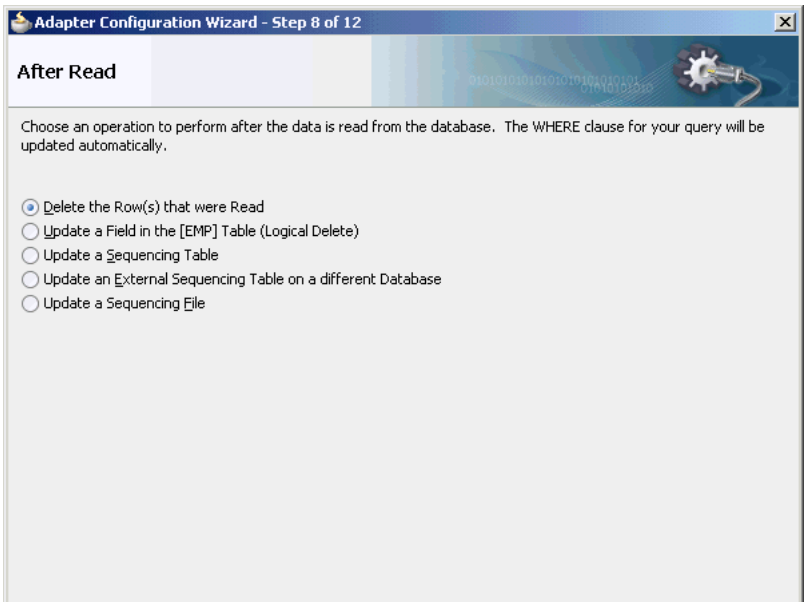
If you selected Perform an Operation on a Table, then you can skip ahead to the Section 9.2.12, "Specifying Advanced Options.".

When configuring an inbound operation, you have the following options about what to do after a row or rows have been read:

- Section 9.2.10.1, "Delete the Rows That Were Read"
- Section 9.2.10.2, "Update a Field in the Table (Logical Delete)"
- Section 9.2.10.3, "Update a Sequencing Table"
- Section 9.2.10.4, "Update an External Sequencing Table on a Different Database"
- Section 9.2.10.5, "Update a Sequencing File"

Figure 9–19 shows these options.

Figure 9–19 The Adapter Configuration Wizard: After Read Page



See Section 9.4.2.2, "Polling Strategies" to continue using the Adapter Configuration Wizard.

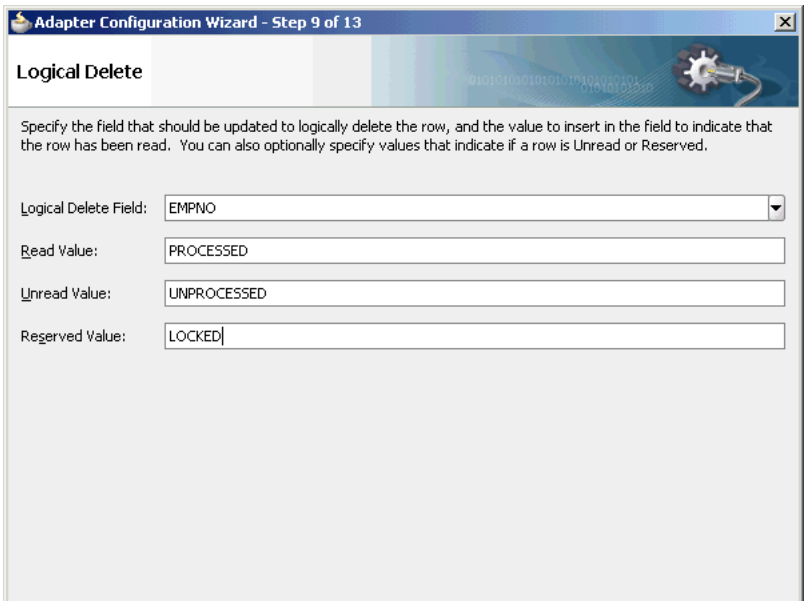
9.2.10.1 Delete the Rows That Were Read

With this option, the rows are deleted from the database after they have been read and processed by the adapter service.

9.2.10.2 Update a Field in the Table (Logical Delete)

With this option, you update a field in the root database table to indicate that the rows have been read. The `WHERE` clause of the query is updated automatically after you complete the configuration, as shown in Figure 9–20.

Figure 9–20 The Adapter Configuration Wizard: Logical Delete Page



When you use this approach, your database table appears, as shown in Figure 9–21.

Figure 9–21 Updating Fields in a Table

EMPLOYEE				
ID*	NAME	SALARY	...	STATUS
150	Dean Koontz	55000	...	PROCESSED
151	Stephen King	75000	...	
152	Patricia Cornwell	58000	...	UNPROCESSED
153	John Grisham	67500	...	PROCESSED
154	Michael Crichton	61250	...	LOCKED

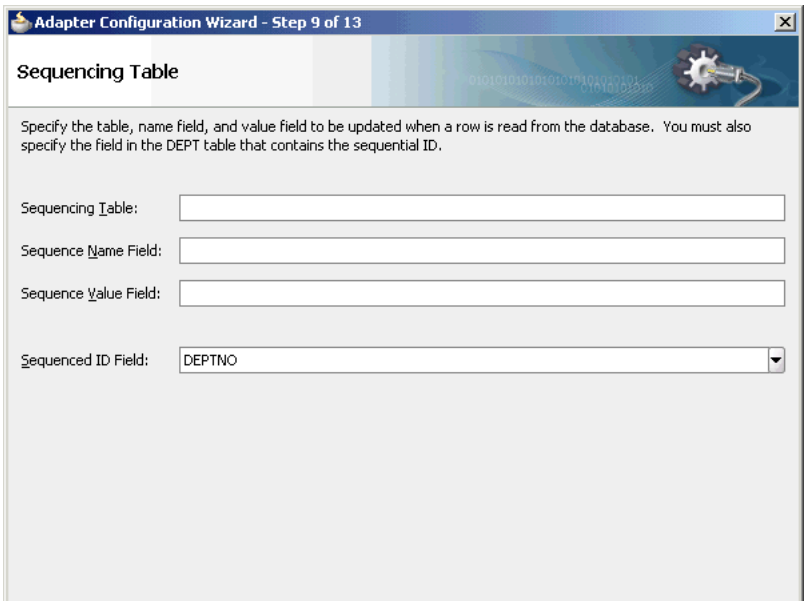
Note the following:

- Rows 150 and 153 have been previously read and processed.
- At the next polling event, row 152 is read and processed because it contains `UNPROCESSED` in the Status column. Because an explicit `Unread Value` was provided, row 151 is not read.
- Row 154 has been flagged as `LOCKED` and is not read. You can use this reserved value if your table is used by other processes.

9.2.10.3 Update a Sequencing Table

With this option, you are keeping track of the last-read rows in a separate sequence table. Figure 9–22 shows the information you provide. The `WHERE` clause of your query is updated automatically after you complete the configuration.

Figure 9–22 The Adapter Configuration Wizard: Sequencing Table Page



When you use these settings, your sequence table appears, as shown in Figure 9–23.

Figure 9–23 Updating a Sequence Table

MY_SEQUENCE	
SEQ_NAME	SEQ_VALUE
EMPLOYEE	154

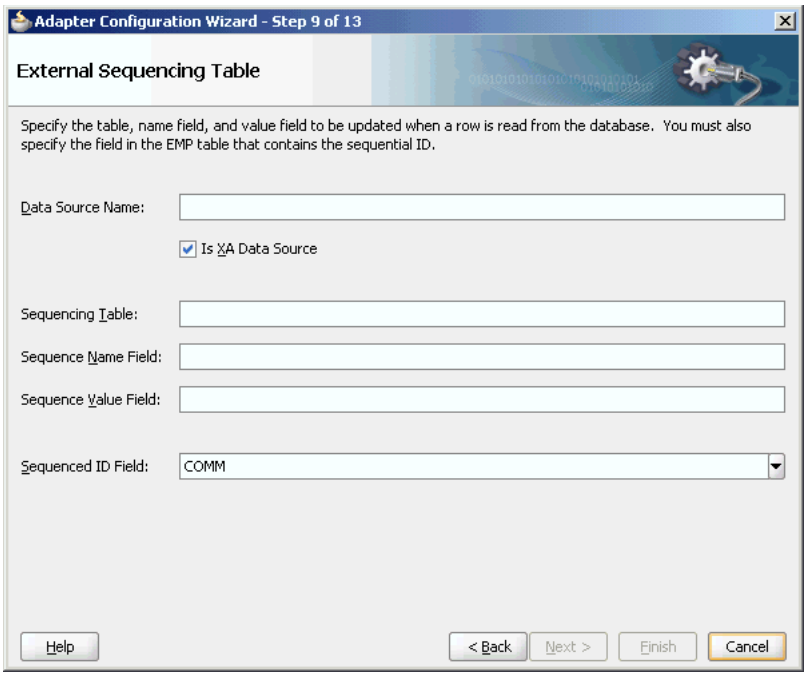
Whenever a row is read, this table is updated with the ID that was just read. Then, when the next polling event occurs, it searches for rows that have an ID greater than the last-read ID (154).

Typical columns used are `event_id`, `transaction_id`, `scn` (system change number), `id`, or `last_updated`. These columns typically have (monotonically) increasing values, populated from a sequence number or `sysdate`.

9.2.10.4 Update an External Sequencing Table on a Different Database

Choose this operation to employ the sequencing table: last updated strategy. Figure 9–24 shows the Adapter Configuration Wizard - External Sequencing Table page in which you specify the details required to perform this operation.

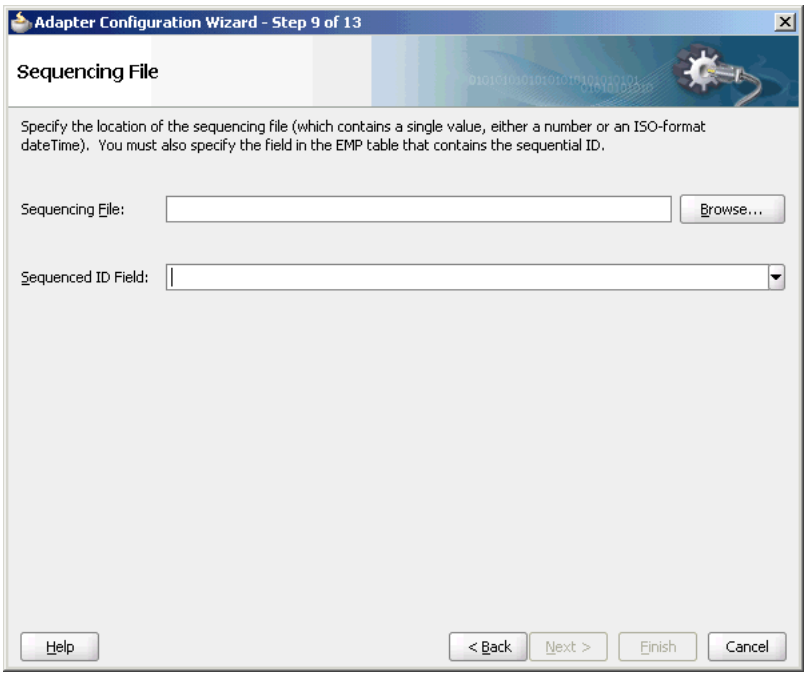
Figure 9–24 The Adapter Configuration Wizard - External Sequencing Table page



9.2.10.5 Update a Sequencing File

Use this option to update a sequencing file. Figure 9–25 shows the Adapter Configuration Wizard - Update a Sequencing File page where you specify the details for performing this operation.

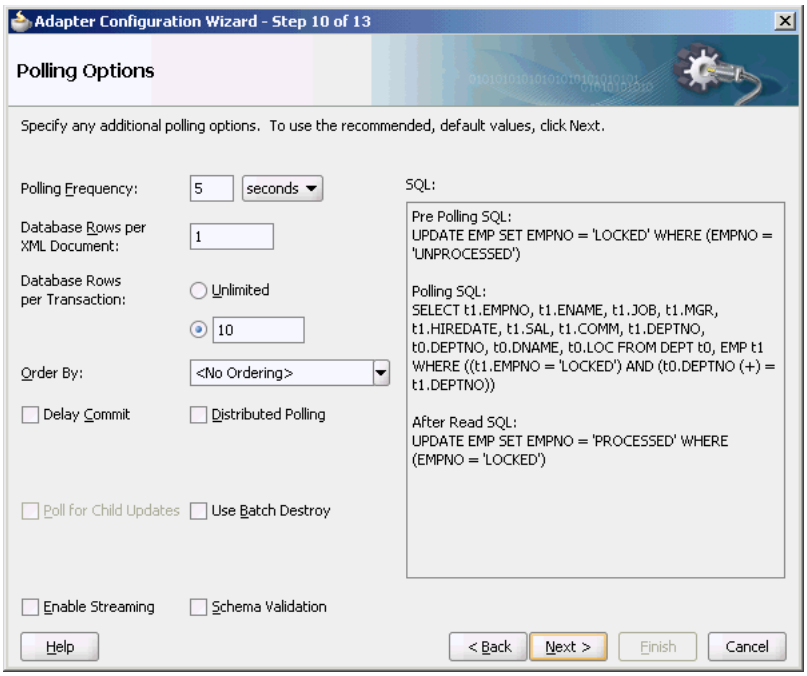
Figure 9–25 Adapter Configuration Wizard - Update a Sequencing File Page



9.2.11 Specifying Polling Options

You can specify additional polling options, if any, in this page. Figure 9–26 shows the Adapter Configuration Wizard - Polling Options page.

Figure 9–26 Specifying Polling Options



In this page, you specify details about how to poll the database table for new rows or events.

From the Polling Frequency list, select how frequently to poll for new records or events.

In the **Database Rows per XML Document** field, specify the number of rows per XML document when sending events to Oracle BPEL PM or Mediator. This is the batch setting between the database adapter and its consumer: Oracle BPEL PM or Mediator.

In the **Database Rows per Transaction** field, select **Unlimited** or enter a value to indicate the number of table rows to process during a single transaction.

When polling the database for events, you can order the returned rows by the selected column by using the **Order By** list. The best practice is to choose **<No Ordering>**, as message ordering regardless is not guaranteed without extra configuration.

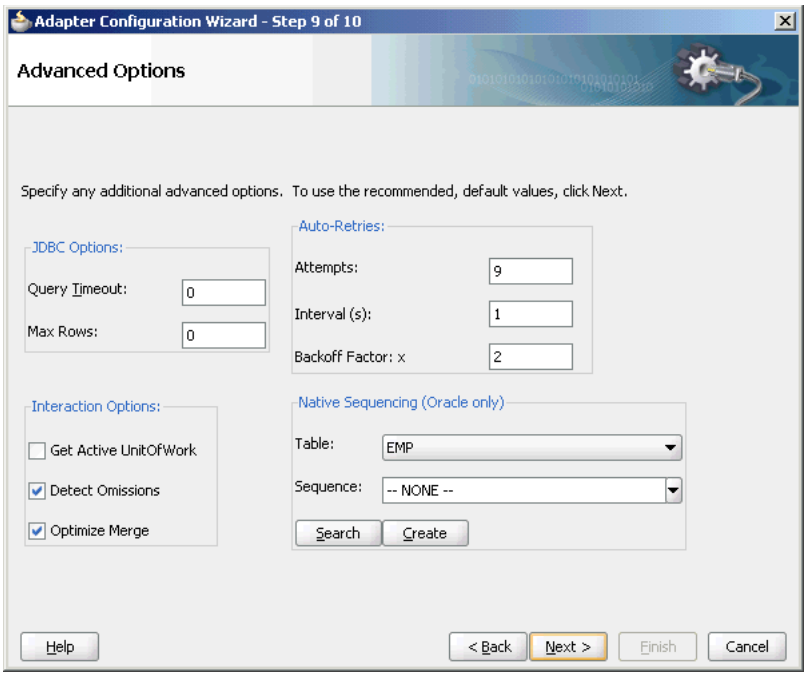
In the SQL field, if the SQL syntax is incorrect, then a message is displayed in red.

For more information about specifying polling options, click **Help** in the Polling Options page or press F1.

9.2.12 Specifying Advanced Options

You can specify advanced options, if any. Figure 9–27 shows the Adapter Configuration Wizard - Advanced Options page. In this page, you can specify advanced JDBC and DBAdapter options, configure retries, and configure native sequencing.

Figure 9–27 Specifying Advanced Options



You must specify JDBC options in the **JDBC Options** section. Set low-level JDBC options on calls to the database. The operation you selected determines which options may appear here.

In the **Auto-Retries** section, specify the value for auto-retry incase of time out. In case of a connection related fault, the Invoke activity can be automatically retried a limited number of times. You can specify the following values in the fields in this section:

- To retry indefinitely, type `unlimited` in the **Attempts** field.
- **Interval** is the delay between retries.
- **Backoff Factor: x** allows you to wait for increasing periods of time between retries. 9 attempts with a starting interval of 1 and a back off of 2 leads to retries after 1, 2, 4, 8, 16, 32, 64, 128, and 256 (2^8) seconds.

In the **Interaction Options**, specify the interaction options, as follows:

- **GetActiveUnitOfWork** is an advanced setting that forces all invoke activities in the same global transaction to use the same SQL connection if going to the same database. This makes it easier to guarantee that later invoke activities can see the changes of earlier invoke activities, but you might have to set this (if using emulated two-phase commit, it should automatically be the same connection). Another difference is that for `MERGE` and `INSERT`, all changes are not written until the global transaction commits, so this setting also changes the timing of when `WRITE` operations occur.
- **Detect Omissions** allows the `MERGE` and `INSERT` operations to ignore empty or missing XML elements in the input payload. For a `MERGE` operation, this prevents valid but unspecified values from being overwritten with NULL. For `INSERT` operations, they are omitted from the `INSERT` statement, allowing default values to take effect.
- **Optimize Merge** should always be set to true, as it is a general enhancement to `MERGE` performance (using an in query for the primary key existence check).

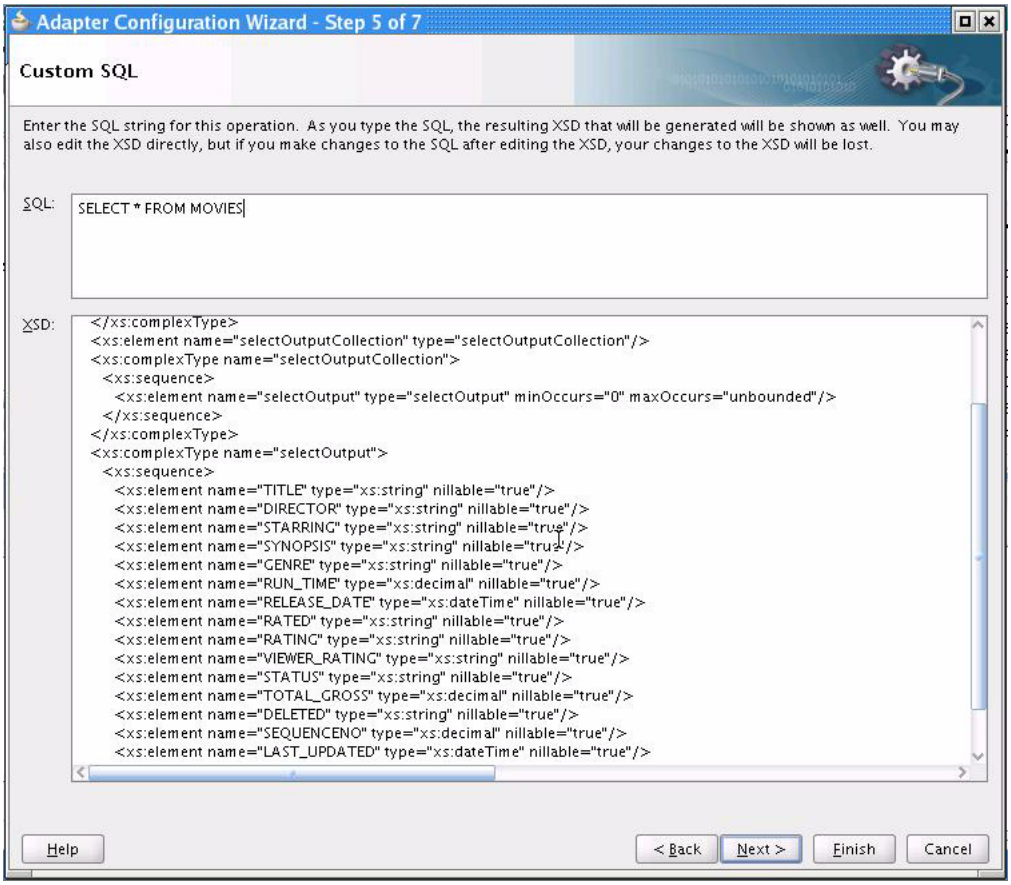
Native Sequencing (Oracle only) allows you to specify that the primary key are assigned from a sequence on any insert. Click **Search** and then select a sequence from the **Sequence** list, or type the name and click **Create**.

For more information about specifying advanced options, click **Help** in the Advanced Options page or press F1.

9.2.13 Entering the SQL String for the Pure SQL Operation

You can enter a SQL string for performing the Execute Pure SQL operation in the Custom SQL page. Figure 9–28 shows the Adapter Configuration Wizard - Custom SQL page.

Figure 9–28 Entering a SQL String



In the **SQL** field, enter a custom SQL string. An XSD schema of your SQL input is automatically created in the **XSD** field.

The XSD field displays the XSD schema of the custom SQL string you entered. You can directly edit the resulting XSD. However, if you make subsequent changes to the SQL string, then your XSD changes are lost.

For more information about entering a SQL string, click **Help** in the Custom SQL page or press F1.

9.3 Oracle Database Adapter Features

This section discusses the Oracle Database Adapter features.

It includes the following topics:

- Section 9.3.1, "Transaction Support"
- Section 9.3.2, "Pure SQL - XML Type Support"
- Section 9.3.3, "Row Set Support Using a Strongly or Weakly Typed XSD"
- Section 9.3.4, "Proxy Authentication Support"
- Section 9.3.5, "Streaming Large Payload"
- Section 9.3.6, "Schema Validation"
- Section 9.3.7, "High Availability"
- Section 9.3.8, "Scalability"
- Section 9.3.9, "Performance Tuning"
- Section 9.3.10, "`detectOmissions` Feature"
- Section 9.3.11, "`OutputCompletedXml` Feature"
- Section 9.3.12, "QueryTimeout for Inbound and Outbound Transactions"
- Section 9.3.13, "Doing Synchronous Post to BPEL (Allow In-Order Delivery)"

9.3.1 Transaction Support

The Oracle Database Adapter enables transaction support, which, along with the inherent data processing, ensures that each modification has a clearly defined outcome, resulting in either success or failure, thus preventing potential corruption of data, executes independently from other changes, and, once completed, leaves underlying data in the same state until another transaction takes place.

There are two types of transaction support, XA Transaction support and Local Transaction support. XA transaction support allows a transaction to be managed by a transaction manager external to a resource adapter, whereas, a local transaction support allows an application server to manage resources that are local to the resource adapter.

To ensure two Oracle Database Adapter invokes commit or rollback as a unit, you must perform the following:

- Both Oracle Database Adapter invokes must be configured to participate in global transactions.
- Both Oracle Database Adapter invokes must participate in the same global transaction.
- The failure of either invoke must cause the global transaction to roll back.

The transaction support is demonstrated in the following tutorial files:

- `XAInsert`
- `InsertWithCatch`
- `DirectSQLPerformance`

To obtain the tutorial sample code, access the Oracle SOA Sample Code site.

> **Note:** You must use a non-XA driver with the `SOALocalTxDataSource` parameter. Switching to an XA driver breaks product functionality.

9.3.1.1 Configuring Oracle Database Adapter for Global Transaction Participation

In the deployment descriptor (`weblogic-ra.xml` file), you must set the `xADataSourceName` parameter. Additionally, the referenced DataSource must be configured for transaction participation by creating a data source in Oracle WebLogic Server Console.

You must create a data source and choose a XA data sources from the list.

> **Note:** True Database XA is only certified on Oracle 10.2.0.4 or 11.1.0.7. For earlier versions, you are safer picking a non-XA data source implementation and selecting **Emulated Two-phase commit** on the next page.

For information about the recommended setting for non-XA and XA data sources used by Oracle JCA Adapters, see Section 2.20, "Recommended Setting for Data Sources Used by Oracle JCA Adapters."

You cannot edit the `data-sources.xml` file in the Oracle WebLogic Server. You must create a data source by using the Oracle WebLogic Server Administration Console, as mentioned in Section 2.18.1, "Creating a Data Source."

9.3.1.2 Both Invokes in Same Global Transaction

Once both the Oracle Database Adapter invokes participate in global transactions, to commit or rollback as a unit, they must be participating in the same global transaction. In BPEL, this requires the understanding of where the transaction boundaries are, at what points does a checkpoint have to write to the dehydration store, commit the current global transaction, and start a new one.

The transaction boundaries in a BPEL process occur either before a Receive activity or wait activity, or before an `onMessage` or pick activity. This may also occur when invoking a synchronous child BPEL process, unless the `bpel.config.transaction` property is set on the partnerlink, as shown in the following code sample.

```
<property name="bpel.config.transaction">required</property>
```

Otherwise, the parent process is broken into two transactions and the child process runs in its own transaction.

9.3.1.3 Failure Must Cause Rollback

Finally, even if both Oracle Database Adapter invokes participate in the same global transaction, the failure of either invoke may not cause the global transaction to rollback.

The only cases where a failure can actually cause a global rollback are:

- A Oracle Database Adapter operation that inserts/updates multiple tables as part of one invoke fails after having succeeded in some writes but not others. In this case, the Oracle Database Adapter marks the global transaction as rollback only, because the invoke operation was not atomic and a commit could cause data corruption.
- The invoke retries multiple times in a database down scenario, until the global transaction times out and is rolled back.
- An explicit `bpelx:rollback` fault is thrown from within the BPEL process.

9.3.1.3.1 Using the Same Sessions for Both Invokes

You must set the `GetActiveUnitOfWork` JCA parameter to true to enable using the same sessions or connections for both the Oracle Database Adapter invokes.

`GetActiveUnitOfWork` is an advanced JCA property you can set on any `DBInteractionSpec`. It causes the invoke to register itself with the two-phase commit callbacks, and all writes to the database are performed as part of the two-phase commit. By setting this property on any failure, the transaction is automatically rolled back, as there is no way to *handle* a fault at this late stage. Similarly, the same underlying TopLink session is used for both invokes, meaning if you merge the same object twice, it is inserted/updated once. All merge invokes that set `GetActiveUnitOfWork` as true are cumulative.

9.3.1.4 Transaction/XA Support

To make two Oracle Database Adapter invokes commit or roll back as a unit requires the following: both Oracle Database Adapter invokes must be configured to participate in global transactions, both invokes must participate in the same global transaction, and the failure of either invoke must cause the global transaction to rollback.

9.3.1.4.1 Configuring an Oracle Database Adapter for Global Transaction Participation

In the deployment descriptor (weblogic-ra.xml), you must set `xADataSourceName`. The matching data source entry must be configured for global transaction participation.

True XA: Two-Phase (XA) Versus One-Phase (Emulated) Commit

XA is a two-phase commit protocol, which is more robust than a one-phase commit or emulated protocol. The difference is that with a one-phase protocol, you may very rarely still see message loss or other rollback/commit inconsistency, on the order of one per one thousand generally.

Oracle RAC Configuration

For more information about Oracle RAC configuration, see the *Oracle Database High Availability Overview guide*.

True XA Configuration with Third Party Drivers

When configuring true XA for third party drivers (that is, Microsoft SQL Server 2008, IBM DB2), see if the driver jars contain a class that implements `javax.sql.XADataSource`.

For data direct drivers, the naming happens to be `com.oracle.ias.jdbcx.db2.DB2DataSource`, or `com.oracle.ias.jdbcx.sqlserver.SQLServerDataSource`.

9.3.1.4.2 Failure Must Cause Rollback

Finally, even if both invokes participate in the same global transaction, the failure of either invoke may not cause the global transaction to roll back.

The only cases where a failure can actually cause a global roll back are:

- An Oracle Database Adapter operation that inserts/updates multiple tables as part of one invoke fails after having succeeded in some writes but not others. In this case, the adapter marks the global transaction rollback only, as the invoke operation was not atomic and a commit could cause data corruption.

- The invoke retries multiple times in a database down scenario, until the global transaction times out and is rolled back.
- An explicit `bpelx:rollback` fault is thrown from within the BPEL process. `GetActiveUnitOfWork="true"` in WSDL.

9.3.2 Pure SQL - XML Type Support

Pure SQL Adapter is an option in the Oracle Database Adapter Wizard that allows you to type the SQL string directly and have an XSD/Web service generated automatically. The database tables are introspected dynamically in the Adapter Configuration Wizard to test the SQL and populate the XSD file better (that is, with valid return types.)

The Pure SQL support allows the Oracle Database Adapter to deal with tables/views as entities and for dealing directly with SQL. You can use Pure SQL:

- for simple data projection style report queries
- in cases where the result set is not table oriented, such as select count(*)
- to perform an update or delete all
- when working with `XMLType` columns and `xquery`
- when using complex SQL, which are not modeled in the Adapter Configuration Wizard expression builder

You can use the Pure SQL Adapter with Oracle `XMLTypes`. It is a natural fit for inserting XML into `XMLType` tables and columns, and retrieving XML using `xquery` selects. Pure SQL is a natural fit for the Oracle Database Adapter that provides a relational-xml mapping that parallels XML DB(XDB) support. So, when using XDB the adapter should be as lightweight and transparent as possible, to let you focus on XDB and `XQuery`.

If your data is in XML (unstructured/semi-structured) format, and you have no relational schema at all that you can map your data to, then you could use XDB. The conventional Oracle Database Adapter allows you to import an existing relational schema as an XML schema to be used with Web services. XDBs XML shredding algorithm can generate a relational schema from an existing XML schema for persistent storage.

Note: Use of schema bound `XMLTypes` requires the `oci` driver, which is not certified in the 11*g* release. Therefore, you must use non-schema bound `XMLTypes` at run time, though you can use schema bound `XMLTypes` at design time to import a representative XSD.

For more information, see:

- Section 9.2.4, "Selecting the Operation Type"
- "Use Cases for Pure SQL" on page 9-54

9.3.3 Row Set Support Using a Strongly or Weakly Typed XSD

Currently a `REF CURSOR` by nature can support any arbitrary result set, so the XSD generated at design time allows this and looks like the XSD that Example 9–1 shows.

Note: Oracle Database stored procedures return result sets that are referred to as `RefCursors`, whereas third-party databases result sets that are returned are referred to as `RowSets`.

Example 9–1 Weakly Typed XSD

```
<refCursorOutputParam>
    <Row>
        <Column name="DEPTNO" sqltype="NUMBER">20</Column>
        ...
    </Row>
</refCursorOutputParam>
```

However the XML output from this is hard to use. It is very difficult to write an Xpath expression or XSL based on a weakly typed XSD and column names as attribute values instead of element names.

Although a row set can represent any result set, it is possible to assume for some procedures that it has the same structure each time, and hence can be described with a strongly typed XSD. A strongly typed XSD is almost a necessity to transform the result set to another XSD later on. A strongly typed XSD looks like the XSD that Example 9–2 shows.

Example 9–2 Strongly Typed XSD

```
<refCursorOutputParam>
    <dept>
        <deptno>20</deptno>
        ...
    </dept>
</refCursorOutputParam>
```

You can use the Adapter Configuration Wizard to create a strongly typed XSD for a row set returned by a stored procedure or function `REF CURSOR` variable. An Oracle Database function is a special stored procedure that always has one out variable, and can be inlined - for example, inside select statements - and so traditionally does not do updates.

Using this feature, you can select a stored procedure (or stored function), enter its arguments, and perform a test execution to retrieve an actual row set. The Adapter Configuration Wizard then introspects the returned row set and generates a strongly typed XSD. You can enter arguments easily through the wizard. For example, you can enter numbers and strings directly, dates as literals (2009/11/11), and you can even enter structs like `MYOBJ('a', 'b')`.

Note: Functions are not supported for IBM DB2 UDB. Only SQL stored procedures are supported.

The Adapter Configuration Wizard row set support using a strongly typed XSD has the following restrictions:

- Oracle Database PL/SQL `record` or `boolean` types are not supported.
- Oracle Database PL/SQL `varray` is not supported.
- Oracle Database PL/SQL `%rowtype` is not supported.
- Oracle Database PL/SQL `table` types are not supported.

- Oracle Database PL/SQL procedures with `IN` only `REF CURSOR` parameters are not supported.

  For an Oracle Database PL/SQL procedure with `REF CURSOR` as an `IN/OUT` parameter, the Adapter Configuration Wizard ignores the `IN` and generates the strongly typed XSD based on the `OUT` parameter.
- Referencing an element in the XSD using `ref` is not supported.
- SQL Server 2008 table valued functions and CLR functions are not supported.

The Oracle Database Adapter supports strongly typed XSD for the following third-party databases:

- Microsoft SQL Server 2005
- Microsoft SQL Server 2008
- IBM DB2 UDB 9.7

The Oracle Database Adapter does not support strongly typed XSD for the following third-party databases:

- IBM DB2 AS/400
- MySQL
- Informix Dynamic Server
- Sybase 15.0.2

For more information, see:

- Section 9.7, "Stored Procedure and Function Support"
- Section 9.7.7.1, "Row Set Support Using a Strongly Typed XSD"
- Section 9.7.7.2, "Row Set Support Using a Weakly Typed XSD"

9.3.4 Proxy Authentication Support

You can connect to your Oracle data store by using Proxy Authentication. On a per-invoke basis, you can set a combination of the following new header properties:

- `jca.db.ProxyUserName`: to use the `OracleConnection.PROXYTYPE_USER_PASSWORD` proxy type, set this property to the proxy user name as a `java.lang.String`.
- `jca.db.ProxyPassword`: to use the `OracleConnection.PROXYTYPE_USER_PASSWORD` proxy type, set this property to the proxy user password as a `java.lang.String`.
- `jca.db.ProxyCertificate`: to use the `OracleConnection.PROXYTYPE_CERTIFICATE` proxy type, set this property to a `base64Binary` encoded `byte[]` array containing a valid certificate.

  This is a more encrypted way of passing the credentials of the user, who is to be proxied, to the database. The certificate contains the distinguished name encoded in it. One way of generating the certificate is by creating a wallet and then decoding the wallet to get the certificate. The wallet can be created using `runutl mkwallet`. It is then necessary to authenticate using the generated certificate.
- `jca.db.ProxyDistinguishedName`: to use the `OracleConnection.PROXYTYPE_DISTINGUISHED_NAME` proxy type, set this property to the proxy distinguished name as a `java.lang.String`.

This is a global name in lieu of the password of the user being proxied for.

- `jca.db.ProxyRoles`: regardless of what proxy type you use, you can optionally set this property to define the roles associated with the proxy user as a `String[]` array where each `java.lang.String` corresponds to a role name.
- `jca.db.ProxyIsThickDriver`: if you are using the OCI driver, set this property to a value of `true` to accommodate differences in the JDBC-level API between the thick and thin drivers.

To run the invoke, a proxy connection is obtained from the data source.

For more information, see Chapter 10, "Proxy Authentication", in the *Oracle Database JDBC Developer's Guide and Reference*

9.3.5 Streaming Large Payload

To enable support to stream payload, you must select the Enable Streaming check box while specifying polling options, as shown in Figure 9–26. When you enable this feature, the payload is streamed to a database instead of getting manipulated in SOA run time as in a memory DOM. You use this feature while handling large payloads. When you select the Enable Streaming check box, a corresponding Boolean property `StreamPayload` is appended to the ActivationSpec properties defined in the respective `.jca` file.

9.3.6 Schema Validation

The `SchemaValidation` [false/true] property is a new activation specification property that has been added, and this can be configured in a `.jca` file. When set to true, all XML files produced by the polling Oracle Database Adapter (for Receive activities) is validated against the XSD file. On failure, the XML record is rejected but still marked as processed by the Oracle Database Adapter.

Databases provide structured storage and the XSD file is generated by the Oracle Database Adapter Wizard itself. However, if you edit the auto generated XSD and add your own restrictions, you may want to start validation. For instance, if you import a VARCHAR(50) field, the auto-generated XSD has the max-length 50 restriction. However, if your BPEL process for some reason can only handle values of fixed length 22, it may want to validate the XML file.

9.3.7 High Availability

The Oracle Database Adapter supports high availability in an active-active setup. In an active-active setup, distributed polling techniques can be used for inbound Database Adapters to ensure that the same data is not retrieved more than once. For more information, see Section 9.3.8.1, "Distributed Polling First Best Practice: SELECT FOR UPDATE (SKIP LOCKED)." Similar to other adapters, an Oracle Database Adapter can also be configured for singleton behavior within an active-passive setup. This allows a high performance multithreaded inbound Oracle Database Adapter instance running in an active-passive setup, to follow a fan out pattern and invoke multiple composite instances across a cluster. The Oracle Database Adapter also supports the high availability feature when there is a database failure or restart. The DB adapter picks up again without any message loss.

9.3.8 Scalability

The following sections describe best practice for multiple Oracle Database Adapter process instances deployed to multiple Oracle BPEL PM or Mediator nodes, including:

- Section 9.3.8.1, "Distributed Polling First Best Practice: SELECT FOR UPDATE (SKIP LOCKED)"
- Section 9.3.8.2, "Distributed Polling Second Best Practice: Tuning on a Single Node First"

9.3.8.1 Distributed Polling First Best Practice: SELECT FOR UPDATE (SKIP LOCKED)

The first best practice for multiple Oracle Database Adapter process instances deployed to multiple Oracle BPEL PM or Mediator nodes is to use the Adapter Configuration Wizard to set both the **Distributed Polling** check box in the Adapter Configuration Wizard and to set `MaxTransactionSize`. Increase concurrency by setting the `adapter _db.JCA` property `NumberOfThreads`.

On an Oracle database, this automatically uses the syntax `SELECT FOR UPDATE SKIP LOCKED`. Concurrent threads each try to select and lock the available rows, but the locks are only obtained on fetch. If an about to be fetched row is locked, the next unlocked row are locked and fetched instead. If many threads all execute the same polling query at the same time, they should all relatively quickly obtain a disjoint subset of unprocessed rows.

On a non-Oracle database the `SELECT FOR UPDATE` safely ensures that the same row cannot be processed multiple times, however you may get less scalability. You should consider either using additionally a partition field or the second best practice which is essentially multi-threading on a single node with fan-out (see Section 9.3.8.2, "Distributed Polling Second Best Practice: Tuning on a Single Node First").

> **Note:** A distributed approach is required to insure that multiple activation instances do not process the same rows.

When configuring this best practice, consider the following:

- Section 9.3.8.1.1, "Configuring PollingInterval, MaxTransactionSize, and ActivationInstances"
- Section 9.3.8.1.2, "Partition Field"
- Section 9.3.8.1.3, "activationInstances"
- Section 9.3.8.1.4, "Indexing and Null Values"
- Section 9.3.8.1.5, "Disabling Skip Locking"
- Section 9.3.8.1.6, "MarkReservedValue"
- Section 9.3.8.1.7, "SequencingPollingStrategy (Last Read or Last Updated)"

9.3.8.1.1 Configuring PollingInterval, MaxTransactionSize, and ActivationInstances

In a distributed scenario, each polling instance tries to balance the load by not greedily attempting to process all unprocessed rows by itself. What that means is that at a time, an instance only fetches at most `MaxTransactionSize` rows.

When using skip locking, if a full `MaxTransactionSize` rows are fetched, the next `MaxTransactionSize` rows can be immediately fetched continuously. This is because concurrent threads do no block each other when using skip locking, so there is no danger of one instance fetching all the rows.

However, with skip locking disabled, all threads tries to lock the same rows, and only one succeeds. Consequently, once this thread has processed `MaxTransactionSize`

rows, it pauses until the next polling interval, to allow other threads to also lock and process rows.

Hence, the maximum throughput with distributed polling enabled but uses SkipLocking disabled is:

```
NumberOfThreads x MaxTransactionSize/PollingInterval
```

> **Note:** Although you may want to increase `MaxTransactionSize`, if you increase it too high, you may start to see transaction timeouts. Table 9-2 lists safe values for `MaxTransactionSize`.

For load balancing purposes, it is dangerous to set the `MaxTransactionSize` too low in a distributed environment with skip locking disabled (where MaxTransactionSize becomes a speed limit). It is best to set the `MaxTransactionSize` close to the per CPU throughput of the entire business process. This way, load balancing occurs only when you need it.

Table 9–2 MaxTransactionSize and MaxRaiseSize Values

MaxTransactionSize	MaxRaiseSize	Description
10	1	When using sequential routing. For 10 rows you have 10 individual instances and 10 XML records passing through SOA.
100		When using parallel routing.
>= 100	`MaxTransactionSize`	When using the adapter to stream rows through as fast as possible.

For load balancing purposes, it is dangerous to set the `MaxTransactionSize` too low in a distributed environment (where it becomes a speed limit). It is best to set the `MaxTransactionSize` close to the per CPU throughput of the entire business process. This way, load balancing occurs only when you need it.

If distributed polling is not set, then the adapter tries to process all unprocessed rows in a single polling interval.

9.3.8.1.2 Partition Field In a distributed scenario there are polling instances on multiple servers, however, per server there can be multiple threads configured. You can configure these activation instances to cooperate somewhat by processing separate rows, possibly improving scaling.

To so, simply add the property `PartitionField` to your `db.jca` file:

```
<property name="PartitionField" value="ID"/>
```

If you set `activationInstances` to 2, then activation instances 1 and 2 (or 0 and 1) would respectively execute:

```
SELECT ... WHERE ... AND MOD (ID, 2) = 0 FOR UPDATE SKIP LOCKED
```

and

```
SELECT ... WHERE ... AND MOD (ID, 2) = 1 FOR UPDATE SKIP LOCKED
```

Activation instance 0 still conflicts with other activation instances with this ID on other servers, but at least it does not conflict with other activation instances with ID 1.

Ensure that the partition field is numeric and that applying `mod` evenly distribute the rows (that is, in this case make sure all the IDs are not either even or odd).

On Oracle Database, you can set the partition field to be `rowid` by setting `db.jca` file property `PartitionField` as follows:

```
<property name="PartitionField" value="rowid"/>
```

Then the SQL is in fact converted to:

```
SELECT ... WHERE ... AND MOD (dbms_rowid.rowid_row_number(rowid), 2) = [0/1] FOR
UPDATE SKIP LOCKED
```

Because Oracle Database skip locking provides scalability, setting a partition field is not recommended. There is a cost of increased database CPU usage with more complex SQL.

9.3.8.1.3 activationInstances T

The adapter framework level property `activationInstances` (configured in `composite.xml`) is interchangeable with `NumberOfThreads` for distributed scenarios.

Setting `activationInstances` to 5 and NumberOfThreads to 5 is equal to setting one to 25 and the other to 1. As the extra work instances are created outside of the DbAdapter, they do not cooperate in any way. Hence, in a multi-threaded single node scenario, always configure `NumberOfThreads` only. Without database level concurrency control through enabling distributed polling, duplicates are read.

Note: In a distributed cluster scenario configuring `NumberOfThreads` or `activationInstances` has the same effect. For a non distributed scenario, you must use `NumberOfThreads`. Hence it is safe to always use `NumberOfThreads` and disregard `activationInstances`.

For more information, see Section 2.13, "Singleton (Active/Passive) Inbound Endpoint Lifecycle Support Within Adapters".

9.3.8.1.4 Indexing and Null Values

Try to index (and/or add explicit constraints on the database for) the primary and all foreign keys to joined tables. If using Logical delete polling, try to index the status column. Try to configure a non-null `MarkUnreadValue` and `MarkReadValue`.

If you have no indexes at all and prefer to have none, you can proceed with the single node multi-threaded approach (see Section 9.3.8.2, "Distributed Polling Second Best Practice: Tuning on a Single Node First"). That way the polling query is executed once, which might be a full table scan, but multiple threads help to exhaust the entire result set until all rows are processed. With a distributed approach all work must be done while the rows are exclusively locked, which means locked in a timed transaction. In a distributed scenario there are many repeated selects, which may harm performance if each one is doing a full table scan.

Note: Performance is very slow if `MarkUnreadValue` is configured as null.

9.3.8.1.5 Disabling Skip Locking

Skip locking has been available on Oracle Database since Oracle 8 but is documented in Oracle 11. You rarely come across an incompatible feature and have to disable it. In that case you can set the Oracle

Database Adapter connector property `usesSkipLocking` to `false` in the `ra.xml` file you deploy with your application as Example 9–3 shows.

Example 9–3 Configuring usersSkipLocking in ra.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd" version="1.5">
...
  <resourceadapter>
        <outbound-resourceadapter>
            <connection-definition>
...
            <config-property>
              <config-property-name>usesSkipLocking</config-property-name>
              <config-property-type>java.lang.Boolean</config-property-type>
                    <config-property-value>false</config-property-value>
            </config-property>
...
            </connection-definition>
...
        </outbound-resourceadapter>
  </resourceadapter>
</connector>
```

For more information on how to configure connector-level properties, see:

- "Configuring the ra.xml File" in the *Oracle Fusion Middleware Programming Resource Adapters for Oracle WebLogic Server*
- "Packaging and Deploying Resource Adapters" in the *Oracle Fusion Middleware Programming Resource Adapters for Oracle WebLogic Server*

9.3.8.1.6 MarkReservedValue If you are using Logical Delete polling and you set `MarkReservedValue`, skip locking is not used.

Formerly, the best practice for multiple Oracle Database Adapter process instances deployed to multiple Oracle BPEL Process Manager or Oracle Mediator nodes was essentially using `LogicalDeletePollingStrategy` or `DeletePollingStrategy` with a unique `MarkReservedValue` on each polling node, and setting `MaxTransactionSize`.

However with the introduction of skip locking in this release, that approach has now been superseded. If you were using this approach previously, you can simply remove (in `db.jca`) or clear (Logical Delete Page of wizard) the `MarkReservedValue`, and you automatically get skip locking.

The benefits of using skip locking over a reserved value include:

- Skip locking scales better in a cluster and under load.
- All work is in one transaction (as opposed to update/reserve, then commit, then select in a new transaction), so the risk of a non-recoverable situation in an HA environments is minimized.
- No unique `MarkReservedValue` must be specified. For this to work you had to configure a complex variable like `R${weblogic.Name-2}-${IP-2}-${instance}`.

9.3.8.1.7 SequencingPollingStrategy (Last Read or Last Updated)

This distributed approach works with Delete or Logical Delete based polling strategies.

The work of the sequencing polling based strategies cannot be distributed as records are initially processed in order.

For example, the second row cannot be marked as processed ahead of the first (setting last read ID to 2 means not just that 2 has been processed but 1 also).

However, as the sequencing polling strategies are non-intrusive, requiring no post updates or deletes to the source tables, they are extremely fast.

Use sequencing polling strategies with a single node and with fan-out on a cluster. It is still safe to use in a cluster; however, the select for update is instead applied on accessing the last read ID in the helper table.

9.3.8.2 Distributed Polling Second Best Practice: Tuning on a Single Node First

The next best practice for multiple Oracle Database Adapter process instances deployed to multiple Oracle BPEL PM or Mediator nodes is to tune on a single node first.

For an Oracle Database Adapter intensive process, such as a database-database integration, performance can be improved by a factor 10 or 100 just by tuning on a single Java Virtual Machine (JVM), scaling |NumberOfThreads|, and setting high values for MaxTransactionSize and MaxRaiseSize.

As Section 9.3.8.1, "Distributed Polling First Best Practice: SELECT FOR UPDATE (SKIP LOCKED)" describes, there may be times where it is best to improve performance on a single node, and then optionally do fan-out to multiple nodes in a cluster. Relying on concurrency control features of the database such as locking can be great, but these are often designed more for preserving data integrity than for high performance scalability.

Cases where it may be best to do polling on a single node in the cluster include using the non-intrusive Sequencing Polling strategy, polling large un-indexed tables, or using a non-Oracle back-end database that does not provide high concurrency locks like skip locks.

Note: For Oracle Database Adapter with polling operation in a clustered environment, you must use the option of distributed polling by selecting the **Distributed Polling** check box in the Adapter Configuration Wizard.

You can also refer to Section 2.13, "Singleton (Active/Passive) Inbound Endpoint Lifecycle Support Within Adapters".

For the samples `MultiTablesPerformance` and `DirectSQLPerformance` that show tuning on a single node, access the Oracle SOA Sample Code site.

9.3.9 Performance Tuning

The Oracle Database Adapter is preconfigured with many performance optimizations. You can, however, make some changes to reduce the number of round trips to the database by implementing performance tuning.

For information about performance tuning, see:

- "Oracle JCA Adapter for Database Tuning" in the Oracle Fusion Middleware Performance and Tuning Guide
- "Inbound Database Adapter Tuning" in the Oracle Fusion Middleware Performance and Tuning Guide

9.3.10 `detectOmissions` Feature

The following are the features of the `detectOmission` feature:

Available Since

Release 10.1.3

Configurable

Yes

Default Value

Design Time: `true`, unless explicitly set to `false`

Use Case

Users may pass incomplete or partial XML to a merge, update, or insert, and see that every column they left unspecified in XML is set to null in the database.

It allows `DBAdapter` merge, insert, or update to differentiate between `null` value and the absence of a value (omission) in XML documents. On a case by case basis, it determines which information in XML is meaningful and which is not. In this way XML is seen as a partial representation of a database row, as opposed to a complete representation. The following table lists examples for null values, and values that can be omitted.

Element Type	Omission	Null

Column	`<director></director>`
	`<director />`
	`<!-- director>…</director -->`
		`<director xsi:nil="true" />`

1-1	`<!-- dept> … </dept -->`	`<dept xsi:nil="true" />`
1-M	`<!-- empCollection>…` `</empCollection -->`	`</empCollection>` `</empCollection>` (empty)

> **Note:** The 1-1 representation `<dept />` denotes an empty department object and should not be used.For 1-M, `<empCollection />` actually means a collection of 0 elements and is considered a meaningful value.For columns, `<director></director>` is not considered an omission in cases where it represents an empty string.

A value considered omitted is omitted from `UPDATE` or `INSERT` SQL. For an update operation, existing (meaningful) values on the database are not overwritten. For an insert operation, the default value on the database is used, as no explicit value is provided in the SQL string.

A DBAdapter receive is not able to produce XML with omissions, and makes use of xsi:nil="true". If you are unable to produce input XML with xsi:nil="true", or are concerned about the difference between <director /> and <director></director>, then it is best to set DetectOmissions="false" in the JCA file.

To treat all null values as omissions, check out the IgnoreNullsMerge sample, which comes with a custom TopLink plugin. The plugin works similar to this feature, but cannot detect subtleties between null and omission.

To obtain the IgnoreNullsMerge sample code, access the Oracle SOA Sample Code site.

When you are expecting an update, you can improve performance, by omitting 1-1 and 1-M relationships. Because the merge operation can skip considering the detail records completely.

Alternatively, map only those columns that you are interested in, and create separate mappings for different invokes. If two updates should update two different sets of columns, create two separate partnernlinks.

Performance

By default, XML is not used as an input to the Oracle Database Adapter containing omissions. Until an XML with omissions is detected, there is no performance overhead. Once omissions are detected, a TopLink descriptor event listener is added. This event listener has some overhead, and every modifyRow about to become a SQLUpdate or SQLInsert must be iterated over, to check for omissions. Hence, every column value sent to the database is checked. If the input XML has mostly omissions, then the cost overhead should be more than compensated by sending fewer values to the database.

Incompatible Interactions

DirectSQL="true" and DetectOmissions="true" - DetectOmissions takes precedence. The following are some examples for incompatible interactions:

- DetectOmissionsMerge
- IgnoreNullsMerge
- OptimizeMerge

> **Note:** For migrated old BPEL project, you must re-run the Database Adapter Wizard to regenerate the JCA file. When re-run the Database Adapter Wizard, the DetectOmissions and OptimizeMerge options appear in the JCA file with default values as DetectOmissions="false" and OptimizeMerge="false".

See the following for more information:

You can also access the forums from Oracle Technology Network at

- The Oracle BPEL Process Manager forum at

  http://forums.oracle.com/forums/forum.jspa?forumID=212

- The TopLink forum at

  http://forums.oracle.com/forums/forum.jspa?forumID=48

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

http://www.oracle.com/technology

9.3.11 OutputCompletedXml Feature

OutputCompletedXml is a feature of the outbound insert activity. The following are some of the features of the OutputCompletedXml feature:

Available Since

Release 10.1.2.0.2

Configurable

OutputCompletedXml appears in the JCA file only when default is true.

Default Value

It is true when TopLink sequencing is configured to assign primary keys on insert from a database sequence, otherwise it is false.

Issue

You can have primary keys auto-assigned on insert from a database sequence. However, the usefulness of this feature is diminished, because insert/merge have no output message, so there is no way to tell which primary keys were assigned.

Note: After configuring sequencing (link), run the Adapter Configuration Wizard again so that the insert/merge WSDL operations can be regenerated with an output message, and WSDL property OutputCompletedXml="true".

Performance

An output XML is provided only when the output XML would be significantly different, so if TopLink sequencing is not used, then this feature is disabled and there is no performance hit. Further, this feature can be explicitly disabled. Likewise, the original input XML is updated and returned; a completely new XML is not built. Also only a shallow update of the XML is performed; if primary keys were assigned to detail records, then these are not reflected in the output XML.

Incompatible Interactions

DirectSQL="true" and "OutputCompletedXml" - OutputCompletedXml takes precedence.

9.3.12 QueryTimeout for Inbound and Outbound Transactions

You can configure QueryTimeout from the Adapter Configuration Wizard-Advanced Options page. This feature exposes the java.sql.Statement level property of the same name. Essentially, QueryTimeout allows you to configure a timeout on the call.

9.3.13 Doing Synchronous Post to BPEL (Allow In-Order Delivery)

In this feature, the entire invocation is in a single thread and global transaction. By default, initiation is asynchronous and the BPEL process is invoked in a separate global transaction. With Oracle Mediator, it is generally a synchronous invoke so this is only specific to an Oracle BPEL process.

To enable this feature, click the **Do Synchronous Post to BPEL (Allow In-Order Delivery)** option in the Adapter Configuration Wizard - Operation page.

9.4 Oracle Database Adapter Concepts

This section includes the following topics related to Oracle Database Adapter Concepts:

- Section 9.4.1, "Relational-to-XML Mapping"
- Section 9.4.2, "SQL Operations as Web Services"

9.4.1 Relational-to-XML Mapping

This section includes the following topics related to Relational-to-XML mapping:

- Section 9.4.1.1, "Relational Types to XML Schema Types"
- Section 9.4.1.2, "Mapping Any Relational Schema to Any XML Schema"
- Section 9.4.1.3, "Querying over Multiple Tables"

For a flat table or schema, the relational-to-XML mapping is easy to see. Each row in the table becomes a complex XML element. The value for each column becomes a text node in the XML element. Both column values and text elements are primitive types.

Table 9–3 shows the structure of the `MOVIES` table. This table is used in the use cases described in this chapter. See Oracle Database Adapter Use Cases for more information.

Table 9–3 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

Table 9–3 (Cont.) MOVIES Table Description

Name	Null?	Type
LAST_UPDATED	--	DATE

The corresponding XML schema definition (XSD) is as follows:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<xs:schema targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/db/top
/ReadS1" xmlns="http://xmlns.oracle.com/pcbpel/
adapter/db/top/ReadS1" elementFormDefault=
"qualified" attributeFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="MoviesCollection" type="MoviesCollection"/>
  <xs:complexType name="MoviesCollection">
     <xs:sequence>
        <xs:element name="Movies" type="Movies" minOccurs="0"
maxOccurs="unbounded"/>
     </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Movies">
     <xs:sequence>
        <xs:element name="title">
           <xs:simpleType>
              <xs:restriction base="xs:string">
                 <xs:maxLength value="50"/>
              </xs:restriction>
           </xs:simpleType>
        </xs:element>
        <xs:element name="director" minOccurs="0" nillable="true">
           <xs:simpleType>
              <xs:restriction base="xs:string">
                 <xs:maxLength value="20"/>
              </xs:restriction>
           </xs:simpleType>
        </xs:element>
        <xs:element name="starring" minOccurs="0" nillable="true">
           <xs:simpleType>
              <xs:restriction base="xs:string">
                 <xs:maxLength value="100"/>
              </xs:restriction>
           </xs:simpleType>
        </xs:element>
        <xs:element name="synopsis" minOccurs="0" nillable="true">
           <xs:simpleType>
              <xs:restriction base="xs:string">
                 <xs:maxLength value="255"/>
              </xs:restriction>
           </xs:simpleType>
        </xs:element>
        <xs:element name="genre" minOccurs="0" nillable="true">
           <xs:simpleType>
              <xs:restriction base="xs:string">
                 <xs:maxLength value="70"/>
              </xs:restriction>
           </xs:simpleType>
        </xs:element>
        <xs:element name="runTime" type="xs:decimal" minOccurs="0"
 nillable="true"/>
        <xs:element name="releaseDate" type="xs:dateTime" minOccurs="0"
```

```
nillable="true"/>
       <xs:element name="rated" minOccurs="0" nillable="true">
          <xs:simpleType>
             <xs:restriction base="xs:string">
                <xs:maxLength value="6"/>
             </xs:restriction>
          </xs:simpleType>
       </xs:element>
       <xs:element name="rating" minOccurs="0" nillable="true">
          <xs:simpleType>
             <xs:restriction base="xs:string">
                <xs:maxLength value="4"/>
             </xs:restriction>
          </xs:simpleType>
       </xs:element>
       <xs:element name="viewerRating" minOccurs="0" nillable="true">
          <xs:simpleType>
             <xs:restriction base="xs:string">
                <xs:maxLength value="5"/>
             </xs:restriction>
          </xs:simpleType>
       </xs:element>
       <xs:element name="status" minOccurs="0" nillable="true">
          <xs:simpleType>
             <xs:restriction base="xs:string">
                <xs:maxLength value="11"/>
             </xs:restriction>
          </xs:simpleType>
       </xs:element>
       <xs:element name="totalGross" type="xs:decimal" minOccurs="0"
nillable="true"/>
       <xs:element name="deleted" minOccurs="0" nillable="true">
          <xs:simpleType>
             <xs:restriction base="xs:string">
                <xs:maxLength value="5"/>
             </xs:restriction>
          </xs:simpleType>
       </xs:element>
       <xs:element name="sequenceno" type="xs:decimal" minOccurs="0"
nillable="true"/>
       <xs:element name="lastUpdated" type="xs:dateTime" minOccurs="0"
nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

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

Table 9–4 and Table 9–5 show the structure of the `EMP` and `DEPT` tables, respectively. These tables are used in the `MasterDetail` use case. See Oracle Database Adapter Use Cases for more information.

Table 9–4 EMP Table Description

Name	Null?	Type
`EMPNO`	NOT NULL	`NUMBER(4)`

Table 9–4 (Cont.) EMP Table Description

Name	Null?	Type
ENAME	--	VARCHAR2(10)
JOB	--	VARCHAR2(9)
MGR	--	NUMBER(4)
HIREDATE	--	DATE
SAL	--	NUMBER(7,2)
COMM	--	NUMBER(7,2)
DEPTNO	--	NUMBER(2)

Table 9–5 DEPT Table Description

Name	Null?	Type
DEPTNO	NOT NULL	NUMBER(2)
DNAME	--	VARCHAR2(14)
LOC	--	VARCHAR2(13)

As the preceding table definitions show, and as is typical of a normalized relational schema, an employee's department number is not stored in the EMP table. Instead, one of the columns of EMP (DEPTNO) is a foreign key, which equals the primary key (DEPTNO) in DEPT.

However, the XML file equivalent has no similar notion of primary keys and foreign keys. Consequently, in the resulting XML file, the same data is represented in a hierarchy, thereby preserving the relationships by capturing the detail record embedded inside the master.

An XML element can contain elements that are either a primitive type (`string`, `decimal`), or a complex type, that is, another XML element. Therefore, an employee element can contain a department element.

The corresponding XML shows how the relationship is materialized, or shown inline. DEPTNO is removed from EMP, and instead you see the DEPT itself.

```
<EmpCollection>
  <Emp>
    <comm xsi:nil = "true" ></comm>
    <empno >7369.0</empno>
    <ename >SMITH</ename>
    <hiredate >1980-12-17T00:00:00.000-08:00</hiredate>
    <job >CLERK</job>
    <mgr >7902.0</mgr
    <sal >800.0</sal>
    <dept>
      <deptno >20.0</deptno>
      <dname >RESEARCH</dname>
      <loc >DALLAS</loc>
    </dept>
  </Emp>
    ...
</EmpCollection>
```

Materializing the relationship makes XML human readable and allows the data to be sent as one packet of information. No cycles are allowed in the XML file; therefore, an element cannot contain itself. This is handled automatically by the Oracle Database Adapter. However, you 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 Oracle Database Adapter does not print an element inside itself. The Oracle Database Adapter prints the following:

```
<Emp>
  <name>Bob</name>
  <spouse>
    <name>June</name>
  </spouse
</Emp>
```

But not:

```
<Emp>
  <name>Bob</name>
  <spouse>
    <name>June</name>
    <spouse>
      <name>Bob</name>
      <spouse>
        ...
      </spouse>
    </spouse>
  </spouse>
</Emp>
```

To avoid duplication, you can do the following:

- Import fewer tables. If you import only `EMP`, then `DEPT` does not appear.
- Remove the relationship between `EMP` and `DEPT` in the Adapter Configuration Wizard. This removes the relationship, but the foreign key column is put back.

In both these cases, the corresponding XML is as follows:

```
<EmpCollection>
  <Emp>
    <comm xsi:nil = "true" ></comm>
    <empno >7369.0</empno>
    <ename >SMITH</ename>
    <hiredate >1980-12-17T00:00:00.000-08:00</hiredate>
    <job >CLERK</job>
    <mgr >7902.0</mgr>
    <sal >800.0</sal>
    <deptno >20.0</deptno>
  </Emp>
   ...
</EmpCollection>
```

Either preceding solution is feasible only if returning foreign key suffices, as opposed to getting back the complete detail record in its entirety.

9.4.1.1 Relational Types to XML Schema Types

Table 9–6 shows how database data types are converted to XML primitive types when you import tables from a database.

Table 9–6 Mapping Database Data Types to XML Primitive Types

Database Type	XML Type (Prefixed with xs:)
VARCHAR, VARCHAR2, CHAR, NCHAR, NVARCHAR, NVARCHAR2, MEMO, TEXT, CHARACTER, CHARACTER VARYING, UNICHAR, UNIVARCHAR, SYSNAME, NATIONAL CHARACTER, NATIONAL CHAR, NATIONAL CHAR VARYING, NCHAR VARYING, LONG, CLOB, NCLOB, LONGTEXT, LONGVARCHAR, NTEXT	string
BLOB, BINARY, IMAGE, LONGVARBINARY, LONG RAW, VARBINARY, GRAPHIC, VARGRAPHIC, DBCLOB, BIT VARYING	base64Binary
BIT, NUMBER(1) DEFAULT 0, SMALLINT DEFAULT 0, SMALLINT DEFAULT 0	boolean
TINYINT, BYTE	byte
SHORT, SMALLINT	short
INT, SERIAL	int
INTEGER, BIGINT	integer
NUMBER, NUMERIC, DECIMAL, MONEY, SMALLMONEY, UNIQUEIDENTIFIER	decimal
FLOAT FLOAT16, FLOAT(16), FLOAT32, FLOAT(32), DOUBLE, DOUBLE PRECIS, REAL	double
TIME, DATE, DATETIME, TIMESTAMP, TIMESTAMP(6), SMALLDATETIME, TIMESTAMPTZ, TIMESTAMPLTZ, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE	dateTime

Essentially, NUMBER goes to DECIMAL, the most versatile XML data type for numbers, VARCHAR2 and CLOB to string, BLOB to base64Binary (to meet the plain-text requirement), and date types to dateTime.

Any type not mentioned in this discussion defaults to java.lang.String and xs:string. Time Stamp support is basic, because only the xs:dateTime format is supported. The BFILE type is specifically not supported.

> **Note:** The user-defined Object, Struct and VARRAY, and REF types are supported in 11*g*.

Because XML is plain text, BLOB and byte values are base 64/MIME encoded so that they can be passed as character data.

9.4.1.2 Mapping Any Relational Schema to Any XML Schema

The Oracle Database Adapter supports mapping any relational schema on any relational database to an XML schema, although not any XML schema of your choice,

because the Adapter Configuration Wizard generates the XML schema with no explicit user control over the layout of the elements. You can control how you map the schema in both the Adapter Configuration Wizard and later in TopLink Workbench. By pairing the Oracle Database Adapter with a transformation step, you can map any relational schema to any XML schema.

9.4.1.3 Querying over Multiple Tables

When executing a SQL `select` statement against multiple related tables there are the following three methods to build the SQL. These ways relate to how to pull in the detail records when the query is against the master record:

- Section 9.4.1.3.1, "Using Relationship Queries (TopLink Default)"
- Section 9.4.1.3.2, "Twisting the Original Select (TopLink Batch-Attribute Reading)"
- Section 9.4.1.3.3, "Returning a Single Result Set (TopLink Joined-Attribute Reading)"
- Section 9.4.1.3.4, "Comparison of the Methods Used for Querying over Multiple Tables"

The following sections contain an outline of these three methods and their comparison. When selecting rows from a single table there are no issues different from selecting from multiple tables.

9.4.1.3.1 Using Relationship Queries (TopLink Default)

Having selected a `Master` row, `TopLink` can always query separately to get all the details belonging to that `Master` table. These hidden queries (relationship queries) are cached in the `TopLink` metadata and must be prepared only once.

Consider the SQL statement in following sample scenario:

```
SELECT DIRECTOR, ..., VIEWER_RATING
       FROM MOVIES
WHERE RATING = 'A';
```

For each master, the SQL statement is as follows:

```
SELECT CRITIC, ..., TITLE
       FROM MOVIE_REVIEWS
WHERE (TITLE = ?)
```

It enables you to bring in all the data with 1 + n query executions, where n is the number of master rows returned by the first query.

This approach is safe but slow, as a large number of round trips to the database are required to pull in all the data.

For configuring using the relationship Queries (TopLink default) approach, you must edit `or_mappings.xml` outside of JDeveloper. In addition, change the batch-reading elements value to false.

9.4.1.3.2 Twisting the Original Select (TopLink Batch-Attribute Reading)

This is a default feature that allows `TopLink` to alter the original SQL `select` statement to read all the details in a second `select` statement, as shown in the following example:

```
SELECT DIRECTOR, ..., VIEWER_RATING
FROM MOVIES
WHERE RATING = 'A'
```

```
SELECT DISTINCT t0.CRITIC, ..., t0.TITLE
FROM MOVIE_REVIEWS t0, MOVIES t1
WHERE ((t1.RATING = 'A') AND (t0.TITLE = t1.TITLE))
```

By considering the original `select` statement in pulling in the details, a total of two (1 + 1 = 2) query executions must be performed.

Advantages

Batch attribute reading has the following advantages:

- All data read in two round trips to database
- The is a default feature in the 10.1.2.0.2 release

Disadvantages

Batch attribute reading has the following disadvantages:

- When using `maxTransactionSize` (on polling receive) or `maxRows` (on invoke select) to limit the number of rows loaded into memory at a time, these settings do not easily carry over to the batch attribute query. It is easier to work with a cursored result when there is only a single result set. (Multiple cursors can be used with difficulty, if the original query has an order by clause).
- `TopLink` can alter a SQL statement, only when it is in a format it can understand. If you use the hybrid SQL approach and set custom SQL for the root `select`, then `TopLink` cannot interpret that SQL to build the batch `select`.
- The `DISTINCT` clause is used on the batch query, to avoid returning the same detail twice if two masters happen to both point to it. The `DISTINCT` clause cannot be used when returning LOBs in the resultset.

Configuration

Configuration is on a per `1-1` or `1-M` mapping basis. By default, all such mappings since the 10.1.2.0.2 release have this property set. To configure, edit `or_mappings.xml` outside JDeveloper and edit the <batch-reading> elements to true (default) or false.

9.4.1.3.3 Returning a Single Result Set (TopLink Joined-Attribute Reading)

The detail tables are outer-joined to the original SQL `select` statement, returning both master and detail in a single result set, as shown in the following example:

```
SELECT DISTINCT t1.DIRECTOR, ..., t1.VIEWER_RATING, t0.CRITIC, ..., t0.TITLE
FROM MOVIE_REVIEWS t0, MOVIES t1
WHERE ((t1.RATING = 'A') AND (t0.TITLE (+) = t1.TITLE))
```

This requires one query execution in total.

Advantages

The advantages include the following:

- In case of using `maxTransactionSize` while polling, the benefits of dealing with a single cursor can be great.
- When following the hybrid SQL route and entering custom SQL statements, you only have to deal with a single SQL statement, whereas `TopLink` normally uses a series of additional hidden SQL statements to bring in related rows.
- `read` consistency: Enables you to read all related rows at the same time, and not at different instances in time for the different tables.

- Performance can be ideal as only a single round trip to the database is required, whereas batch attribute reading requires one for each table queried.

Disadvantages

There are some drawbacks, however, namely the cost of returning duplicate data. For example, consider that you read the `Master` and `Detail` tables; `Master` has 100 columns in each row, and `Detail` has 2 columns in each row. Each row in the table, `Master` also, typically has 100 related `Detail` rows.

With one query in each table, the result sets for the preceding example appears, as shown in the following example:

```
Master
Column1 column2 ….. column100

Master1 ...

Detail

Detail
Column1 column2
Detail1   ...
Detail2
...
Detail100 ...
```

In this example, 300 column values are returned as shown:

```
(columns in master + columns in detail x details per master) =
(          100            +           2               x             100
)  =  300
```

With one query for all tables, the result set appears, as shown in the following example:

```
Master                                          Detail

Column1 Column2 ... Column100                   Column1  Column2
Master1    ...                                  Detail1     ...
Master1    ...                                  Detail2     ...
Master1    ...                                  Detail100   ...
```

When there is one query for all tables, 10,200 column values are returned in a single result set, versus 300 in two result sets, as shown here:

```
((columns in master + columns in detail) x details per master) =
((          100     +            2     ) x       100         ) =  10,200
```

This can have a serious drain on network traffic and computation because 97 percent of the data returned is duplicate data. Also, if the master had two related tables `detail1` and `detail2` and there were 100 each in each master, then the number of column values returned would be over 10 million per master row.

In general, you can use the following simple formula to estimate the relative cost of returning all rows in a single result set:

```
        (Master columns + Detail1 columns + Detail2 columns + ... ) x
               Detail1s per Master x
               Detail2s per Master x ...
bloat = ____________________________________________________________

       (Master columns + Detail1 columns X Detail1s per Master +
```

```
                Detail2 columns X Detail2s per Master + ...)
```

For `1-1` relationships, this value is always 1, and if in the same example each master had two columns only and the details had 100 columns instead, and each master had only 3 or 4 details each, then the bloat would be

```
                (2 + 100) x 4        408
bloat =         ____________  = ___________  ~=  1
                (2 + 100 x 4)        402
```

Another disadvantage is that this setting could distort the meaning of the `maxRows` setting on an outbound select.

Configuration

To configure, select **Use Outer Joins to return a Single Result Set for both Master and Detail Tables** on the Adapter Configuration Wizard - Define Selection Criteria page.

Note:

When you create a SQL query such as the following by using the TopLink Expression Builder, the result may not be as expected:

```
SELECT DISTINCT t1.TABLE1_ID, t1.COLUMN_A FROM TABLE2 t0,
TABLE1 t1 WHERE ((t0.STATUS = 1) AND (t0.TABLE1_ID = t1.
TABLE1_ID))
```

The expected result for this query is that only rows with Table 1's and their owned Table 2's with status = 1 be returned.

However, what this query actually translates to is "table 1's, where any of its table 2's have status = 1," resulting in the return of table 1's that match the selection criteria, and `ALL` of the table 2's they own, including those with other statuses, whether or not their statuses =1. The `DISTINCT` keyword ensures the table 1's are not repeated and the join happens across table 2.

The misunderstanding happens in the way Toplink works. Through the Expression Builder, you can only specify a selection criteria for Table 1 and have no control over the Table 2's they own, this part is automatically done.

However, you can get the expected result by using either of the following two approaches:

1.) Query directly for table 2 using the selection criteria of status = 1, that is, do not go through table 1 and get the table 2's they own.

2.) Use direct (custom `SQL`), as shown in the following example:

```
SELECT TABLE1.TABLE1_ID, TABLE1.COLUMN_A, TABLE2.STATUS
FROM TABLE2, TABLE1 WHERE TABLE2.STATUS=1 AND TABLE1.
TABLE1_ID = TABLE2.TABLE1_ID
```

9.4.1.3.4 Comparison of the Methods Used for Querying over Multiple Tables

Superficially, returning a single result set looks best (1 query), followed by batch attribute reading (altering the `select` statement: 2 queries), and finally by default

relationship reading (n + 1 queries). However, there are several pitfalls to both of the more advanced options, as explained below:

Altering User-Defined SQL

If you specify `custom/hybrid` SQL, the `TopLink` cannot alter that SQL string to build the details `select`. For this reason, you must avoid using `hybrid` SQL and build `selects` using the wizards' `visual expression builder` as often as possible.

Show Me the SQL

The additional queries executed by `TopLink` in both, the default and the batch attribute reading cases can be somewhat of a mystery to users. For this reason, the `raw` SQL shown to users in the Adapter Configuration Wizard assumes returning a single result set, to make things clearer and also to improve manageability.

Returning Too Many Rows At Once

Databases can store vast quantities of information, and a common pitfall of `select` statements which return too much information at once. On a `DBAdapter receive`, a `maxTransactionSize` property can be set to limit the number of rows which are read from a cursored result set and processed in memory at a time. A similar `max-rows` setting exists for one time invoke `select` statements. However, this setting is very risky.

9.4.2 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 a readme file. 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.

This section includes the following topics:

- Section 9.4.2.1, "DML Operations"
- Section 9.4.2.2, "Polling Strategies"

9.4.2.1 DML Operations

Data manipulation language (DML) operations align with basic SQL `INSERT`, `UPDATE`, and `SELECT` operations. SQL `INSERT`, `UPDATE`, `DELETE`, and `SELECT` are all mapped to Web service operations of the same name. The `MERGE` is either an `INSERT` or `UPDATE`, based on the results of an existence check. A distinction is made between the data manipulation operations—called outbound writes—and the `SELECT` operations—called outbound reads. The connection between the Web service and the SQL for `merge` (the default for outbound write) and `queryByExample` are not as obvious as for basic SQL `INSERT`, `UPDATE`, and `SELECT`.

This section includes the following topics:

- Merge
- querybyExample
- Use Cases for Outbound Invoke Operations

Merge

`Merge` first reads the corresponding records in the database, calculates any changes, and then performs a minimal update. `INSERT`, `UPDATE`, and `MERGE` make the most

sense when you are thinking about a single row and a single table. However, your XML can contain complex types and map to multiple rows on multiple tables. Imagine a DEPT with many EMPS, each with an ADDRESS. In this case, you must calculate which of possibly many rows have changed and which to insert, update, or delete. If a specific row did not change or only one field changed, then the DML calls is minimal.

querybyExample

Unlike the SELECT operation, queryByExample does not require a selection criteria to be specified at design time. Instead, for each invoke, a selection criteria is inferred from an exemplary input XML record.

For instance, if the output xmlRecord is an employee record, and the input is a sample xmlRecord with lastName = "Smith", then on execution, all employees with a last name of Smith are returned.

A subset of queryByExample is to query by primary key, which can be implemented by passing in sample XML records where only the primary key attributes are set.

Use queryByExample when you do not want to create a query using the visual query builder and want the flexibility of allowing the input record to share the same XML schema as the output records.

The queryByExample operation is slightly less performant because a new SELECT must be prepared for each execution. This is because the attributes that are set in the example XML record can vary each time, and therefore the selection criteria vary.

Input xmlRecord:

```
<Employee>
      <id/>
      <lastName>Smith</lastName>
</Employee>
```

Output xmlRecord:

```
<EmployeeCollection>
      <Employee>
         <id>5</id>
         <lastName>Smith</lastName>
         ....
      </Employee>
      <Employee>
         <id>456</id>
         <lastName>Smith</lastName>
         ....
      </Employee>
</EmployeeCollection>
```

Use Cases for Outbound Invoke Operations

Outbound invoke operations are demonstrated in the following tutorial files:

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

- PureSQLSelect
- QueryByExample

To obtain these files, access the Oracle SOA Sample Code site.

Use Cases for Pure SQL

A new option in 10.1.3.1 enables you to specify any arbitrary SQL string, and an XML representing the inputs and outputs to the SQL is generated. Pure SQL operations are demonstrated in the following tutorial files:

- UpdateAll
- SelectCount
- SelectGroupBy
- SelectStar

To obtain these files, access the Oracle SOA Sample Code site.

Advanced Use Cases for Outbound Invoke Operations

Advanced outbound invoke operations are demonstrated in the following tutorial files:

- InsertWithClobs
- XAInsert
- NativeSequencingInsert

To obtain these files, access the Oracle SOA Sample Code site.

9.4.2.2 Polling Strategies

The inbound receive enables you to listen to and detect events and changes in the database, which in turn can be the initiators of a business process. This is not a one-time action, but rather an activation. A polling thread is started, which polls a database table for new rows or events.

Whenever a new row is inserted into the MOVIES table, the polling operation raises it to the SCA Run Time. The strategy is to poll every record once. The initial SELECT has to be repeated over time, to receive the rows that exist at the start and all new rows as they are inserted over time. However, a new row once read is not likely to be deleted, and therefore can possibly be read repeatedly with each polling.

The various ways to poll for events, called polling strategies, also known as after-read strategies or publish strategies, range from simple and intrusive to sophisticated and 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 operations that you can perform after the data is read from the database. This section also discusses the strategies and factors to help you determine which strategy to employ for a particular situation:

- Delete the Row(s) that were Read
- Update a Field in the [Table_Name] Table (Logical Delete)
- Update a Sequencing Table
- Update an External Sequencing Table on a Different Database

- Control Table Strategy
- Update a Sequencing File

Delete the Row(s) that were Read

Choose this operation to employ the physical delete polling strategy. This operation polls the database table for records and deletes them after processing. Use this strategy to capture events related to `INSERT` operations and cannot capture database events related to `DELETE` and `UPDATE` operations on the parent table. This strategy cannot be used to poll child table events. This strategy allows multiple adapter instances to go against the same source table. There is zero data replication.

Preconditions: You must have deletion privileges on the parent and associated child tables to use the delete polling strategy. Table 9–7 describes the requirements for using the delete polling strategy.

Table 9–7 Delete Polling Strategy Preconditions

Requirements Met	Conflicts With
Poll for inserts	No delete on source
Shallow delete	No updates on source
Cascading delete	Poll for updates
Minimal SQL	Poll for deletes
Zero data replication	Poll for child updates
Default	--
Allows raw SQL	--
Concurrent polling	--

> **Note:** In *Shallow delete* and *Cascading delete*, the delete operation can be configured to delete the top-level row, to cascade all, or to cascade on a case-by-case basis.
>
> *Concurrent polling* can be configured for both delete and logical delete polling strategies.

Configuration: You can configure the delete polling strategy to delete the top-level row, to cascade all, or to cascade on a case-by-case basis. This strategy enables deleting only the parent rows and not the child rows, cascaded deletes, and optional cascaded deletes, determined on a case-by-case basis. You can configure the polling interval for performing an event publish at design time.

Delete Cascade Policy: The optional advanced configuration is to specify the cascade policy of the `DELETE` operation. For instance, after polling for an employee with an address and many phone numbers, the phone numbers are deleted because they are privately owned (default for one-to-many), but not the address (default for one-to-one). This can be altered by configuring `or_mappings.xml`, as in the following example:

```
<database-mapping>
   <attribute-name>orders</attribute-name>
   <reference-class>taxonomy.Order</reference-class>
   <is-private-owned>true</is-private-owned>
```

You can also configure the activation itself to delete only the top level (master row) or to delete everything.

A receive operation appears in an inbound JCA as follows:

```
<connection-factory location="eis/DB/Connection1" UIConnectionName="Connection1"
adapterRef=""/>
  <endpoint-activation portType="dd_ptt" operation="receive">
    <activation-spec className="oracle.tip.adapter.db.DBActivationSpec">
      <property name="DescriptorName" value="dd.Emp"/>
      <property name="QueryName" value="ddSelect"/>
      <property name="MappingsMetaDataURL" value="dd-or-mappings.xml"/>
      <property name="PollingStrategy" value="LogicalDeletePollingStrategy"/>
      <property name="MarkReadColumn" value="STATUS"/>
      <property name="MarkReadValue" value="PROCESSED"/>
      <property name="MarkReservedValue" value="RESERVED-1"/>
      <property name="MarkUnreadValue" value="UNPROCESSED"/>
      <property name="PollingInterval" value="5"/>
      <property name="MaxRaiseSize" value="1"/>
      <property name="MaxTransactionSize" value="10"/>
      <property name="ReturnSingleResultSet" value="false"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

Update a Field in the [*Table_Name*] Table (Logical Delete)

Choose this operation to employ the logical delete polling strategy. This strategy involves updating a special field on each row processed and updating the `WHERE` clause at 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 Oracle Database Adapter.

Preconditions: You must have the logical delete privilege or a one-time alter schema (add column) privilege on the source table. Table 9–8 describes the requirements for using the logical delete polling strategy.

Table 9–8 Logical Delete Polling Strategy Preconditions

Requirements Met	Conflicts With
Poll for inserts	No updates on source
No delete on source	Poll for deletes
Minimal SQL	--
Zero data replication	--
Minimal configuration	--
Allows raw SQL	--
Poll for updates	--
Poll for child updates	--
Concurrent polling	--

Note: The requirements of the following are met, as follows:

- **Poll for updates**: By adding a trigger
- **Poll for child updates**: By adding a trigger
- **Concurrent polling**: By specifying additional mark unread and reserved values.

Configuration: The logical delete polling strategy requires minimal configuration. You must specify the mark read column and the value that indicates a processed record.

A receive operation appears in an inbound WSDL as follows:

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

Given the configuration for logical delete, the Oracle Database Adapter appends the following `WHERE` clause to every polling query:

```
AND (STATUS IS NULL) OR (STATUS <> 'PROCESSED')
```

Database Configuration: A status column on the table being polled must exist. If it does not exist already, you can add one to an existing table.

Support for Polling for Updates: Given that rows are not deleted with each read, it is possible to repetitively read a row multiple times. You must add a trigger to reset the mark read field whenever a record is changed, as follows:

```
create trigger Employee_modified
before update on Employee
for each row
begin
   :new.STATUS := 'MODIFIED';
end;
```

Support for Concurrent Access Polling: Just as a single instance should never process an event more than once, the same applies to a collection of instances. Therefore, before processing a record, an instance must reserve that record with a unique value. Again, the status column can be used:

```
<operation name="receive">
   <jca:operation
      ActivationSpec="oracle.tip.adapter.db.DBActivationSpec"
         …
      PollingStrategyName="LogicalDeletePollingStrategy"
      MarkReadField="STATUS"
      MarkUnreadValue="UNPROCESSED"
      MarkReservedValue="RESERVED${IP-2}-${weblogic.Name-1}-${instance}"
      MarkReadValue="PROCESSED"
```

The polling query instead appears, as shown in the following example:

```
Update EMPLOYE set STATUS = 'RESERVED65-1-1' where (CRITERIA) AND (STATUS =
'UNPROCESSED');
```

```
Select … from EMPLOYEE where (CRITERIA) AND (STATUS = 'RESERVED65-1-1');
```

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

```
Update EMPLOYEE set STATUS = 'PROCESSED' where (CRITERIA) AND (STATUS =
'RESERVED65-1-1');
```

Update a Sequencing Table

Choose this operation to employ the sequencing table: last-read Id strategy. This polling strategy involves using a helper table to remember a sequence value. The source table is not modified; instead, rows that have been read in a separate helper table are recorded. A sequence value of `1000`, for example, means that every record with a sequence less than that value have been processed. Because many tables have some counter field that is always increasing and maintained by triggers or the application, this strategy can often be used for noninvasive polling. No field on the processed row must be modified by the Oracle Database Adapter.

Native sequencing with a 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 you can use a sequencing strategy such as the `sequence` field to order the rows in a sequence for processing. When the rows are ordered in a line, the Oracle Database Adapter knows which rows are processed and which are not with a single unit of information.

Preconditions: You must have a sequencing table or create table privilege on the source schema. The source table has a column that is monotonically increasing with every `INSERT` (an Oracle native sequenced primary key) or `UPDATE` (the last-modified timestamp). Table 9–9 describes the requirements for using the sequencing polling strategy.

Table 9–9 Sequencing Polling Strategy Preconditions

Requirements Met	Conflicts With
Poll for inserts	Poll for deletes
Poll for updates	Allows raw SQL
No delete on source	Concurrent polling
No updates on source	--
One extra SQL select	--
Zero data replication	--
Moderate configuration	--
Poll for child updates	--

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

```
<adapter-config name="ReadS" adapter="Database Adapter"
xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
    <connection-factory location="eis/DB/DBConnection1"
UIConnectionName="DBConnection1" adapterRef=""/>
  <endpoint-activation portType="ReadS_ptt" operation="receive">
    <activation-spec className="oracle.tip.adapter.db.DBActivationSpec">
      <property name="DescriptorName" value="ReadS.PerfMasterIn"/>
      <property name="QueryName" value="ReadSSelect"/>
```

```
      <property name="MappingsMetaDataURL" value="ReadS-or-mappings.xml"/>
      <property name="PollingStrategy" value="SequencingPollingStrategy"/>
      <property name="SequencingTable" value="PC_SEQUENCING"/>
      <property name="SequencingColumn" value="PK"/>
      <property name="SequencingTableKeyColumn" value="TABLE_NAME"/>
      <property name="SequencingTableValueColumn" value="LAST_READ_ID"/>
      <property name="SequencingTableKey" value="PERF_MASTER_IN"/>
      <property name="PollingInterval" value="60"/>
      <property name="MaxRaiseSize" value="1"/>
      <property name="MaxTransactionSize" value="10"/>
      <property name="ReturnSingleResultSet" value="false"/>
    </activation-spec>
  </endpoint-activation>
</adapter-config>
```

The sequencing field type can be excluded if it is actually a number.

Database Configuration: A sequencing table must be configured once for a given database. Multiple processes can share the same table. Given the `ActivationSpec` specified in the preceding example, the `CREATE TABLE` command looks as follows:

```
CREATE TABLE SEQUENCING_HELPER
(
TABLE_NAME VARCHAR2(32) NOT NULL,
LAST_READ_DATE DATE
)
;
```

Polling for Updates: In the preceding example, the polling is for new objects or updates, because every time an object is changed, the modified time is updated.

A sample trigger to set the modified time on every `insert` or `update` is as follows:

```
create trigger Employee_modified
before insert or update on Employee
for each row
begin
  :new.modified_date := sysdate;
end;
```

Using a Sequence Number: A sequence number can be used for either insert or update polling. Native sequencing returns monotonically increasing primary keys, when an increment by 1 is used. You can also use the sequence number of a materialized view log.

Update an External Sequencing Table on a Different Database

Choose this operation to employ the sequencing table: last updated strategy. This polling strategy involves using a helper table to remember a `last_updated` value. A `last_updated` value of `2005-01-01 12:45:01 000`, for example, means that every record last updated at that time or earlier have been processed. Because many tables have rows with a `last_updated` or `creation_time` column maintained by triggers or the application, this strategy can often be used for noninvasive polling. Fields on the processed row never require modification by the Oracle Database Adapter.

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

See Update a Sequencing Table for information about preconditions and configuration.

Update a Sequencing File

This strategy works similar to Update an External Sequencing Table on a Different Database, the only difference is that the control information is stored in a file instead of a table.

Control Table Strategy

Choose this operation to employ the control table polling strategy. This polling strategy involves using a control table to store the primary key of every row that has yet to be processed. With a natural join between the control table and the source table (by primary key), polling against the control table is practically the same as polling against the source table directly. However, an extra layer of indirection allows the following:

- Destructive polling strategies such as the delete polling strategy can be applied to rows in the control table alone while shielding any rows in the source table.
- Only rows that are meant to be processed have their primary key appear in the control table. You can use information that is not in the rows themselves to control which rows to process (a good `WHERE` clause may not be enough).
- The entire row is not copied to a control table, and any structure under the source table, such as detail rows, can also be raised without copying.

Streams and materialized view logs make good control tables.

Preconditions: You must have the create/alter triggers privilege on the source table. Table 9–10 describes the requirements for using the control table polling strategy.

Table 9–10 Control Table Polling Strategy Preconditions

Requirements Met	Conflicts With
Poll for inserts	Advanced configuration: the native XML from the database has control header, and triggers are required.
Poll for updates	--
Poll for deletes	--
Poll for child updates	Minimal data replication (primary keys are stored in control table)
No delete on source	--
No updates on source	--
No extra SQL selects	--
Concurrent polling	--
Allows raw SQL	--
Auditing	--

Using triggers, whenever a row is modified, an entry is added to a control table, containing the name of the master table, and the primary keys. At design time, the control table is defined to be the root table, with a one-to-one mapping to the master table, based on the matching primary keys. The control table can contain extra control information, such as a time stamp, and operation type (`INSERT`, `UPDATE`, and so on).

The delete polling strategy is useful with this setup. It is important to keep the control table small, and if the option `shouldDeleteDetailRows="false"` is used, then

only the control rows are deleted, giving you a nondestructive delete (the `DELETE` is not cascaded to the real tables).

It is possible to reuse the same control table for multiple master tables. In TopLink, you can map the same table to multiple descriptors by mapping the control table as one abstract class with multiple children. Each child has a unique one-to-one mapping to a different master table. The advantage of this approach is that you can specify for each child a class indicator field and value so that you do not need an explicit `WHERE` clause for each polling query.

The following are sample triggers for polling for changes both to a department table and any of its child employee rows:

```
CREATE OR REPLACE TRIGGER EVENT_ON_DEPT
   AFTER INSERT OR UPDATE ON DEPARTMENT
   REFERENCING NEW AS newRow
   FOR EACH ROW
   DECLARE X NUMBER;
BEGIN
   SELECT COUNT(*) INTO X FROM DEPT_CONTROL WHERE (DEPTNO = :newRow.DEPTNO);
   IF X = 0 then
   insert into DEPT_CONTROL values (:newRow. DEPTNO);
   END IF;
END;
CREATE OR REPLACE TRIGGER EVENT_ON_EMPLOYEE
   AFTER INSERT OR UPDATE ON EMPLOYEE
   REFERENCING OLD AS oldRow NEW AS newRow
   FOR EACH ROW
   DECLARE X NUMBER;
BEGIN
   SELECT COUNT(*) INTO X FROM DEPT_CONTROL WHERE (DEPTNO = :newRow.DEPTNO);
   IF X = 0 then
   INSERT INTO DEPT_CONTROL VALUES (:newRow.DEPTNO);
   END IF;
   IF (:oldRow.DEPTNO <> :newRow.DEPTNO) THEN
      SELECT COUNT(*) INTO X FROM DEPT_CONTROL WHERE (DEPTNO = :oldRow.DEPTNO);
      IF (X = 0) THEN
         INSERT INTO DEPT_CONTROL VALUES (:oldRow.DEPTNO);
      END IF;
   END IF;
END;
```

Use Cases for Polling Strategies

Polling strategies are demonstrated in the following tutorials:

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

To obtain these files, access the Oracle SOA Sample Code site.

Advanced Use Cases for Polling Strategies

Advanced polling strategies are demonstrated in the following tutorials:

- `DistributedPolling`

- PollingExternalSequencing
- PollingFileSequencingStrategy
- PollingForChildUpdates
- PollingNoAfterReadStrategy
- PollingOracleSCNStrategy
- PollingPureSQLOtherTableInsert
- PollingPureSQLSysdateLogicalDelete
- PollingWithParameters

To obtain these files, access the Oracle SOA Sample Code site.

9.5 Deployment

The Oracle Database Adapter comes deployed to the application server by the install. It contains a single adapter instance entry eis/DB/SOADemo, which points to the data source jdbc/SOADataSource. The connection information to the database is inside the data source definition.

When deploying a SOA project that uses the OracleAS Adapter for Databases, you might have to add a new adapter instance and restart the application server first. This could be because you want to point to a database other than the one referred in jdbc/SOADataSource, or because you chose a name for the adapter instance that does not yet exist. For instance, if you create a connection in JDeveloper named Connection1, then by default the DB Adapter service points to eis/DB/Connection1, as shown in Figure 9–7.

You can also check which adapter instance the service is pointing to by looking at the db.jca file, as shown in the following code snippet:

```
<connection-factory location="eis/DB/Connection1" UIConnectionName="Connection1"
adapterRef="" />
```

In the preceding example, the location is the JNDI name of the adapter instance at run time, and UIConnectionName is the name of the connection used in JDeveloper.

You can create a DB Adapter instance through the Oracle WebLogic Server Administration Console, as mentioned in Section 2.18, "Adding an Adapter Connection Factory" or by directly editing the weblogic-ra.xml file. Following these steps are screenshots that show how to create an adapter instance through the Oracle WebLogic Administration Console. The following are the steps to edit weblogic-ra.xml:

1. Search fmwhome/ for DbAdapter.rar.
2. Unzip the file.
3. Edit META-INF/weblogic-ra.xml (and possibly ra.xml.)
4. Jar the file again.
5. Restart the application server.

The following is a sample adapter instance in weblogic-ra.xml:

```
<connection-instance>
  <jndi-name>eis/DB/SOADemo</jndi-name>
    <connection-properties>
       <properties>
```

```
            <property>
              <name>xADataSourceName</name>
              <value>jdbc/SOADataSource</value>
                </property>
                <property>
                  <name>dataSourceName</name>
                  <value> </value>
                </property>
                <property>
                  <name>platformClassName</name>
                  <value>Oracle10Platform</value>
                </property>
             </properties>
          </connection-properties>
</connection-instance>
```

The four mandatory properties are: `jndi-name`, `xADataSourceName`, `dataSourceName`, and `platformClassName`. The `jndi-name` property must match the location attribute in the `db.jca` file, and is the name of the adapter instance.

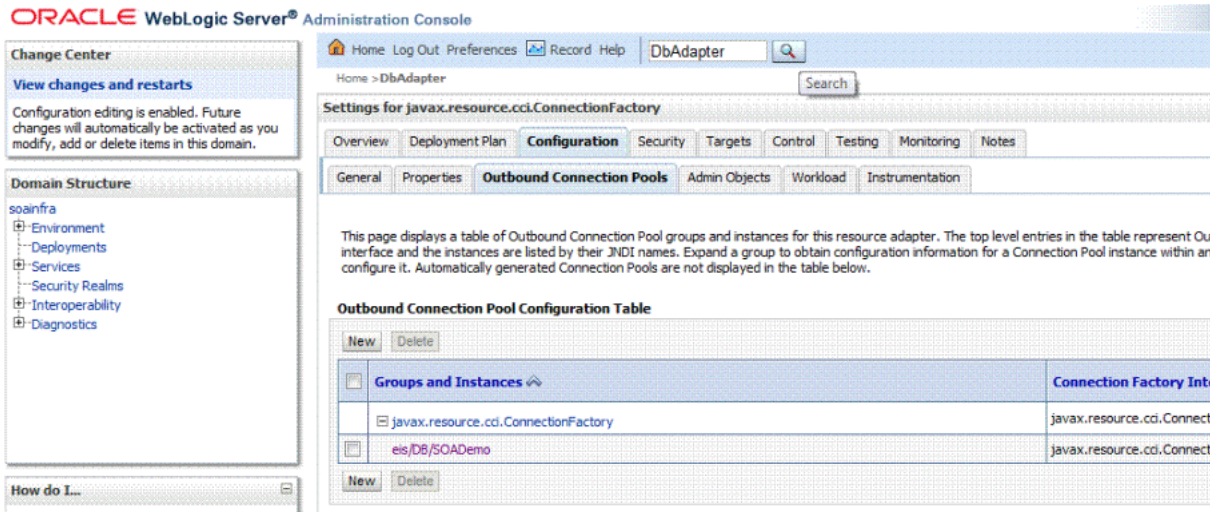
The `xADataSourceName` property is the name of the underlying data source (which has the connection information).

The `platformClassName` indicates which SQL to generate. For information about `PlatformClassName`, see Table 9–11, " Database Platform Names".

The following screenshots show how to edit Database Adapter properties using the Oracle WebLogic Administation Console

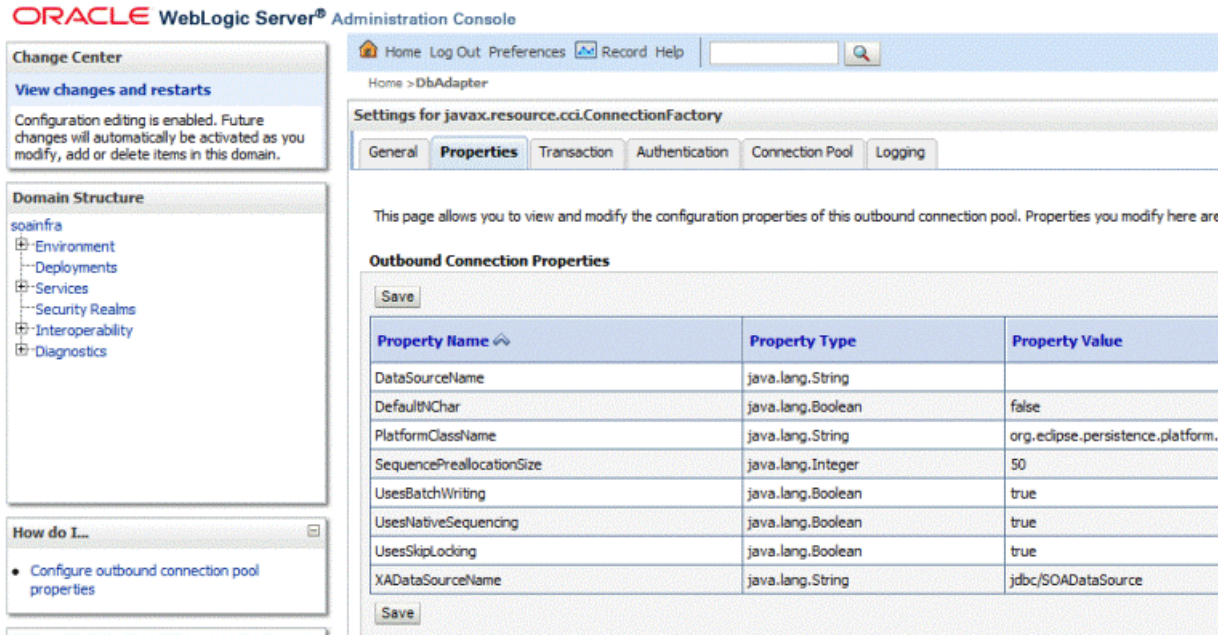
The first screenshot shows navigation to the Outbound Connectio Pools within the WebLogic Administration Console. This is the actual Database Adapter Configuration, from where you can go to the subsequent page to edit the Database Adapter properties.

Figure 9–29 The Outbound Connection Pools Tab of the WebLogic Console



The second screenshot shows editing properties from the WebLogic Console that you edit accordingly andas needed. Name, Type and Value are displayed on a per-property basis.

Figure 9–30 Database Adapter Properties in the Oracle WebLogic Administration Console



Most Common Mistakes

The following are the two most common mistakes with deployment:

- Not creating an adapter instance entry that matches the location attribute in your `db.jca` file (or not creating one at all.)
- Setting the location attribute in the `db.jca` file to the name of the data source directly.
- Not changing platformClassName when connecting to a database other than Oracle.

For the latter, there is a level of indirection in that you give the name of the adapter instance (`eis/DB/...`), which itself points to the data source pool (`jdbc/...`). It is a common mistake to miss this indirection and give the name `jdbc/...` directly in the location attribute.

Data Source Configuration

For the relevant Data Source configuration for your application server, see Section 9.6, "JDBC Driver and Database Connection Configuration." When configuring an Oracle data source, ensure that you use the `thin XA` option.

Additional Adapter Instance Properties

This section briefly describes additional properties in the DB Adapter instance beyond `xADataSourceName`, `dataSourceName`, and `platformClassName`. When adding a property to `weblogic-ra.xml`, you must ensure that the property is also declared in `ra.xml` (also in `DbAdapter.rar`). For example, the following is a code snippet of the `ra.xml` file for the property `xADataSourceName` in `weblogic-ra.xml`:

```
<config-property>
<config-property-name>xADataSourceName </config-property-name>
<config-property-type>java.lang.String</config-property-type>
<config-property-value></config-property-value>
</config-property>
```

For information about the Oracle Database Adapter instance properties, see Appendix A.5, "Oracle Database Adapter Properties." Apart from the properties mentioned there, you can also add the properties listed in the following table:

Property Name	Type
usesNativeSequencing	Boolean
usesSkipLocking	Boolean
usesStringBinding	Boolean
usesByteArrayBinding	Boolean
usesStreamsForBinding	Boolean
eventListenerClass	String
logTopLinkAll	Boolean
maxBatchWritingSize	Integer
nonRetriableSQLErrorCodes	String
shouldOptimizeDataConversion	Boolean
shouldTrimStrings	Boolean
driverClassName	String
sequencePreallocationSize	Integer
tableQualifier	String
usesBatchWriting	Boolean
usesNativeSQL	Boolean

The preceding properties appear in the `oracle.toplink.sessions.DatabaseLogin` object. See `TopLink` API reference information on `DBConnectionFactory` Javadoc and `DatabaseLogin` Javadoc at `http://download.oracle.com/docs/cd/B10464_02/web.904/b10491/index.html`.

9.5.1 Deployment with Third Party Databases

Table 9–11 lists databases and their advanced properties, which are database platform variables. Set the `platformClassName` name to a listed variable. Setting `platformClassName` is mandatory if you are using an advanced database.features that are not uniform across databases, such as native sequencing or stored procedures.

As an example, to execute a stored procedure on DB2 versus SQL Server, the DbAdapter must generate and send different SQL.Use the example below for use with the SQLServer Platform :

```
execute <procedure> @<arg1>=? ...
```

when using the DB2 Platform:

```
call <procedure>(?, ...)
```

The `platformClassName` setting indicates which SQL to generate. Since most databases offer non-uniform features (that is, variants on the ANSI SQL 92 language specification), it is safest to configure `platformClassName` accurately.The default value is `Oracle10Platform`, and should be changed to the appropriate variable if you are connecting to a different database vendor.

Note: Providing the qualified class name with package is not necessary if it starts with org.eclipse.persistence.platform.database

Table 9–11 Database Platform Names

Database	PlatformClassName
Oracle10+ (including 11*g*)	`org.eclipse.persistence.platform.database.Oracle10Platform`
Oracle9+ (optional):	`org.eclipse.persistence.platform.database.Oracle9Platform`
Oracle8	`org.eclipse.persistence.platform.database.Oracle8Platform`
Oracle7	`org.eclipse.persistence.platform.database.OraclePlatform`
DB2	`org.eclipse.persistence.platform.database.DB2Platform`
DB2 on AS400e	`oracle.tip.adapter.db.toplinkext.DB2AS400Platform`
Informix	`org.eclipse.persistence.platform.database.InformixPlatform`
SQLServer	`org.eclipse.persistence.platform.database.SQLServerPlatform`
MySQL	`org.eclipse.persistence.platform.database.MySQLPlatform`
Any other database	`org.eclipse.persistence.platform.database.DatabasePlatform`

9.6 JDBC Driver and Database Connection Configuration

In this release, Oracle JCA Adapters are certified against the following third-party databases using Oracle WebLogic Server Type 4 JDBC drivers:

- Microsoft SQL Server 2005 and 2008 (all SP levels included)
- Sybase 15
- Informix 11.5
- DB2 9.7 and later FixPaks
- MySQL 5.x+

Note: Only major databases and versions are certified. Working with other databases should be feasible when they provide a working JDBC driver, and you rely on core ANSI SQL relational features, such as Create, Read, Update, and Delete (CRUD) operations on tables and views. Issues tend to be more prevalent due to the fact that not all JDBC drivers implement database metadata introspection the same way. However, it should be possible to import matching tables on a certified database and then point to the uncertified database at runtime. The information provided in this section for uncertified databases is meant as a guide only.

For more information, see the following topics:

- Section 9.6.1, "Creating a Database Connection Using a Native or Bundled Oracle WebLogic Server JDBC Driver"
- Section 9.6.2, "Creating a Database Connection Using a Third-Party JDBC Driver"
- Section 9.6.3, "Summary of Third-Party JDBC Driver and Database Connection Information"
- Section 9.6.4, "Location of JDBC Driver JAR Files and Setting the Class Path"

9.6.1 Creating a Database Connection Using a Native or Bundled Oracle WebLogic Server JDBC Driver

To create a database connection when using a native or bundled Oracle WebLogic Server JDBC driver:

1. Ensure that the appropriate JDBC driver JAR files are installed and set the class path.

   For more information, see:

   - *Oracle Fusion Middleware Configuring and Managing JDBC Data Sources for Oracle WebLogic Server*
   - *Oracle Fusion Middleware Type 4 JDBC Drivers for Oracle WebLogic Server*
   - *Oracle Fusion Middleware Programming JDBC for Oracle WebLogic Server*

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

   The New Gallery page is displayed.

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

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

4. Select **Database Connection**, and then click **OK**.

   The Create Database Connection page is displayed.

5. For **Create Connection In**, select **IDE Connections**.

6. Enter a name for this connection in the Connection Name field.

   For example, `SQLServer`.

7. Select the appropriate driver from the **Connection Type** menu.

8. Enter your credentials (such as user name, password, and role, if applicable).

9. Enter your connection information.

   For example,
   `jdbc:sqlserver://HOST-NAME:PORT;databaseName=DATABASE-NAME`

   For more information, see:

   - Table 9–12, " Database Driver Selection (from Weblogic Server Console)"
   - Sample entries in the deployment descriptor file (`weblogic-ra.xml`).

10. Click **Test Connection**.
11. If the connection is successful, click **OK**.

9.6.2 Creating a Database Connection Using a Third-Party JDBC Driver

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

1. Install the appropriate JDBC driver JAR files and set the class path.

   For more information, see Section 9.6.4, "Location of JDBC Driver JAR Files and Setting the Class Path".

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

   The New Gallery page is displayed.

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

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

4. Select **Database Connection**, and then click **OK**.

   The Create Database Connection page is displayed.

5. For **Create Connection In**, select **IDE Connections**.
6. Enter a name for this connection in the Connection Name field.

   For example, `SQLServer`.

7. Select **Generic JDBC** from Connection Type.
8. Enter your user name, password, and role information.
9. Click **New** for Driver Class.

   The Register JDBC Driver dialog is displayed.

   Perform Steps 10, 11 and 19 in the Register JDBC Driver dialog.

10. Enter the driver name (for example, *some*`.jdbc.`*`Driver`*) for **Driver Class**.

    For example, `com.microsoft.sqlserver.jdbc.SQLServerDriver`.

11. Click **Browse** for Library.

    The Select Library dialog is displayed.

    Perform Steps 12 and 18 in the Select Library dialog.

12. Click **New** to create a library.

    The Create Library dialog is displayed.

    Perform Steps 13 through 17 in the Create Library dialog.

13. Specify a name in the **Library Name** field.

For example, `SQL Server JDBC`.

14. Click **Class Path**, and then click **Add Entry** to add each JAR file of your driver to the class path.

    The Select Path Entry dialog is displayed.

15. Select a JDBC class file and click **Select**.

    For example, select `sqljdbc.jar`.

16. Click **OK** when you have added all the class files to the Class Path field.
17. Click **OK** to exit the Create Library dialog.
18. Click **OK** to exit the Select Library dialog.
19. Click **OK** to exit the Register JDBC Driver dialog.
20. Enter your connection string name for **JDBC URL** and click **Next**.

    For example,
    `jdbc:sqlserver://`*`HOST-NAME`*`:`*`PORT`*`;databaseName=`*`DATABASE-NAME`*

    For more information, see:

    - Table 9–12, " Database Driver Selection (from Weblogic Server Console)"
    - Sample entries in the deployment descriptor file (`weblogic-ra.xml`).

21. Click **Test Connection**.
22. If the connection is successful, click **OK**.

9.6.3 Summary of Third-Party JDBC Driver and Database Connection Information

Table 9–12, " Database Driver Selection (from Weblogic Server Console)" summarizes the connection information for common third-party databases.

For information about `PlatformClassName`, see Table 9–11, " Database Platform Names".

For more information, see:

- Section 9.6.3.1, "Using a Microsoft SQL Server"
- Section 9.6.3.2, "Using a Sybase Database"
- Section 9.6.3.3, "Using an Informix Database"
- Section 9.6.3.4, "Using an IBM DB2 Database"
- Section 9.6.3.5, "Using a MySQL Database"

Table 9–12 Database Driver Selection (from Weblogic Server Console)

Database	JDBC Driver
Microsoft SQL Server	▪ Oracle's MS SQL Server Driver (Type 4 XA) ▪ Oracle's MS SQL Server Driver (Type 4)
Sybase	▪ Oracle's Sybase Driver (Type 4 XA) ▪ Oracle's Sybase Driver (Type 4)

Table 9–12 (Cont.) Database Driver Selection (from Weblogic Server Console)

Database	JDBC Driver
Informix	■ Oracle's Informix Driver (Type 4 XA) ■ Oracle's Informix Driver (Type 4)
IBM DB2	■ Oracle's DB2 Driver (Type 4 XA) ■ Oracle's DB2 Driver (Type 4)
MySQL	MySQL's Driver (Type 4) Versions: using `com.mysql.jdbc.Driver`

9.6.3.1 Using a Microsoft SQL Server

You must note the following when connecting to a SQL Server database:

- User name and password
  - SQL Server 2005 installs with Windows authentication as the default. Therefore, you do not log in with a user name and password; rather, your Windows user account either has privilege or does not. JDBC requires you to provide a user name and password.
- Connect string

  From the `sqlcmd` login, you can deduce what your connect string is, as in the following examples:

  Example 1:

  ```
  sqlcmd
  1>
  jdbc:microsoft:sqlserver://localhost:1433
  ```

  Example 2:

  ```
  sqlcmd -S user.mycompany.com\SQLExpress
  1>
  jdbc:microsoft:sqlserver://user.mycompany.com\SQLExpress:1433
  ```

  Example 3:

  ```
  sqlcmd -S user.mycompany.com\SQLExpress -d master
  1>
  jdbc:microsoft:sqlserver://user.mycompany.com\SQLExpress:1433;databasename=
  master
  ```

  A full URL is as follows:

  ```
  jdbc:microsoft:sqlserver://serverName[\instanceName]:tcpPort[;SelectMethod=curs
  or][;databasename=databaseName]
  ```

- Database name

  If you must explicitly supply the database name, but do not know it, go to

  ```
  C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data
  ```

  If you see a file named `master.mdf`, then one database name is `master`.

- TCP port

  Ensure that SQL Server Browser is running and that your SQL Server service has TCP/IP enabled and is listening on static port 1433. Disable dynamic ports. In SQL

Native Client Configuration/Client Protocols, ensure that TCP/IP is enabled and that the default port is 1433.

- JDBC drivers

  You must download the JDBC drivers separately. From `www.microsoft.com`, click **Downloads** and search on *jdbc*. You can also try using the DataDirect driver.

9.6.3.2 Using a Sybase Database

This section includes the following topics:

- Section 9.6.3.2.1, "Using a Sybase JConnect JDBC Driver"

9.6.3.2.1 Using a Sybase JConnect JDBC Driver

URL: `jdbc:weblogic:sybase://SERVER-NAME:PORT;databaseName=DATABASE-NAME`

Driver Class: `weblogic.jdbc.sybase.SybaseDriver`

Driver Jar: `jConnect-6_0\classes\jconn3.jar`

For information about the Sybase JConnect JDBC driver, refer to the following link:

`http://www.sybase.com/products/middleware/jconnectforjdbc`.

9.6.3.3 Using an Informix Database

This section includes the following topics:

- Section 9.6.3.3.1, "Using an Informix JDBC Driver"

9.6.3.3.1 Using an Informix JDBC Driver

URL: `jdbc:informix-sqli://`*`HOST-NAME-OR-IP`*`:`*`PORT-OR-SERVICE-NAME`*`/`*`DATABASE-NAME`*`:INFORMIXSERVER=`*`SERVER-NAME`*

Driver Class: `com.informix.jdbc.IfxDriver`

Driver Jar: `ifxjdbc.jar`

For information about the Informix JDBC driver, refer to the following link:

`http://www-01.ibm.com/software/data/informix/tools/jdbc/`.

9.6.3.4 Using an IBM DB2 Database

This section includes the following topics:

- Section 9.6.3.4.1, "IBM DB2 Driver"
- Section 9.6.3.4.2, "JT400 Driver (AS400 DB2)"
- Section 9.6.3.4.3, "IBM Universal Driver"

9.6.3.4.1 IBM DB2 Driver

URL: `jdbc:db2:localhost:NAME`

Driver Class: `com.ibm.db2.jcc.DB2Driver`

Driver Jar (v8.1): `db2jcc.jar`, `db2jcc_javax.jar`, `db2jcc_license_cu.jar`

For information about DataDirect driver, refer to the following link:

`http://www.datadirect.com/techres/jdbcproddoc/index.ssp`

9.6.3.4.2 JT400 Driver (AS400 DB2)

URL: `jdbc:as400://`*`hostname`*`;translate binary=true`

Driver Class: `com.ibm.as400.access.AS400JDBCDriver`

Driver Jar: `jt400.jar`

For correct character set translation, use `translate binary=true`.

9.6.3.4.3 IBM Universal Driver

URL: `jdbc:db2://hostname:port/schemaname`

Driver Class: `com.ibm.db2.jcc.DB2Driver`

Driver Jar: `db2jcc.jar`, `db2jcc4.jar` and `db2java.zip`

9.6.3.5 Using a MySQL Database

Use the following information:

URL: `jdbc:mysql://hostname:3306/dbname`

Driver Class: `com.mysql.jdbc.Driver`

Driver Jar: `mysql-connector-java-3.1.10-bin.jar`

9.6.4 Location of JDBC Driver JAR Files and Setting the Class Path

This section describes the location of JDBC JAR files and setting the class path at run time and design time.

Run Time

For both Windows and Linux, you must perform the following steps:

1. Drop the vendor-specific driver JAR files to the `user_projects/domains/soainfra/lib` directory.
2. Drop the vendor-specific driver JAR files to the `<Weblogic_Home>/server/lib`.
3. Edit the classpath to include the vendor-specific jar file in `<Weblogic_HOME>/common/bin/commEnv.sh`

Design Time

For both Windows and Linux, drop the JDBC JAR to the `Oracle/Middleware/jdeveloper/jdev/lib/patches` directory.

9.7 Stored Procedure and Function Support

This section describes how the Oracle Database Adapter supports the use of stored procedures and functions.

This section includes the following topics:

- Section 9.7.1, "Design Time: Using the Adapter Configuration Wizard"
- Section 9.7.2, "Supported Third-Party Databases"
- Section 9.7.3, "Design Time: Artifact Generation"
- Section 9.7.4, "Run Time: Before Stored Procedure Invocation"
- Section 9.7.5, "Run Time: After Stored Procedure Invocation"
- Section 9.7.6, "Run Time: Common Third-Party Database Functionality"
- Section 9.7.7, "Advanced Topics"

9.7.1 Design Time: Using the Adapter Configuration Wizard

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

This section includes the following topics:

- Section 9.7.1.1, "Using Top-Level Standalone APIs"
- Section 9.7.1.2, "Using Packaged APIs and Overloading"

9.7.1.1 Using Top-Level Standalone APIs

This section describes how to use the Adapter Configuration Wizard with APIs that are not defined in PL/SQL packages. You use the Adapter Configuration Wizard – Stored Procedures to select a procedure or function and generate the XSD file. See Section 9.8, "Oracle Database Adapter Use Cases" if you are not familiar with how to start the Adapter Configuration Wizard.

The following are the steps to select a stored procedure or function by using the Adapter Configuration Wizard:

1. Drag and drop **Database Adapter** from the Service Adapters list to the Exposed Services swim lane in the composite.xml page.

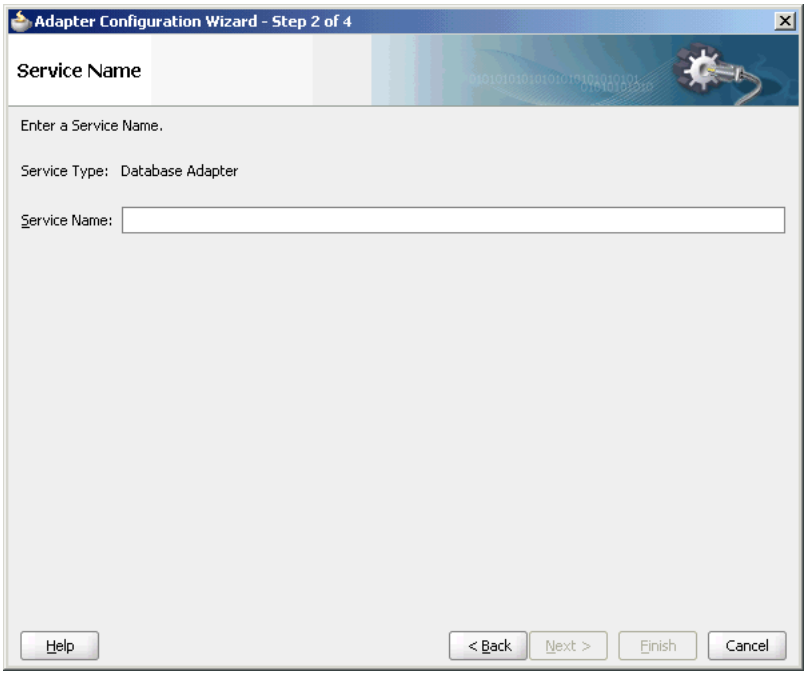
   The Adapter Configuration Wizard is displayed, as shown in Figure 9–31.

Figure 9–31 The Adapter Configuration Wizard

2. Click **Next**. The Service Name page is displayed, as shown in Figure 9–32.

Note: Note that the name of stored procedures or packages that refers to database or user-defined data types must not include the character $ in it. The presence of $ in the name would cause the XSD file generation to fail.

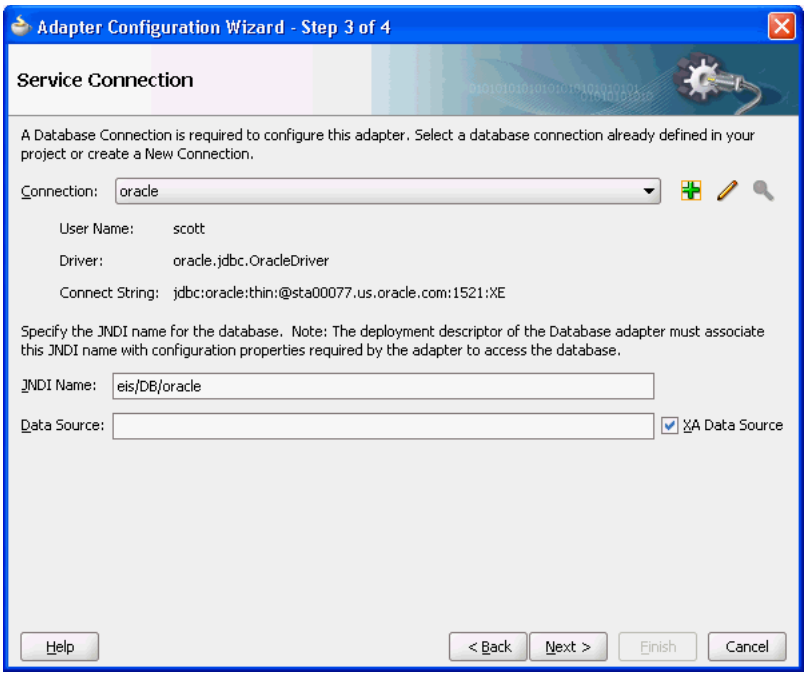
Figure 9–32 Specifying the Service Name



3. In the **Service Name** field, enter a service name, and then click **Next**. The Service Connection page is displayed.

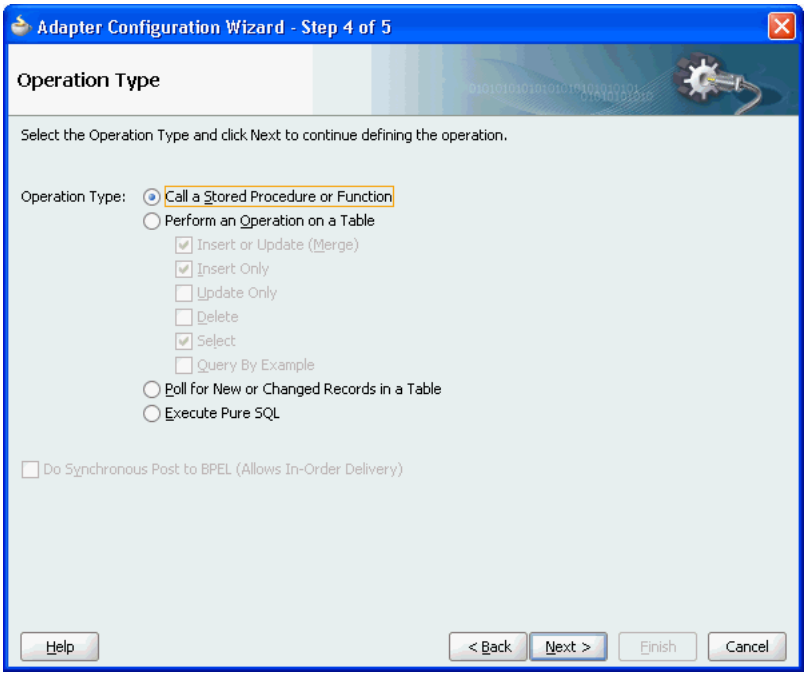
   You associate a connection with the service, as shown in Figure 9–33. A database connection is required to configure the adapter service. Select an existing connection from the list or create a new connection.

Figure 9–33 Setting the Database Connection in the Adapter Configuration Wizard



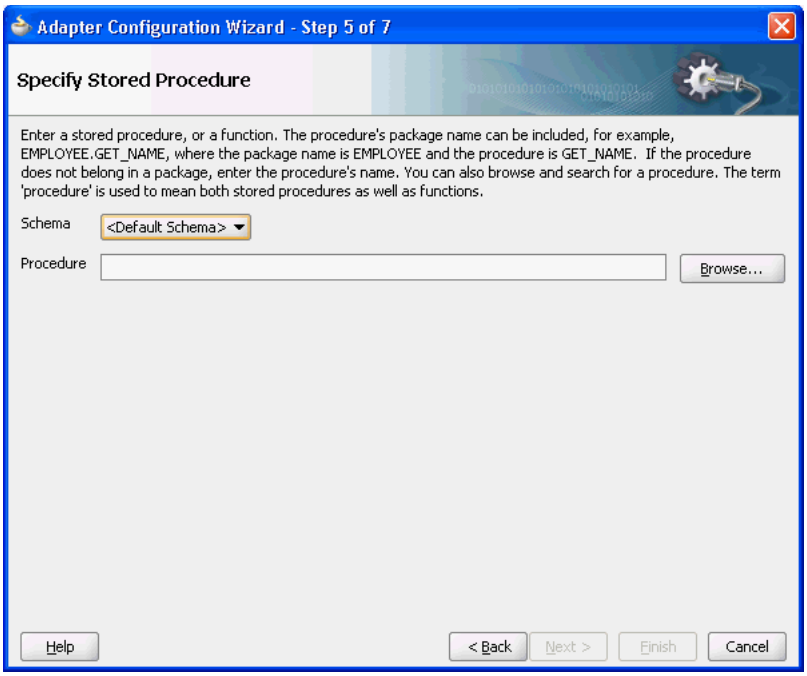
4. Click **Next**. The Operation Type page is displayed.
5. For the **Operation Type**, select **Call a Stored Procedure or Function**, as shown in Figure 9–34.

Figure 9–34 Calling a Stored Procedure or Function in the Adapter Configuration Wizard



6. Click **Next**. The Specify Stored Procedure page is displayed, as shown in Figure 9–35. This is where you specify a stored procedure or function.

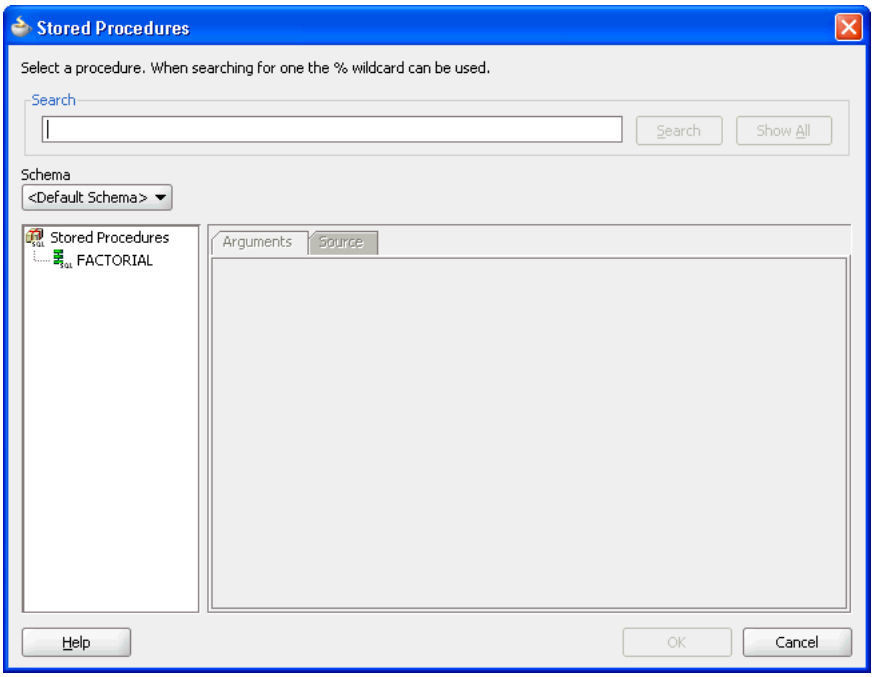
Figure 9–35 The Specify Stored Procedure Page



7. Next, you select the schema and procedure or function. You can select a schema from the list or select **<Default Schema>**, in which case the schema associated with the connection is used. If you know the procedure name, enter it in the Procedure field. If the procedure is defined inside a package, 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, as shown in Figure 9–36.

Figure 9–36 Searching for a Procedure or Function



Select a schema from the list or select **<Default Schema>**. A list of the available procedures is displayed in the left window. To search for a particular API in a long list of APIs, enter search criteria in the **Search** field. For example, to find all APIs that begin with `XX`, enter `XX%` and click the **Search** button. Clicking the **Show All** button displays all available APIs.

Figure 9–37 shows how you can select the FACTORIAL function. The Arguments tab displays the parameters of the function, including their names, type, mode (`IN`, `IN/OUT` or `OUT`) and the numeric position of the parameter in the definition of the procedure. The return value of a function has no name and is always an `OUT` parameter at position zero (0).

Figure 9–37 Viewing the Arguments of a Selected Procedure

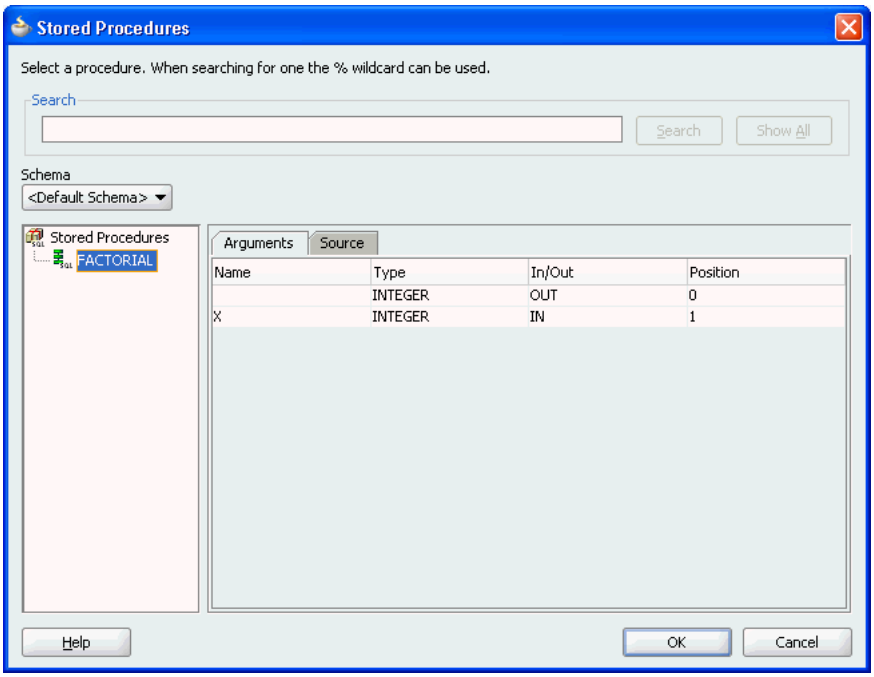
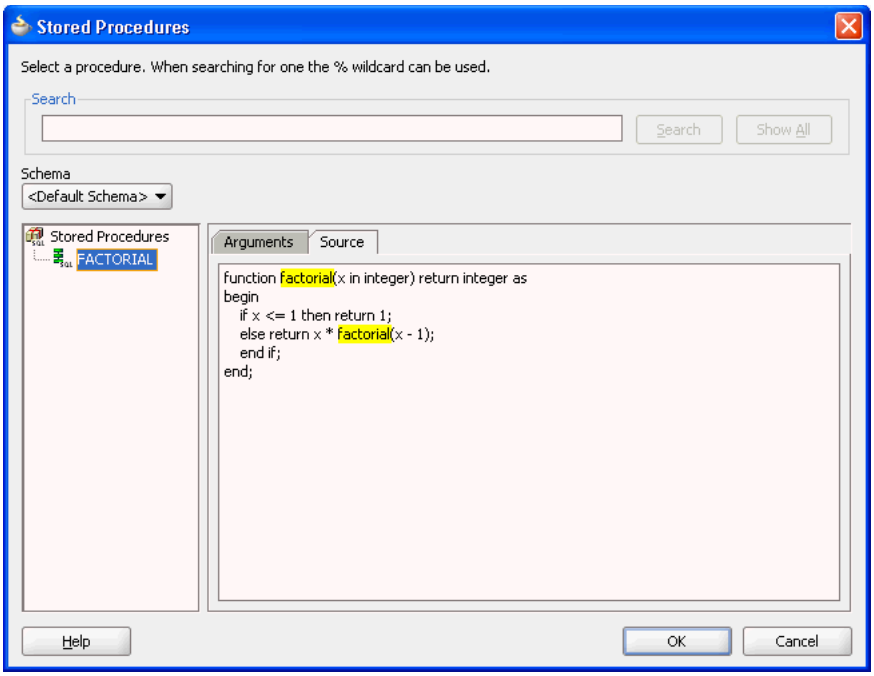


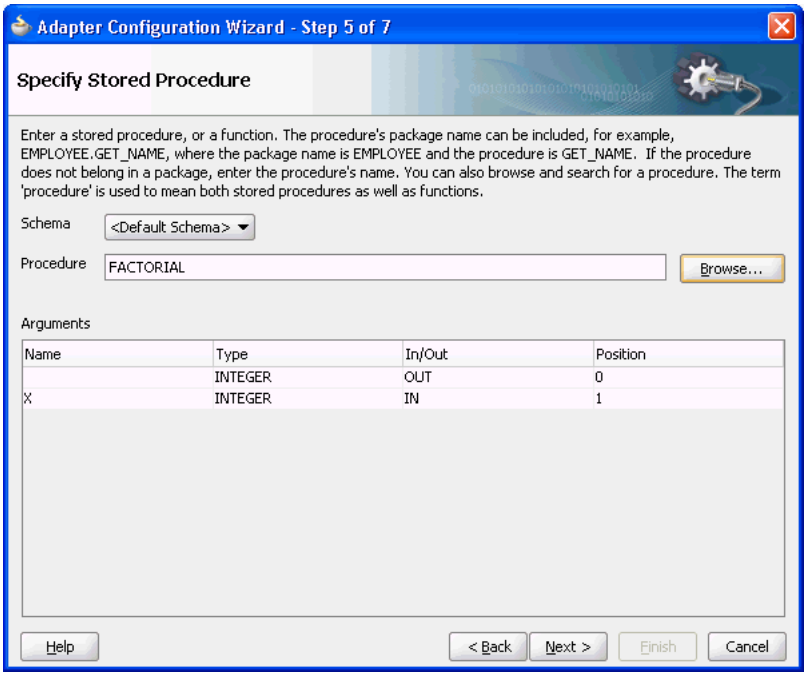
Figure 9–38 shows how the Source tab displays the code that implements the function. Text that matches the name of the function is highlighted.

Figure 9–38 Viewing the Source Code of a Selected Procedure



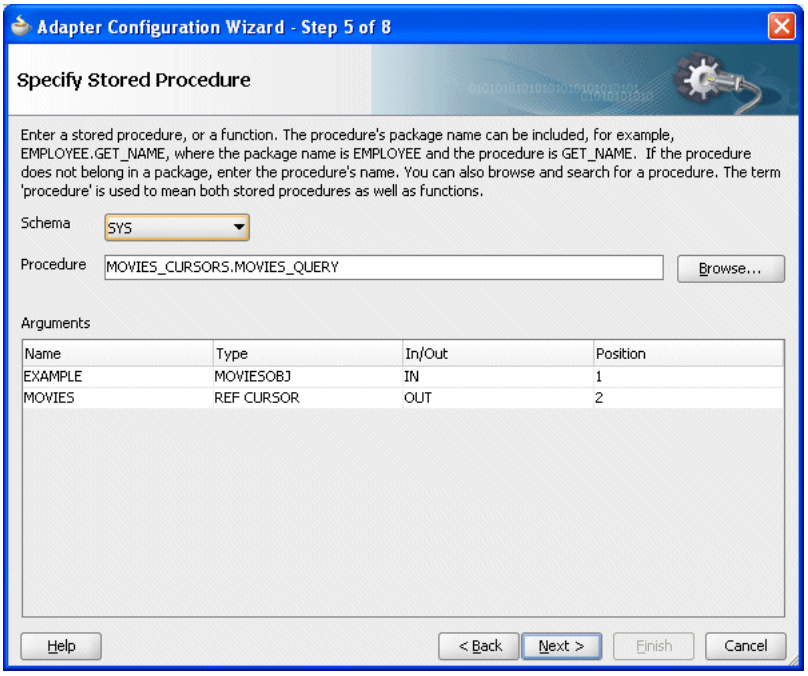
8. Click **OK** after selecting a procedure or function. Information about the API is displayed, as shown in Figure 9–39. Click **Back** or **Browse** to make revisions.

Figure 9–39 Viewing Procedure or Function Details in the Adapter Configuration Wizard



9. Click **Next**. If the stored procedure or function has an output parameter of type row set (`REF CURSOR` on Oracle Database), as Figure 9–40 shows, you can define a strongly or weakly typed XSD for this ref cursor.

Figure 9–40 Viewing Procedure or Function Details in the Adapter Configuration Wizard: Row Set Type

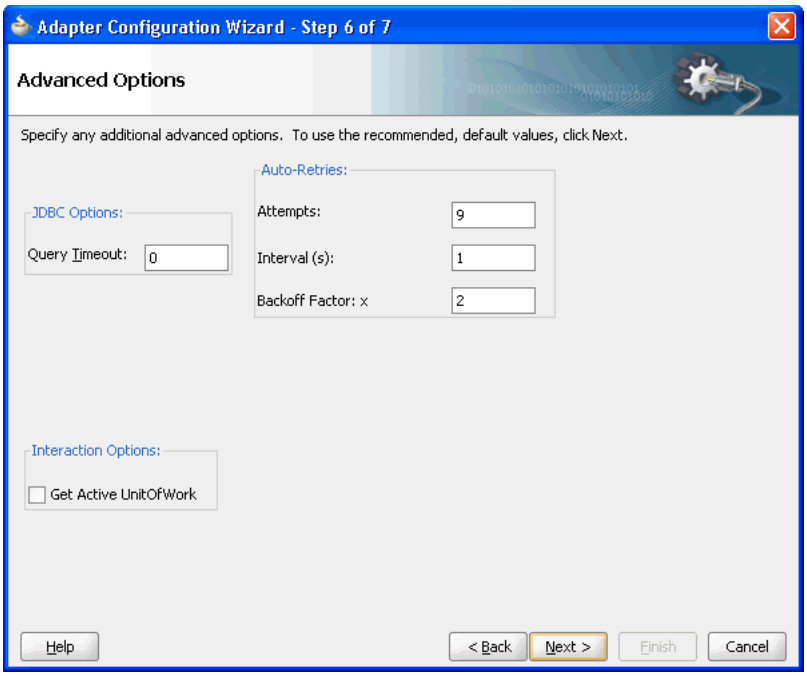


For more information, see:

- Section 9.7.7.1, "Row Set Support Using a Strongly Typed XSD"
- Section 9.7.7.2, "Row Set Support Using a Weakly Typed XSD"

10. Click **Next**. The Advanced Options page is displayed, as shown in Figure 9–41. Enter any advanced options, such as the JDBC `QueryTimeout` value. Other options include retry parameters, such as the number of retry attempts and the interval between them.

Figure 9–41 The Advanced Options Page



11. After specifying all options, click **Next**, and then click **Finish** to complete the Adapter Configuration Wizard.

    When you have finished using the Adapter Configuration Wizard, three files are added to the existing project:

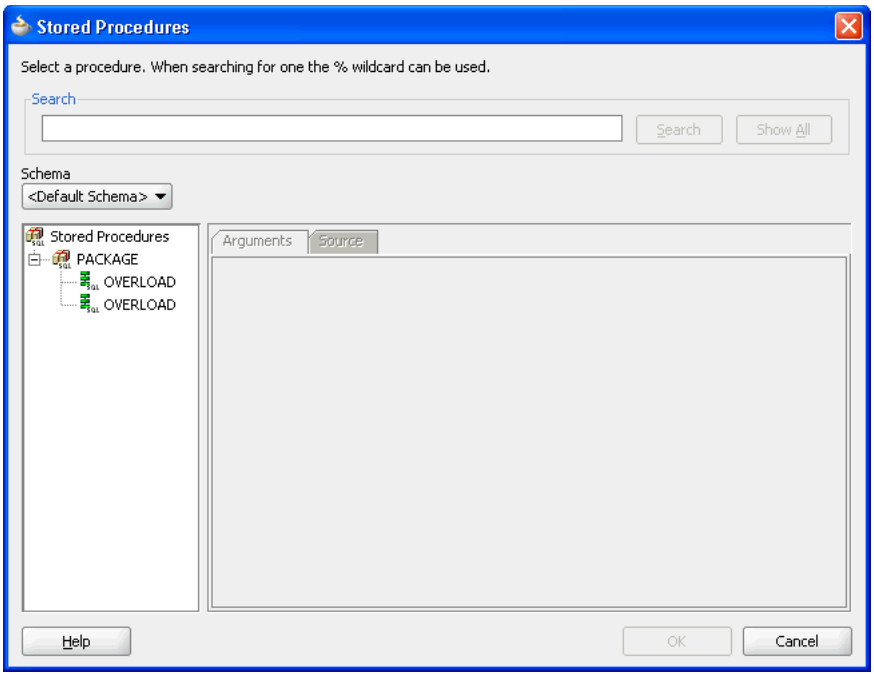
    - *servicename*`.wsdl` (for example, `Factorial.wsdl`)
    - *service_name_db*`.jca` (for example, `Factorial_db.jca`)
    - *schema_package_procedurename*`.xsd` (for example, `SCOTT_FACTORIAL.xsd`)

9.7.1.2 Using Packaged APIs and Overloading

Using APIs defined in packages is similar to using standalone APIs. The only difference is that you can expand the package name to see a list of all the APIs defined within the package, as shown in Figure 9–42.

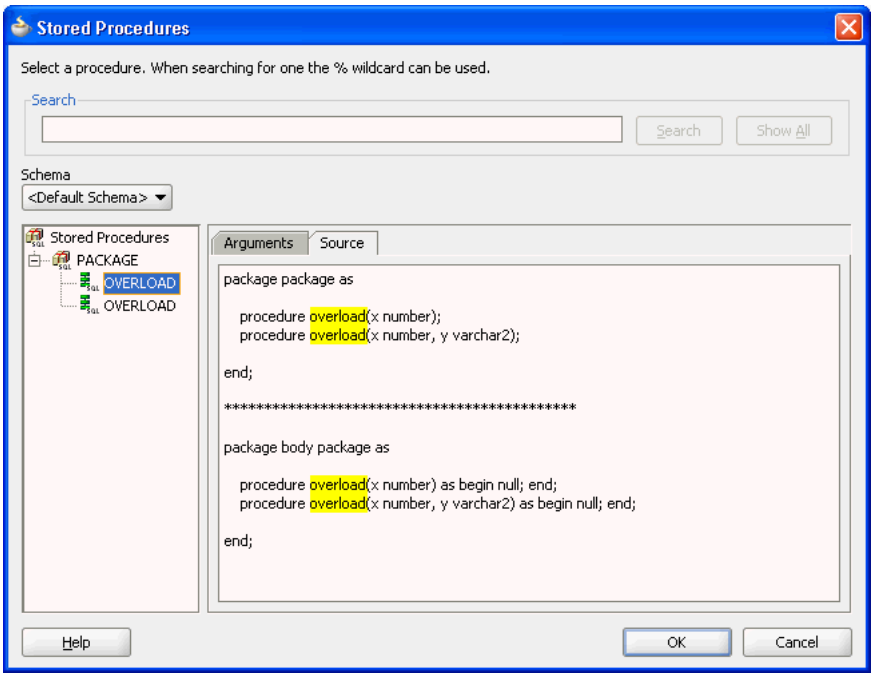
APIs that have the same name but different parameters are called overloaded APIs. As shown in Figure 9–42, the package called **PACKAGE** has two overloaded procedures called **OVERLOAD**.

Figure 9–42 A Package with Two Overloaded Procedures



As Figure 9–43 shows, the code for the entire PL/SQL package is displayed, regardless of which API from the package is selected when you view the Source tab. Text that matches the name of the procedure is highlighted.

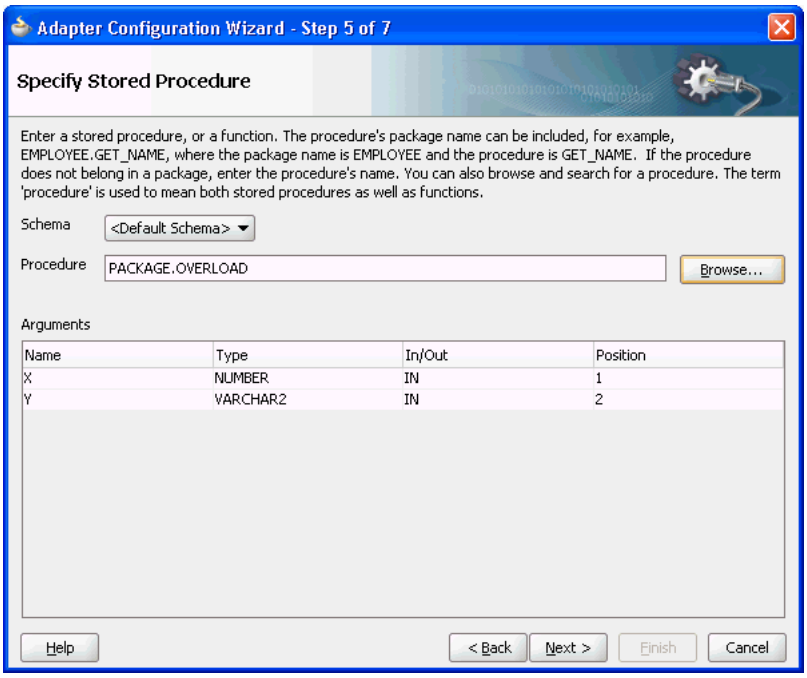
Figure 9–43 Viewing the Source Code of an Overloaded Procedure



After you select a procedure or function and click **OK**, information about the API is displayed, as shown in Figure 9–44. The schema, procedure name, and a list of arguments are displayed. Note how the procedure name is qualified with the name of the package (**PACKAGE.OVERLOAD**). Click **Back** or **Browse** to make revisions, or

Next. Enter values for any of the advanced options. Click **Next** followed by **Finish** to conclude.

Figure 9–44 Viewing Procedure or Function Details in the Adapter Configuration Wizard



When you have finished using the Adapter Configuration Wizard, the following files are added to the existing project:

- `Overload.wsdl`, `Overload_db.jca`
- `SCOTT_PACKAGE_OVERLOAD_2.xsd`.

  The `_2` appended after the name of the procedure in the XSD filename differentiates the overloaded APIs. Numeric indexes are used to differentiate between overloaded APIs.

9.7.2 Supported Third-Party Databases

For stored procedures the following databases are supported: Oracle, DB2, Informix Dynamic Server, MySQL, Microsoft SQL Server, and Sybase Adaptive Server Enterprise. Contact support for specific versions that have been certified. If your particular version is more recent than one mentioned here it is probably supported.

For more information on Oracle JCA Adapters support for third-party JDBC drivers and databases, see Section 9.6, "JDBC Driver and Database Connection Configuration".

This section includes the following topics:

- Section 9.7.2.1, "Terms Used."
- Section 9.7.2.2, "Supported Third-Party Databases"
- Section 9.7.2.3, "Creating Database Connections"

9.7.2.1 Terms Used

ProductName
This is the name of the database.

Database Name	Supported Database
IBM DB2	IBM DB2 v 9.x
Microsoft SQL Server	SQLServer 2000 or 2005
MySQL	MySQL v5.6

DriverClassName
This is the name of the JDBC Driver Class.

Database Name	JDBC Driver
IBM DB2	`com.ibm.db2.jcc.DB2Driver`
Microsoft SQL Server	`com.microsoft.sqlserver.jdbc.SQLServerDriver`
MySQL	`com.mysql.jdbc.Driver`

ConnectionString
This is the JDBC Connection URL.

Database Name	Connection String
IBM DB2	`jdbc:db2://`*`hostname`*`:`*`port`*`/`*`database-name`*
Microsoft SQL Server	`jdbc:sqlserver://`*`hostname`*`:`*`port`*`;databaseName=`*`name`*
MySQL	`jdbc:mysql://`*`host`*`:`*`port`*`/`*`database-name`*

Username
This is the database user name.

Password
This is the password associated with the user name.

ProcedureName
This is the name of the stored procedure or the function.

ServiceName
This is the service name for the desired operation.

DatabaseConnection
This is the JNDI name of the connection. For example, `eis/DB/<DatabaseConnection>`.

Destination
This is the destination directory for the generated files. For example, `C:\Temp`.

Parameters

The parameters of the stored procedure (for versions of MySQL before 5.2.6 only.)

QueryTimeout

The JDBC query timeout value (in seconds.) The `QueryTimeout` property specifies the maximum number of seconds that the JDBC driver should wait for the specified stored procedure or function to execute. When the threshold is exceeded, `SQLException` is thrown. If the value is zero, then the driver waits indefinitely.

9.7.2.2 Supported Third-Party Databases

The Adapter Configuration Wizard supports Oracle Database, IBM DB2, AS/400, Microsoft SQL Server, and MySQL v5.2.6 or higher.

This section includes the following topics:

- Section 9.7.2.2.1, "Microsoft SQL Server"
- Section 9.7.2.2.2, "DB2 Data Types"
- Section 9.7.2.2.3, "IBM DB2 AS/400"
- Section 9.7.2.2.4, "MySQL"

9.7.2.2.1 Microsoft SQL Server

Table 9–13 lists the supported data types for SQL Server stored procedures and functions:

Table 9–13 Data Types for SQL Server Stored Procedures and Functions

SQL Data Type	XML Schema Type
BIGINT	long
BINARY	base64Binary
IMAGE	
TIMESTAMP	
VARBINARY	
BIT	boolean
CHAR	string
SQL_VARIANT	
SYSNAME	
TEXT	
UNIQUEIDENTIFIER	
VARCHAR	
XML (2005 only)	
DATETIME	dateTime
SMALLDATETIME	
DECIMAL	decimal
MONEY	
NUMERIC	
SMALLMONEY	

Table 9–13 (Cont.) Data Types for SQL Server Stored Procedures and Functions

SQL Data Type	XML Schema Type
FLOAT	float
REAL	
INT	int
SMALLINT	short
TINYINT	unsignedByte

Besides, the data types mentioned in the preceding table, alias data types are also supported. Alias data types are created by using the `sp_addtype` database engine stored procedure or the `CREATE TYPE` Transact-SQL statement (only for SQL Server 2005.) The use of the Transact-SQL statement is the preferred method for creating alias data types. The use of `sp_addtype` is being deprecated.

9.7.2.2.2 DB2 Data Types

Table 9–14 lists the supported data types for DB2 SQL stored procedures:

Table 9–14 Data Types for DB2 SQL Stored Procedures

SQL Data Type	XML Schema Type
BIGINT	long
BLOB	base64Binary
CHAR FOR BIT DATA	
VARCHAR FOR BIT DATA	
CHARACTER	string
CLOB	
VARCHAR	
DATE	dateTime
TIME	
TIMESTAMP	
DECIMAL	decimal
DOUBLE	double
INTEGER	int
REAL	float
SMALLINT	short

The names of other data types are also supported implicitly. For example, `NUMERIC` is equivalent to `DECIMAL` (as is `DEC` and `NUM` as well.)

IBM DB2 supports structured data types (user-defined). However, there is no support for these types in the JDBC drivers. Consequently, a structured data type may not be used as the data type of a parameter in a stored procedure. IBM DB2 also supports user-defined functions. The adapter, however, does not support these functions.

In the Adapter Configuration Wizard, stored procedures are grouped by database user. A *schema* in IBM DB2 is equivalent to a schema in Oracle. Both represent the name of a database user.

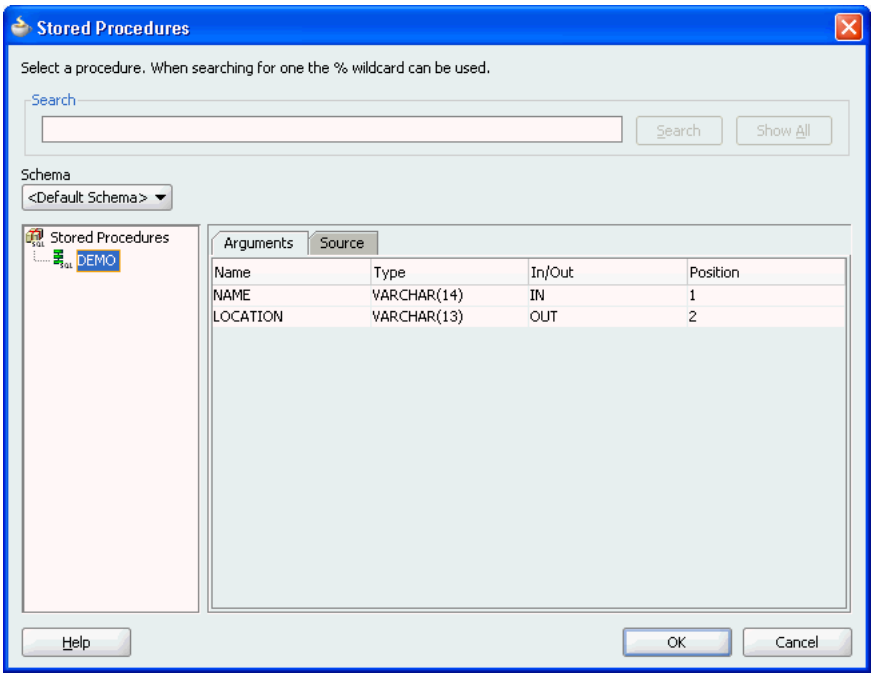
For IBM DB2, **<Default Schema>** refers to the current database user.

Click **<Default Schema>** to select a different database user. The stored procedures in the **Browse** page are those that the database user created in the database specified as <database> in the JDBC Connection URL.

The Adapter Configuration Wizard does not support changing to a different database.

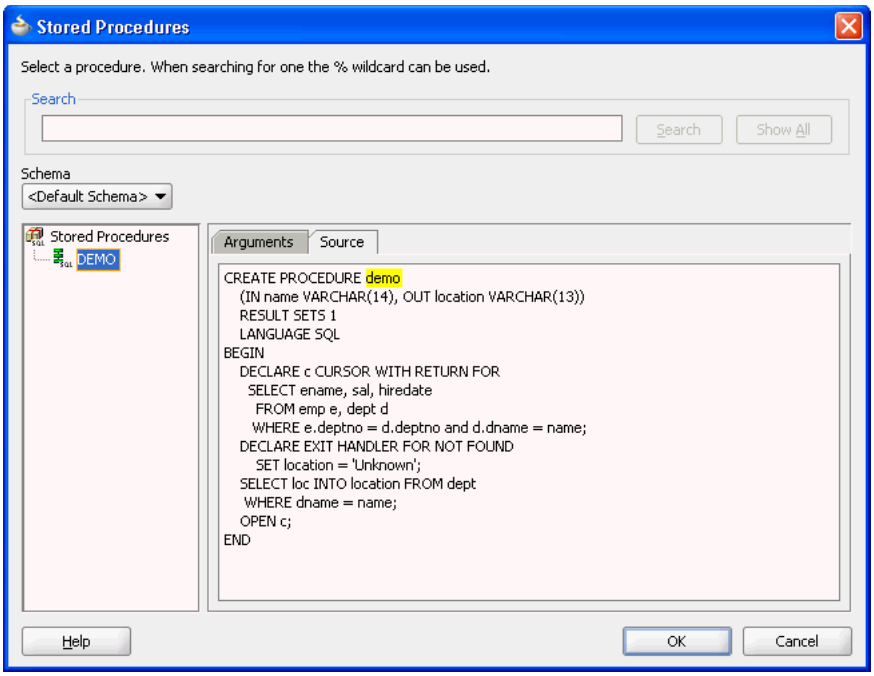
Select the stored procedure in the Stored Procedures dialog, as shown in Figure 9–45. The arguments are shown in the Arguments tab. Click **Search** to find database stored procedures that the user created in the specified database. For example, 'd%' or 'D%' would both find the `DEMO` stored procedure. Clicking **Show All** reveals all of the procedures that the current user created in the specified database.

Figure 9–45 The Stored Procedures Dialog



You can view the source code of the stored procedure by clicking the **Source** tab, as shown in Figure 9–46.

Figure 9–46 The Source Code of the Stored Procedure



9.7.2.2.3 IBM DB2 AS/400

Table 9–15 lists the supported data types for IBM DB2 AS/400 stored procedures:

Table 9–15 Data Types for IBM DB2 AS/400 Stored Procedures

SQL Data Type	XML Schema Type
BINARY BINARY LARGE OBJECT BINARY VARYING	base64Binary
CHARACTER CHARACTER LARGE OBJECT CHARACTER VARYING	string
DATE TIME TIMESTAMP	dateTime
DECIMAL NUMERIC	decimal
DOUBLE PRECISION	double
BIGINT	long
INTEGER	int
REAL	float
SMALLINT	short

Distinct types are also supported for data types that are created using the `CREATE DISTINCT TYPE` statement. These data types work in the same way as they do in IBM DB2.

The IBM DB2 AS/400 implementation is based on queries from catalog tables in the QSYS2 schema. The adapter tries to determine whether the QSYS2.SCHEMATA table exists. If it does, then the Adapter Configuration Wizard queries tables in the QSYS2 schema. Therefore, if your IBM DB2 AS/400 database supports the QSYS2 schema, then the Adapter Configuration Wizard and the adapter run time should both work.

The Adapter Configuration Wizard checks the SYSCAT schema first, and then the QSYS2 schema. The adapter does not support the catalog tables in the SYSIBM schema.

9.7.2.2.4 MySQL

Use the Adapter Configuration Wizard to access stored procedures on MySQL v5.6 or later using catalog tables in the INFORMATION_SCHEMA schema. Versions of MySQL before v5.6 lack a PARAMETERS table in the INFORMATION_SCHEMA schema.

Without a PARAMETERS table, the MySQL database does not provide any information about the parameters of a stored procedure. It is therefore necessary to supply this information using a required property in the properties file. The Parameters property contains the signature of the stored procedure.

Property	Description
IsFunction	Determines whether the API is a function or a procedure
SchemaName	The name of the database where the API is defined
Parameters	The parameters of the stored procedure

The value of the Parameters property is a comma-delimited list of parameters, each of which has the following syntax

```
Parameter ::= {IN | INOUT | OUT} Parameter_Name SQL_Datatype
```

All three elements of a parameter definition are required.

Consider the following MySQL stored procedure:

```
CREATE PROCEDURE demo
(IN x VARCHAR (10), INOUT y INT, OUT z CHAR (20))
BEGIN
...
END
```

The Parameters property must be specified as shown in the following example:

```
Parameters=IN x VARCHAR (10), INOUT y INT,  OUT z CHAR (20)
```

The generated XSD for the stored procedure is invalid unless the parameters are specified correctly in the parameters property. The following is a sample of a properties file for MySQL:

```
ProductName=MySQL
DriverClassName=com.mysql.jdbc.Driver
ConnectionString=jdbc:mysql://<host>:<port>/<database>
Username=<username>
Password=<password>
SchemaName=<database>
ProcedureName=demo
Parameters=IN x VARCHAR(10),INOUT y INT,OUT z TEXT(20)
ServiceName=MySQLDemoService
DatabaseConnection=mysql
```

Note: For MySQL, the `SchemaName`, `Parameters`, and `IsFunction` properties are all required properties.

Table 9–16 lists the supported data types for MySQL stored procedures:

Table 9–16 Data Types for MySQL Stored Procedures

SQL Data Type	XML Schema Type
BINARY	base64Binary
BLOB	
LONGBLOB	
MEDIUMBLOB	
TINYBLOB	
VARBINARY	
BOOLEAN	boolean
CHAR	string
LONGTEXT	
MEDIUMTEXT	
TEXT	
TINYTEXT	
VARCHAR	
DATE	dateTime
DATETIME	
TIMESTAMP	
DECIMAL	decimal
NUMERIC	
REAL	
DOUBLE	double
FLOAT	float
TINYINT	byte
TINYINT UNSIGNED	unsigned_byte
SMALLINT	short
SMALLINT UNSIGNED	unsigned_short
INTEGER	int
INT	
MEDIUMINT	
INTEGER UNSIGNED	unsigned_int
INT UNSIGNED	
MEDIUMINT UNSIGNED	
BIGINT	long
BIGINT UNSIGNED	unsigned_long

The character length for any SQL data type that corresponds with `STRING` can be specified using the '(#)' notation in the `Parameters` property, for example, `VARCHAR(20)`. Specifying the length of any other SQL data type does not have any effect.

`UNSIGNED` integer data types are treated as though they were `SIGNED` integer data types when using the Adapter Configuration Wizard.

Stored procedures in MySQL are grouped by database specified by <database> in the JDBC Connection URL. For MySQL, <**Default Schema**> refers to the database that the user is connected to (usually specified in the JDBC connection URL.) Click **<Default Schema>** to select a different database. Click **Search** to search for specific stored procedures in the current database specified in the JDBC Connection URL. For example, 'd%' or 'D%' would both find stored procedures beginning with 'd' or 'D.' Click **Show All** to reveal all procedures in the current database.

9.7.2.3 Creating Database Connections

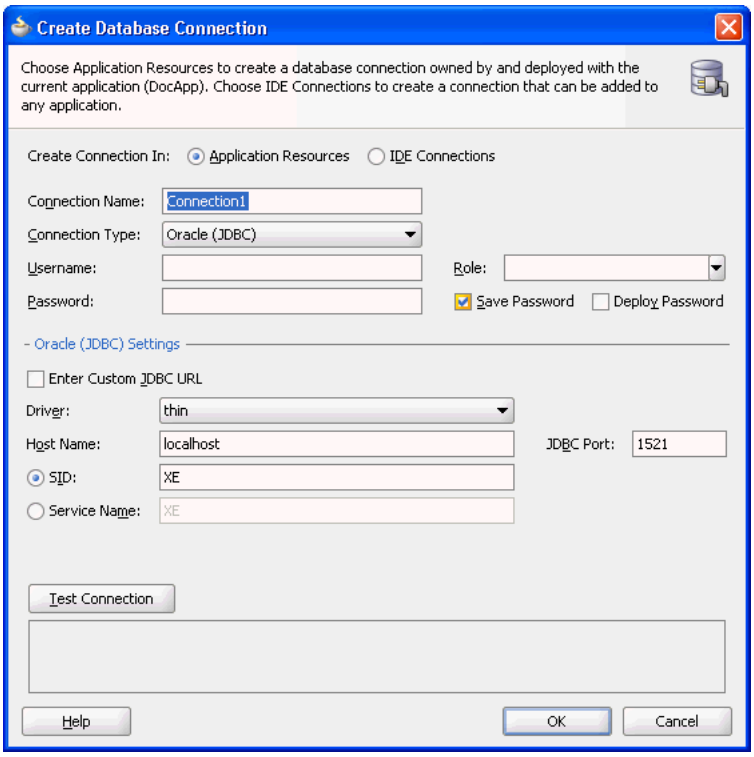
Database connections must be created in JDeveloper to access catalog tables necessary for the Adapter Configuration Wizard to work.

The following are the steps to create a database connection by using JDeveloper:

1. Select **Database Navigator** from View.
2. Right-click the application name, then click **New** followed by **Connections**. Select **Database Connection**.

   The Create Database Connection page is displayed, as shown in Figure 9–47.

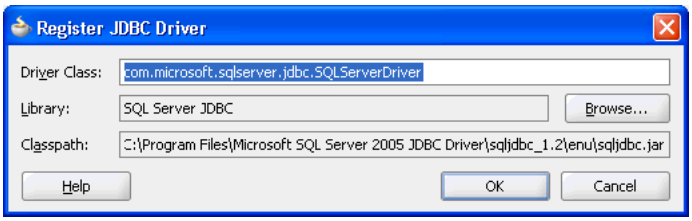
Figure 9–47 The Create Database Connection



3. Enter a connection name in the **Connection Name** field. For example, `sqlserver`.
4. Select **Generic JDBC** as the Connection Type from the Connection Type list.

5. Enter your Username, Password, and role information.
6. Click **New** for Driver Class. The Register JDBC Driver dialog is displayed, as shown in Figure 9–48.

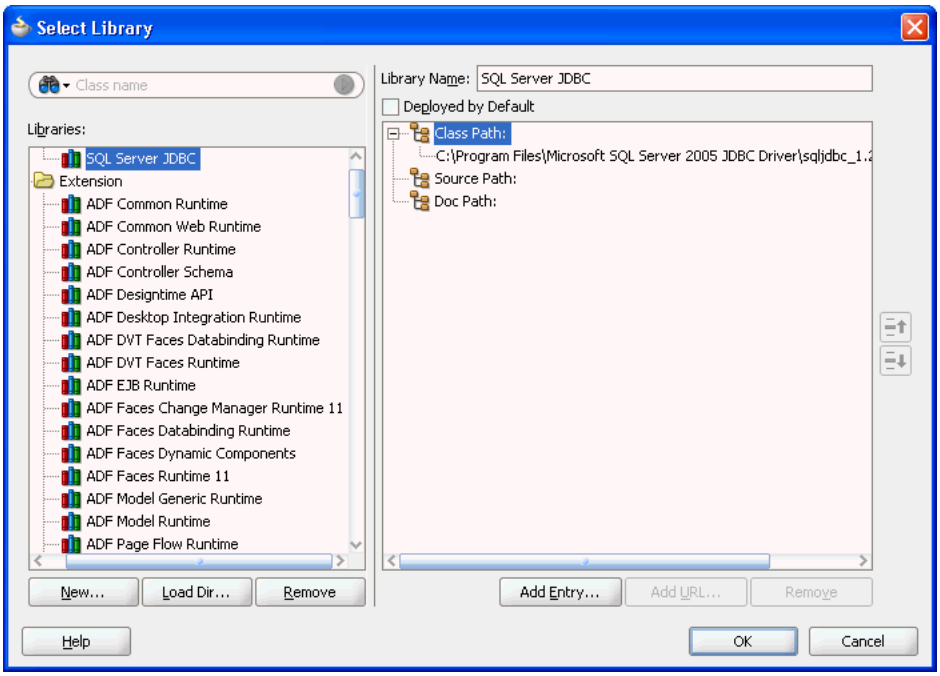
Figure 9–48 The Register JDBC Driver Dialog



7. Enter the Driver Class (for example, `com.microsoft.sqlserver.jdbc.SQLServerDriver`).
8. Create a library or edit an existing one by using the following steps:
   a. Click **Browse** in the Register JDBC Driver dialog.
   b. Click **New** in the Select Library dialog.

      The Select Library dialog is displayed, as shown in Figure 9–49.

Figure 9–49 The Select Library Dialog

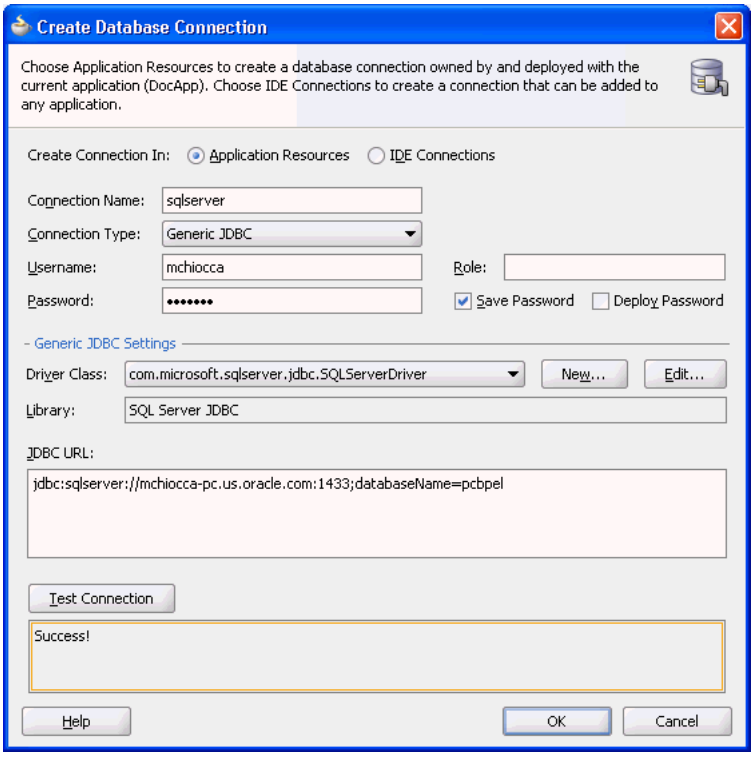


   c. Select an existing library or click **New** to create one.

      The Create Library dialog is displayed.
   d. Enter a library name, for example, `SQL Server JDBC`.
   e. Click **Add Entry** to add JDBC jar files to the class path.
   f. Click **OK** twice to exit the Create Library windows.
   g. Click **OK** to exit the Register JDBC Driver window.

9. Enter your connection string name for **JDBC URL**.
10. Click **Test Connection**.
11. If the connection is successful, then a screen, as shown in Figure 9–50 is displayed.

Figure 9–50 The Create Database Connection Dialog



12. Click **OK** followed by **Finish**.

9.7.3 Design Time: Artifact Generation

The Adapter Configuration Wizard – Stored Procedures is capable of creating a WSDL file and a valid XSD file that describes the signature of a stored procedure or function. The following sections describe the relevant structure and content of both the WSDL and the XSD files, and their relationship with each other.

This section includes the following topics:

- Section 9.7.3.1, "The WSDL–XSD Relationship"
- Section 9.7.3.2, "JCA File"
- Section 9.7.3.3, "Oracle Data Types"
- Section 9.7.3.4, "Generated XSD Attributes"
- Section 9.7.3.5, "User-Defined Types"
- Section 9.7.3.6, "Complex User-Defined Types"
- Section 9.7.3.7, "Object Type Inheritance"
- Section 9.7.3.8, "Object References"
- Section 9.7.3.9, "Referencing Types in Other Schemas"

- Section 9.7.3.10, "XSD Pruning Optimization"

9.7.3.1 The WSDL–XSD Relationship

In the paragraphs that follow, the operation name, `Factorial`, and procedure name, `Factorial`, are taken from an example cited previously (see Figure 9–39). The generated WSDL imports the XSD file.

```
<types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema">
    <import
namespace="http://xmlns.oracle.com/pcbpel/adapter/db/SCOTT/FACTORIAL/"
      schemaLocation="xsd/SCOTT_FACTORIAL.xsd"/>
  </schema>
</types>
```

The namespace is derived from the schema, package, and procedure name, and appears as the `targetNamespace` in the generated XSD.

A root element called `InputParameters` is created in the XSD file for specifying elements that correspond to the `IN` and `IN/OUT` parameters of the stored procedure. Another root element called `OutputParameters` is also created in the XSD file for specifying elements only if there are any `IN/OUT` or `OUT` parameters. `IN/OUT` parameters appear in both root elements.

These root elements are represented in the XSD file as an unnamed `complexType` definition whose sequence includes one element for each parameter. If there are no `IN` or `IN/OUT` parameters, then the `InputParameters` root element is still created; however, `complexType` is empty. A comment in the XSD file indicates that there are no such parameters. An example of a root elements follows.

```
<element name="InputParameters"
  <complexType>
    <sequence>
      <element …>
       …
    </sequence>
  </complexType>
</element>
```

The WSDL defines message types whose parts are defined in terms of these two root elements.

```
<message name="args_in_msg"
  <part name="InputParameters" element="InputParameters"/>
</message>
<message name="args_out_msg"
  <part name="OutputParameters" element="OutputParameters"/>
</message>
```

The `db` namespace is equal to the `targetNamespace` of the generated XSD. The `args_in_msg` message type always appears in the WSDL while `args_out_msg` is included only if the `OutputParameters` root element is generated in the XSD file.

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

```
<portType name="Factorial_ptt">
  <operation name="Factorial">
    <input message="tns:args_in_msg"/>
```

```
    <output message="tns:args_out_msg"/>
  </operation>
</portType>
```

The input message always appears while the output message depends on the existence of the `OutputParameters` root element in the XSD file. The `tns` namespace is derived from the operation name and is defined in the WSDL as

```
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/db/Factorial/"
```

The root elements in the XSD file define the structure of the parts used in the messages that are passed into and sent out of the Web service encapsulated by the WSDL.

The input message in the WSDL corresponds to the `InputParameters` root element from the XSD file. The instance XML supplies values for the `IN` and `IN/OUT` parameters of the stored procedure. The output message corresponds to the `OutputParameters` root element. This is the XML file that gets generated after the stored procedure has executed. It holds the values of any `IN/OUT` and `OUT` parameters.

9.7.3.2 JCA File

The JCA file provides adapter configuration information for the service. A connection factory is specified so that the adapter run time can connect to the database, as shown in the following example. Non-managed connection properties should not be specified directly in the JCA file. Instead you should create a connection factory on the application server, and refer to it by name in the JCA file (<connection-factory location).

```
<connection-factory location="eis/DB/oracle" UIConnectionName="oracle"
adapterRef="">
</connection-factory>
```

The JNDI name, *eis/DB/oracle*, was earlier specified as the service connection in the Adapter Configuration Wizard.

End point properties for the interaction are also specified. The name of the schema, package, and procedure are specified, as shown in the following example. The operation name ties the JCA file back to the service WSDL.

```
<connection-factory location="eis/db/oracle" UIConnectionName="oracle"
adapterRef=""/>
<endpoint-interaction portType="Factorial_ptt" operation="Factorial">
  <interaction-spec
    className="oracle.tip.adapter.db.DBStoredProcedureInteractionSpec">
      <property name="ProcedureName" value="FACTORIAL"/>
      <property name="GetActiveUnitOfWork="false"/>
  </interaction-spec>
  </output>
</endpoint-interaction>
```

Note the operation name and procedure name. If an explicit schema had been chosen or if the procedure had been defined in a package, then values for these properties would also be listed here.

Note: Non-managed connection details are not created in the `DBAdapter.jca` files when you start JDeveloper in the normal mode. However, non-managed connection details are created in the `DBAdapter .jca` files when you start JDeveloper in the preview mode.

9.7.3.3 Oracle Data Types

Many primitive data types 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 9–17 lists the supported data types for Oracle stored procedures and functions.

Table 9–17 Data Types for Oracle Stored Procedures and Functions

SQL or PL/SQL Type	XML Schema Type
`BINARY_DOUBLE` `DOUBLE PRECISION`	`double`

`BINARY_FLOAT`
`FLOAT`
`REAL`
	`float`

`BINARY_INTEGER`
`INTEGER`
`PLS_INTEGER`
`SMALLINT`
	`int`

`BLOB`
`LONG RAW`
`RAW`
	`base64Binary`

`CHAR`
`CLOB`
`LONG`
`STRING`
`VARCHAR2`
	`string`

`DATE`
`TIMESTAMP`
`TIMESTAMP WITH TIME ZONE`
	`dateTime`

`DECIMAL` `NUMBER`	`decimal`

9.7.3.4 Generated XSD Attributes

Table 9–18 lists the attributes used in the generated XSDs.

Table 9–18 Generated XSD Attributes

Attribute	Example	Purpose
`name`	`name="param"`	Name of an element
`type`	`type="string"`	XML schema type

Table 9–18 (Cont.) Generated XSD Attributes

Attribute	Example	Purpose
db:type	db:type="VARCHAR2"	SQL or PL/SQL type
db:index	db:index="1"	Position of a parameter
db:default	db:default="true"	Has a default clause
minOccurs	minOccurs="0"	Minimum occurrences
maxOccurs	maxOccurs="1"	Maximum occurrences
nillable	nillable="true"	Permits null values

The db namespace is used to distinguish attributes used during run time from standard XML schema attributes. The db:type attribute is used to indicate what the database type is so that a suitable JDBC type mapping can be obtained at run time. The db:index attribute is used as an optimization by both the design-time and run-time components to ensure that the parameters are arranged in the proper order. Parameter indexes begin at 1 for procedures and 0 for functions. The return value of a function is represented as an OutputParameter element whose name is the name of the function and whose db:index is 0. The db:default attribute is used to indicate whether or not a parameter has a default clause.

The minOccurs value is set to 0 to allow for an IN parameter to be removed from the XML file. This is useful when a parameter has a default clause defining a value for the parameter (for example, X IN INTEGER DEFAULT 0). At run time, if no element is specified for the parameter in the XML file, the parameter is omitted from the invocation of the stored procedure, thus allowing the default value to be used. Each parameter can appear at most once in the invocation of a stored procedure or function. Therefore, maxOccurs, whose default value is always 1, is always omitted from elements representing parameters.

The nillable attribute is always set to true to allow the corresponding element in the instance XML to have a null value (for example, <X/> or <X></X>). In some cases, however, to pass an element such as this element, which does have a null value, you must state this explicitly (for example, <X xsi:nil="true"/>). The namespace, xsi, used for the nillable attribute, must be declared explicitly in the instance XML (for example, xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance").

9.7.3.5 User-Defined Types

The Adapter Configuration Wizard can also generate valid definitions for user-defined types such as collections (VARRAY and nested tables) and OBJECT. These are created as complexType definitions in the XSD file.

For VARRAY, the complexType definition defines a single element in its sequence, called *name*_ITEM, where *name* is the name of the VARRAY element. All array elements in the XML file are so named. Given the following VARRAY type definition,

```
SQL> CREATE TYPE FOO AS VARRAY (5) OF VARCHAR2 (10);
```

and a VARRAY element, X, whose type is FOO, the following complexType is generated:

```
<complexType name="FOO">
  <sequence>
    <element name="X_ITEM" db:type="VARCHAR2" minOccurs="0" maxOccurs="5"
nillable="true"/>
      <simpleType>
```

```
        <restriction base="string">
          <maxLength value="10"/>
        </restriction>
      </simpleType>
  </sequence>
</complexType>
```

The `minOccurs` value is `0` to allow for an empty collection. The `maxOccurs` value is set to the maximum number of items that the collection can hold. The `db:index` attribute is not used. Having `nillable` set to `true` allows individual items in the `VARRAY` to be null.

Note the use of the restriction specified on the element of the `VARRAY`, `FOO`. This is used on types such as `CHAR` and `VARCHAR2`, whose length is known from the declaration of the `VARRAY` (or nested table). It specifies the type and maximum length of the element. An element value that exceeds the specified length causes the instance XML to fail during schema validation.

The attribute values of a parameter declared to be of type `FOO` look as follows in the generated XSD:

```
<element name="X" type="db:FOO" db:type="Array" db:index="1" minOccurs="0"
nillable="true"/>
```

The `type` and `db:type` values indicate that the parameter is represented as an array defined by the `complexType` called `FOO` in the XSD file. The value for `db:index` is whatever the position of that parameter is in the stored procedure.

A nested table is treated almost identically to a `VARRAY`. The following nested table type definition,

```
SQL> CREATE TYPE FOO AS TABLE OF VARCHAR2 (10);
```

is also generated as a `complexType` with a single element in its sequence, called *name*`_ITEM`. The element has the same attributes as in the `VARRAY` example, except that the `maxOccurs` value is unbounded because nested tables can be of arbitrary size.

```
<complexType name="FOO">
  <sequence>
    <element name="X_ITEM" … maxOccurs="unbounded" nillable="true">
      …
    </element>
  </sequence>
</complexType>
```

An identical restriction is generated for the `X_ITEM` element in the `VARRAY`. The attributes of a parameter, `X`, declared to be of this type, are the same as in the `VARRAY` example.

`collections` (`Varray` and nested table) are not supported if they are defined inside of a PL/SQL package specification. For example:

```
SQL> create package pkg as
   >   type vary is varray(10) of number;
   >   type ntbl is table of varchar2(100;
   >   procedure test(v in vary, n in ntbl);
   > end;
   > /
```

If a user selects the *test* procedure in the Adapter Configuration Wizard for stored procedures, an error occurs stating that the types are not supported. However, if the

vary and ntbl type definitions were defined at the root level, outside of the package, then choosing the *test* procedure works without issue. The supported way to use collection types (Varray and nested table) is shown in the following example:

```
SQL> create type vary as varray(10) of number;
SQL> create type ntbl as table of varchar2(10);
SQL> create package pkg as
   >   procedure test(v in vary, n in ntbl);
   > end;
   /
```

An OBJECT definition is also generated as a complexType. Its sequence holds one element for each attribute in the OBJECT.

The following OBJECT,

```
SQL> CREATE TYPE FOO AS OBJECT (X VARCHAR2 (10), Y NUMBER);
```

is represented as a complexType definition called FOO with two sequence elements.

```
<complexType name="FOO">
  <sequence>
    <element name="X" db:type="VARCHAR2" minOccurs="0" nillable="true"/>
      <simpleType>
        <restriction base="string">
          <maxLength value="10"/>
        </restriction>
      </simpleType>
    <element name="Y" type="decimal" db:type="NUMBER" minOccurs="0"
nillable="true"/>
  </sequence>
</complexType>
```

The minOccurs value is 0 to allow for the element to be removed from the XML file. This causes the value of the corresponding attribute in the OBJECT to be set to null at run time. The nillable value is true to allow empty elements to appear in the XML file, annotated with the xsi:nil attribute, to indicate that the value of the element is null. Again, the db:index attribute is not used.

Note the use of a restriction on the VARCHAR2 attribute. The length is known from the declaration of the attribute in the OBJECT.

9.7.3.6 Complex User-Defined Types

User-defined types can be defined in arbitrarily complex ways. An OBJECT can contain attributes whose types are defined as any of the user-defined types mentioned in the preceding section. The type of an attribute in an OBJECT can be another OBJECT, VARRAY, or a nested table, and so on. The base type of a VARRAY or a nested table can also be an OBJECT. Allowing the base type of a collection to be another collection supports multidimensional collections.

9.7.3.7 Object Type Inheritance

The Adapter Configuration Wizard is capable of generating a valid XSD for parameters whose types are defined using OBJECT-type inheritance. Given the following type hierarchy,

```
SQL> CREATE TYPE A AS OBJECT (A1 NUMBER, A2 VARCHAR2 (10)) NOT FINAL;
SQL> CREATE TYPE B UNDER A (B1 VARCHAR2 (10));
```

and a procedure containing a parameter, X, whose type is B,

```
SQL> CREATE PROCEDURE P (X IN B) AS BEGIN … END;
```

the Adapter Configuration Wizard generates an `InputParameters` element for parameter `X` as

```
<element name="X" type="db:B" db:index="1" db:type="Struct" minOccurs="0"
nillable="true"/>
```

where the definition of `OBJECT` type `B` in the XSD file is generated as the following `complexType`.

```
<complexType name="B">
  <sequence>
    <element name="A1" type="decimal" db:type="NUMBER" minOccurs="0"
nillable="true"/>
    <element name="A2" db:type="VARCHAR2" minOccurs="0"  nillable="true">
      ...
    </element>
    <element name="B1" db:type="VARCHAR2" minOccurs="0"  nillable="true">
      ...
    </element>
  </sequence>
</complexType>
```

Restrictions on the maximum length of attributes `A2` and `B1` are added appropriately. Notice how the `OBJECT` type hierarchy is flattened into a single sequence of elements that corresponds to all of the attributes in the entire hierarchy.

9.7.3.8 Object References

The Adapter Configuration Wizard can also generate a valid XSD for parameters that are references to `OBJECT` types (for example, object references) or are user-defined types that contain an object reference somewhere in their definition. In this example,

```
SQL> CREATE TYPE FOO AS OBJECT (…);
SQL> CREATE TYPE BAR AS OBJECT (F REF FOO, …);
SQL> CREATE PROCEDURE PROC (X OUT BAR, Y OUT REF FOO) AS BEGIN … END;
```

the Adapter Configuration Wizard generates `complexType` definitions for `FOO` and `BAR` as indicated, except that for `BAR`, the element for the attribute, `F`, is generated as

```
<element name="F" type="db:FOO" db:type="Ref" minOccurs="0" nillable="true"/>
```

where the `type` and `db:type` attribute values indicate that `F` is a reference to the `OBJECT` type `FOO`.

For a procedure `PROC`, the following elements are generated in the `OutputParameters` root element of the XSD file:

```
<element name="X" type="db:BAR" db:index="1" db:type="Struct" minOccurs="0"
nillable="true"/>
<element name="Y" type="db:FOO" db:index="2" db:type="Ref" minOccurs="0"
nillable="true"/>
```

For `Y`, note the value of the `db:type` attribute, `Ref`. with the `type` attribute, the element definition indicates that `Y` is a reference to `FOO`.

There is a restriction on the use of object references that limits their parameter mode to `OUT` only. Passing an `IN` or `IN/OUT` parameter into an API that is either directly a `REF` or, if the type of the parameter is user-defined, contains a `REF` somewhere in the definition of that type, is not permitted.

9.7.3.9 Referencing Types in Other Schemas

You can refer to types defined in other schemas if the necessary privileges to access them have been granted. For example, suppose type OBJ was declared in SCHEMA1:

```
SQL> CREATE TYPE OBJ AS OBJECT (…);
```

The type of a parameter in a stored procedure declared in SCHEMA2 can be type OBJ from SCHEMA1:

```
CREATE PROCEDURE PROC (O IN SCHEMA1.OBJ) AS BEGIN … END;
```

This is possible only if SCHEMA1 granted permission to SCHEMA2 to access type OBJ:

```
SQL> GRANT EXECUTE ON OBJ TO SCHEMA2;
```

If the required privileges are not granted, an error occurs when trying to create procedure PROC in SCHEMA2:

```
PLS-00201: identifier "SCHEMA1.OBJ" must be declared
```

Because the privileges have not been granted, type OBJ from SCHEMA1 is not visible to SCHEMA2; therefore, SCHEMA2 cannot refer to it in the declaration of parameter O.

9.7.3.10 XSD Pruning Optimization

Some user-defined object types can have a very large number of attributes. These attributes can also be defined in terms of other object types that also have many attributes. In short, one object type can become quite large depending on the depth and complexity of its definition.

Depending on the situation, many attributes of a large object type may not even be necessary. It is sometimes desirable to omit these attributes from the object's schema definition. This can be done by physically removing the unwanted XSD elements from the definition of the object type.

See the following example where a stored procedure has a parameter whose type is a complex user-defined type:

```
SQL> CREATE TYPE OBJ AS OBJECT (A, NUMBER, B <SqlType>, C <SqlType>, ...);
SQL> CREATE PROCEDURE PROC (O OBJ) AS BEGIN ... END;
```

The InputParameters root element contains a single element for the parameter, O from the API's signature. A complexType definition is to be added to the generated XSD for the object type, as shown in the following code snippet:

```
<complexType name="OBJ">
  <sequence>
    <element name="A" type="decimal" db:type="NUMBER" minOccurs="0"
nillable="true"/>
    <element name="B" .../>
    <element name="C" .../>
    ...
  </sequence>
</complexType>
```

If attributes B and C are not required, then their element in the complexType definition of OBJ can be removed regardless of its type. Values are not required for these attributes in the instance XML. If parameter O had been an output parameter, then elements corresponding with the pruned attributes are also omitted in the generated XML.

Suppose that the type of parameter A was also a user-defined object type and that the definition of OBJ changed accordingly, as shown in the following example:

```
SQL> CREATE TYPE FOO AS OBJECT (X NUMBER, Y NUMBER, Z NUMBER);
SQL> CREATE TYPE OBJ AS OBJECT (A FOO, B <SqlType>, C <SqlType, ...);
```

In such a case, the API remains unchanged. Elements corresponding to unwanted attributes in the definition of FOO can also be removed regardless of their type. So, for example, if Y is not required, then its element in the `complexType` definition of FOO can be removed in the XSD file.

Pruning the XSD file in this fashion improves the run-time performance of the adapter and can significantly reduce memory consumption, as well.

> **Note:** Only attributes in user-defined object types can be pruned. You cannot prune (remove) a parameter of the stored procedure by removing its element from the `InputParameters` root element. This can result in an error at run time unless the parameter has a default clause.

9.7.4 Run Time: Before Stored Procedure Invocation

This section discusses important considerations of stored procedure support and a brief overview of some important details regarding what happens before the invocation of a stored procedure or function.

This section includes the following topics:

- Section 9.7.4.1, "Value Binding"
- Section 9.7.4.2, "Data Type Conversions"

9.7.4.1 Value Binding

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

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

> **Note:** There is one notable difference that distinguishes Microsoft SQL Server from IBM DB2, MySQL, and AS/400. SQL Server supports parameters that can include a default value in the definition of a stored procedure. Because IBM DB2, MySQL, and AS/400 do not support parameter defaults, every parameter must be represented as an element in the instance XML.

In the first case, the value is taken from the XML file as is and is converted to the appropriate object according to its type. That object is then bound to its corresponding parameter during preparation of the stored procedure invocation.

In the second and third cases, the actual value extracted from the XML file is null. The type converter accepts null and returns it without any conversion. The null value is bound to its corresponding parameter regardless of its type. Essentially, this is equal to passing null for parameter X.

The fourth case has two possibilities. The parameter either has a default clause or it does not. If the parameter has a default clause, then the parameter can be excluded from the invocation of the stored procedure. This allows the default value to be used for the parameter. If the parameter is included, then the value of the parameter is used, instead. If the parameter does not have a default clause, then the parameter must be included in the invocation of the procedure. Elements for all parameters of a function must be specified. If an element in the instance XML is missing, then the function is invoked with fewer arguments than is expected.

A null value is bound to the parameter by default:

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

Here, no value is bound to the parameter. In fact, the parameter can be excluded from the invocation of the stored procedure. This allows the value of 0 to default for parameter X.

To summarize, the following PL/SQL is executed in each of these three cases:

1. `"BEGIN PROC (X=>?); END;" - X = 100`
2. `"BEGIN PROC (X=>?); END;" - X = null`
3. There are two possibilities:
   a. `"BEGIN PROC (); END;" - X = 0` (X has a default clause)
   b. `"BEGIN PROC (X=>?); END;" - X = null` (X does not have a default clause)

With the exception of default clause handling, these general semantics also apply to item values of a collection or attribute values of an OBJECT whose types are a supported primitive data types. The semantics of <X/> when the type is user-defined are, however, quite different.

For a collection, whether it is a VARRAY or a nested table, the following behavior can be expected, given a type definition such as

```
SQL> CREATE TYPE ARRAY AS VARRAY (5) OF VARCHAR2 (10);
```

and XML for a parameter, X, which has type ARRAY, that appears as follows:

```
<X>
    <X_ITEM xsi:nil="true"/>
    <X_ITEM>Hello</X_ITEM>
    <X_ITEM xsi:nil="true"/>
    <X_ITEM>World</X_ITEM>
</X>
```

The first and third elements of the VARRAY are set to null. The second and fourth are assigned their respective values. No fifth element is specified in the XML file; therefore, the VARRAY instance has only four elements.

Assume an OBJECT definition such as

```
SQL> CREATE TYPE OBJ AS OBJECT (A INTEGER, B INTEGER, C INTEGER);
```

and XML for a parameter, X, which has type OBJ, that appears as

```
<X>
    <A>100</A>
    <C xsi:nil="true"/>
</X>
```

The value `100` is assigned to attribute `A`, and null is assigned to attributes `B` and `C`. Because there is no element in the instance XML for attribute `B`, a null value is assigned.

The second case, `<X/>`, behaves differently if the type of `X` is user-defined. Rather than assigning null to `X`, an initialized instance of the user-defined type is created and bound instead.

In the preceding `VARRAY` example, if `<X/>` or `<X></X>` is specified, then the value bound to `X` is an empty instance of the `VARRAY`. In PL/SQL, this is equivalent to calling the type constructor and assigning the value to `X`. For example,

```
X := ARRAY();
```

Similarly, in the preceding `OBJECT` example, an initialized instance of `OBJ`, whose attribute values have all been null assigned, is bound to `X`. Similar to the `VARRAY` case, this is equivalent to calling the type constructor. For example,

```
X := OBJ(NULL, NULL, NULL);
```

To specifically assign a null value to `X` when the type of `X` is user-defined, add the `xsi:nil` attribute to the element in the XML file, as in

```
<X xsi:nil="true"/>
```

9.7.4.2 Data Type Conversions

This section describes the conversion of data types such as `CLOB`, `DATE`, `TIMESTAMP`, and binary data types including `RAW`, `LONG RAW` and `BLOB`, and similar data types supported by third-party databases.

Microsoft SQL Server, IBM DB2, AS/400, and MySQL support binding various forms of binary and date data types to parameters of a stored procedure, as summarized in Table 9–19.

Table 9–19 Third-Party Database: Binding Binary and Date Values to Parameters of a Stored Procedure

XML Schema Type	IBM DB2 Data Type	AS/400 Data Type	Microsoft SQL Server Data Type	MySQL Data Type
base64Binary	BLOB	BINARY	BINARY	BINARY
	CHAR FOR BIT DATA	BINARY LARGE OBJECT	IMAGE	TINYBLOB
	VARCHAR FOR BIT DATA	BINARY VARYING	TIMESTAMP	BLOB
			VARBINARY	MEDIUMBLOB
				LONGBLOB
				VARBINARY
dateTime	DATE	DATE	DATETIME	DATE
	TIME	TIME	SMALLDATETIME	DATETIME
	TIMESTAMP	TIMESTAMP		TIMESTAMP

For a `CLOB` parameter, if the length of the `CLOB` parameter is less than 4 kilobytes, then the text extracted from the XML file is bound to the parameter as a `String` type with

no further processing. If the length of the CLOB parameter is greater than 4 kilobytes or if the mode of the parameter is IN/OUT then a temporary CLOB parameter is created. The XML file data is then written to the temporary CLOB before the CLOB is bound to its corresponding parameter. The temporary CLOB parameter is freed when the interaction completes. For other character types, such as CHAR and VARCHAR2, the data is simply extracted and bound as necessary. It is possible to bind an XML document to a CLOB parameter (or VARCHAR2 if it is large enough). However, appropriate substitutions for <, >, and so on, must first be made (for example, < for < and > for >).

A few data types require special processing before their values are bound to their corresponding parameters. These include data types represented by the XML Schema types base64Binary and dateTime.

The XML schema type, dateTime, represents TIME, DATE, and TIMESTAMP. The XML values for these data types must adhere to the XML schema representation for dateTime. Therefore, a simple DATE string, 01-JAN-05, is invalid. XML schema defines dateTime as YYYY-MM-DDTHH:mm:ss. Therefore, the correct DATE value is 2005-01-01T00:00:00. Values for these parameters must be specified using this format in the instance XML.

Data for binary data types must be represented in a human readable manner. The chosen XML schema representation for binary data is base64Binary. The type converter uses the javax.mail.internet.MimeUtility encode and decode APIs to process binary data. The encode API must be used to encode all binary data into base64Binary form so that it can be used in an XML file. The type converter uses the decode API to decode the XML data into a byte array. The decode API is used to convert the base64Binary data into a byte array.

For a BLOB parameter, if the length of a byte array containing the decoded value is less than 2 kilobytes, then the byte array is bound to its parameter with no further processing. If the length of the byte array is greater than 2 kilobytes or if the mode of the parameter is IN/OUT, then a temporary BLOB is created. The byte array is then written to the BLOB before it is bound to its corresponding parameter. The temporary BLOB is freed when the interaction completes. For other binary data types, such as RAW and LONG RAW, the base64Binary data is decoded into a byte array and bound as necessary.

Conversions for the remaining data types are straightforward and require no additional information.

9.7.5 Run Time: After Stored Procedure Invocation

After the procedure (or function) executes, the values for any IN/OUT and OUT parameters are retrieved. These correspond to the values of the elements in the OutputParameters root element in the generated XSD.

This section includes the following topics:

- Section 9.7.5.1, "Data Type Conversions"
- Section 9.7.5.2, "Null Values"
- Section 9.7.5.3, "Function Return Values"

9.7.5.1 Data Type Conversions

Conversions of data retrieved are straightforward. However, CLOB (and other character data), RAW, LONG RAW, and BLOB conversions, and conversions for similar data types supported by third-party databases, require special attention.

When a CLOB is retrieved, the entire contents of that CLOB are written to the corresponding element in the generated XML. Standard DOM APIs are used to construct the XML file. Hence, character data, for types such as CLOB, CHAR, and VARCHAR2, is messaged as needed to make any required substitutions so that the value is valid and can be placed in the XML file for subsequent processing. Therefore, substitutions for <and>, for example, in an XML document stored in a CLOB are made so that the value placed in the element within the generated XML for the associated parameter is valid.

Raw data, such as for RAW and LONG RAW data types, is retrieved as a byte array. For 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 file for the corresponding element. The MimeUtility decode API must be used to decode this value back into a byte array.

Conversions for the remaining data types are straightforward and require no additional information.

9.7.5.2 Null Values

Elements whose values are null appear as empty elements in the generated XML and are annotated with the xsi:nil attribute. Thus, the xsi namespace is declared in the XML file that is generated. Generated XML for a procedure PROC, which has a single OUT parameter, X, whose value is null, looks as follows:

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

XML elements for parameters of any type (including user-defined types) appear this way if their value is null.

9.7.5.3 Function Return Values

The return value of a function is treated as an OUT parameter at position 0 whose name is the name of the function itself. For example,

```
CREATE FUNCTION FACTORIAL (X IN INTEGER) RETURN INTEGER AS
BEGIN
    IF (X <= 0) THEN RETURN 1;
    ELSE RETURN FACTORIAL (X - 1);
    END IF;
END;
```

An invocation of this function with a value of 5, for example, results in a value of 120 and appears as <FACTORIAL>120</FACTORIAL> in the OutputParameters root element in the generated XML.

9.7.6 Run Time: Common Third-Party Database Functionality

The common third-party database functionality at run time includes the following:

- Section 9.7.6.1, "Processing ResultSets"
- Section 9.7.6.2, "Returning an INTEGER Status Value"

9.7.6.1 Processing ResultSets

All third-party databases share the same functionality for handling `ResultSets`. The following is a SQL Server example of an API that returns a `ResultSet`:

```
1> create procedure foo ... as select ... from ...;
2> go
```

A `RowSet` defined in the generated XSD represents a `ResultSet`. A `RowSet` consists of zero or more rows, each having one or more columns. A row corresponds with a row returned by the query. A column corresponds with a column item in the query. The generated XML for the API shown in the preceding example after it executes is shown in the following example:

```
<RowSet>
  <Row>
    <Column name="<column name>" sqltype="<sql datatype">value</Column>
    ...
  </Row>
    ...
</RowSet>
…
```

The `name` attribute stores the name of the column appearing in the query while the `sqltype` attribute stores the SQL datatype of that column, for example `INT`. The *value* is whatever the value is for that column.

It is possible for an API to return multiple `ResultSets`. In such cases, there is one `RowSet` for each `ResultSet` in the generated XML. All `RowSets` always appear first in the generated XML.

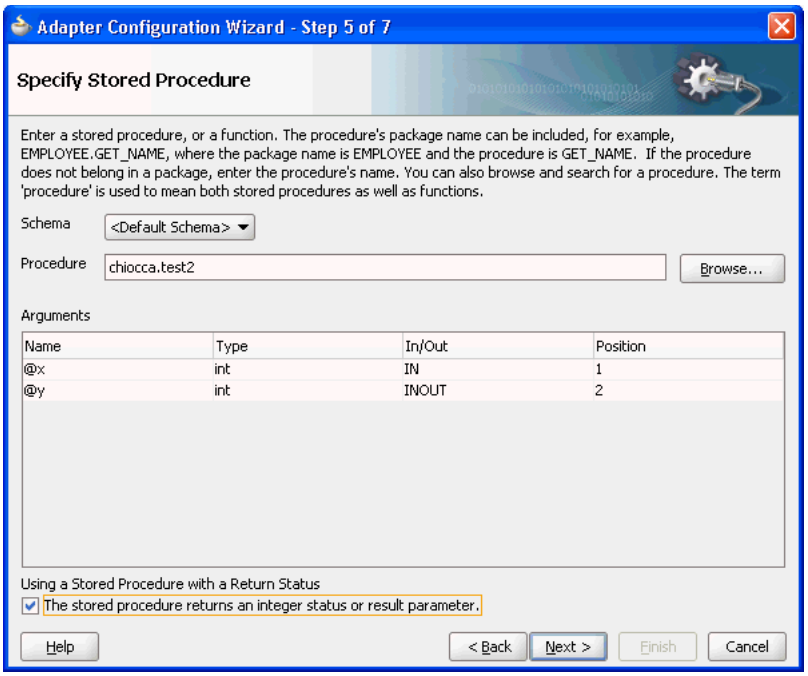
9.7.6.2 Returning an INTEGER Status Value

Some databases support returning an `INTEGER` status value using a `RETURN` statement in a stored procedure. Microsoft SQL Server and AS/400 both support this feature. In both cases, the Adapter Configuration Wizard cannot determine whether a stored procedure returns a status value. Therefore, you must specify that the stored procedure is returning a value. You can use a check box to make this indication.

After choosing a stored procedure in the **Stored Procedures** dialog, the Specify Stored Procedure page appears, as shown in Figure 9–51. The check box appears at the bottom of the page. Select the box to indicate that the procedure contains a `RETURN` statement. You can view the source code of the procedure to determine whether a `RETURN` statement exists.

The check box appears only for stored procedures on databases that support this feature. The check box is not displayed for functions. The value returned by the stored procedure appears as an element in the `OutputParameters` root element in the generated XSD. The name of the element is the name of the stored procedure. The value of a `return` statement is lost after the execution of the stored procedure if the check box is not selected.

Figure 9–51 The Specify Stored Procedure Page



9.7.7 Advanced Topics

This section discusses scenarios for types that are not supported directly using the stored procedure functionality that the Oracle Database Adapter provides. The following sections describe workarounds that address the have to use these data types:

- Section 9.7.7.1, "Row Set Support Using a Strongly Typed XSD"
- Section 9.7.7.2, "Row Set Support Using a Weakly Typed XSD"
- Section 9.7.7.3, "Support for PL/SQL Boolean, PL/SQL Record, and PL/SQL Table Types"

9.7.7.1 Row Set Support Using a Strongly Typed XSD

Currently a `REF CURSOR` by nature can support any arbitrary result set, so the XSD generated at design time is weakly typed.

However the XML output from this is hard to use. It is very difficult to write an Xpath expression or XSL based on a weakly typed XSD and column names as attribute values instead of element names.

Although a row set can represent any result set, it is possible to assume for some procedures that it has the same structure each time, and hence can be described with a strongly typed XSD. A strongly typed XSD is almost a necessity to transform the result set to another XSD later on. You can use the Adapter Configuration Wizard to generate a strongly typed XSD for a `REF CURSOR`.

If a weakly typed XSD is sufficient for your use case, see Section 9.7.7.2, "Row Set Support Using a Weakly Typed XSD".

This section includes the following topics:

- Section 9.7.7.1.1, "Design Time"
- Section 9.7.7.1.2, "Run Time"

For more information, see Section 9.3.3, "Row Set Support Using a Strongly or Weakly Typed XSD".

9.7.7.1.1 Design Time

If the stored procedure or function you select contains an output parameter of type `RowSet`, you can define a strongly typed XSD for this ref cursor as follows:

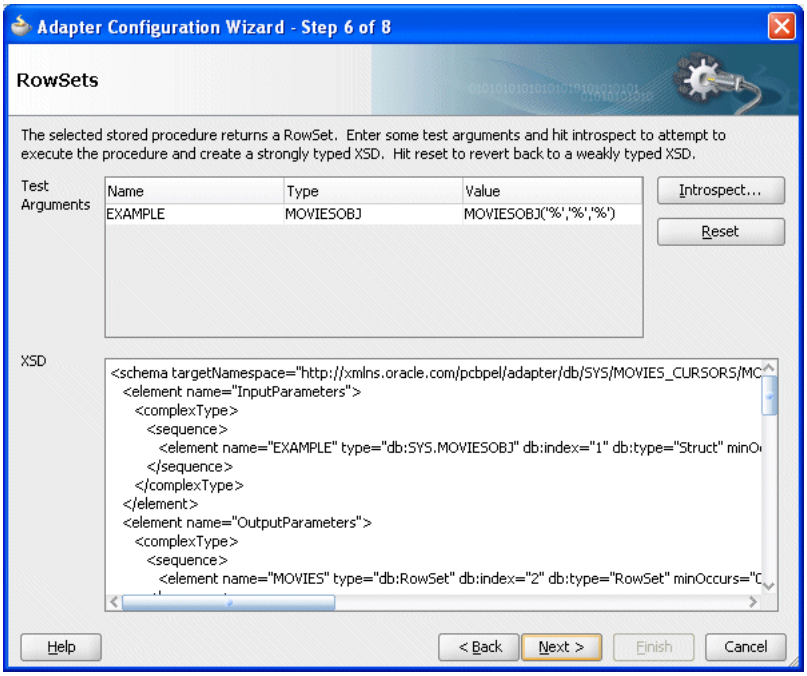
1. Using the Adapter Configuration Wizard, select a stored procedure or function that contains an output parameter of type `RowSet`.

   See steps 1 through 8 in Section 9.7.1.1, "Using Top-Level Standalone APIs".

2. Click **Next**. The RowSets page is displayed, as shown in Figure 9–52.

   By default, the Adapter Configuration Wizard generates a weakly typed XSD for this ref cursor shown in the **XSD** text field. Example 9–4 shows this default, weakly typed XSD.

Figure 9–52 RowSets Page



Example 9–4 Default Weakly Typed XSD

```
<schema targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/db/SYS/MOVIES_
CURSORS/MOVIES_QUERY/" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:db="http://xmlns.oracle.com/pcbpel/adapter/db/SYS/MOVIES_CURSORS/MOVIES_QUERY/"
elementFormDefault="qualified">
   <element name="InputParameters">
      <complexType>
         <sequence>
            <element name="EXAMPLE" type="db:SYS.MOVIESOBJ" db:index="1" db:type="Struct"
minOccurs="0" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <element name="OutputParameters">
      <complexType>
         <sequence>
            <element name="MOVIES" type="db:RowSet" db:index="2" db:type="RowSet"
```

```
minOccurs="0" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="RowSet">
      <sequence>
         <element name="Row" minOccurs="0" maxOccurs="unbounded">
            <complexType>
               <sequence>
                  <element name="Column" maxOccurs="unbounded" nillable="true">
                     <complexType>
                        <simpleContent>
                           <extension base="string">
                              <attribute name="name" type="string" use="required"/>
                              <attribute name="sqltype" type="string" use="required"/>
                           </extension>
                        </simpleContent>
                     </complexType>
                  </element>
               </sequence>
            </complexType>
         </element>
      </sequence>
   </complexType>
   <complexType name="SYS.MOVIESOBJ">
      <sequence>
         <element name="TITLE" db:type="VARCHAR2" minOccurs="0" nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
               </restriction>
            </simpleType>
         </element>
         <element name="DIRECTOR" db:type="VARCHAR2" minOccurs="0" nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
               </restriction>
            </simpleType>
         </element>
         <element name="STARRING" db:type="VARCHAR2" minOccurs="0" nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
               </restriction>
            </simpleType>
         </element>
      </sequence>
   </complexType>
</schema>
```

3. For each of the stored procedure or function arguments:
   - Double-click in the **Value** column.
   - Enter a valid value for the argument.

     Enter numbers and strings directly, dates as literals (for example, 2009/11/11), and structs as say `MYOBJ('a', 'b')`.
   - Press **Enter**.

Note: You must choose values that are valid for the argument type and that exist in the database.

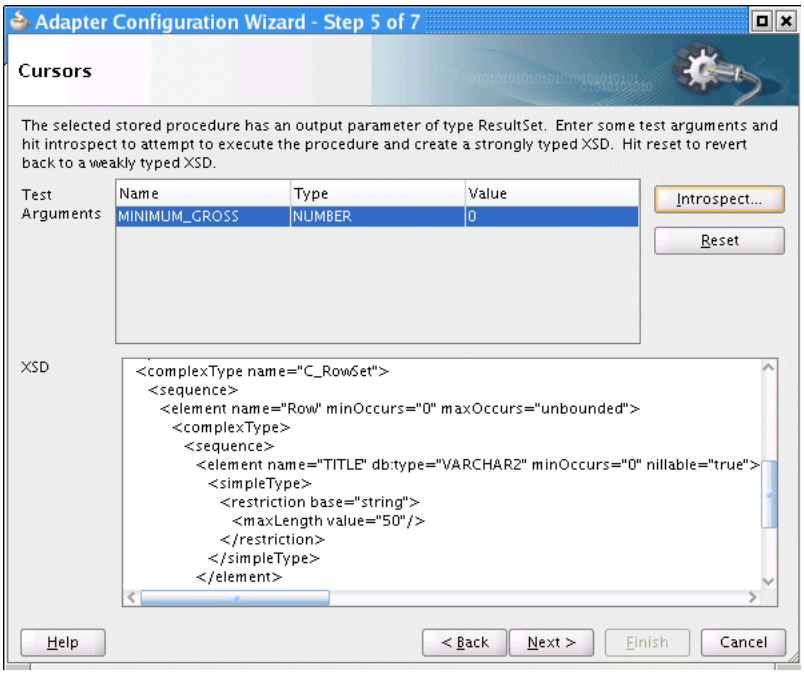
Oracle recommends that you specify a value for all arguments to ensure that the correct stored procedure or function signature is executed.

4. Click **Introspect**.

   The Adapter Configuration Wizard executes the stored procedure or function using the arguments you specify:

   a. If the stored procedure or function returns a row set with at least 1 row, the RowSets page is updated to display a strongly typed XSD in the **XSD** text field. Example 9–5 shows the strongly typed XSD that replaces the default, weakly typed XSD that Example 9–4 shows.

Figure 9–53 RowSets Page: Successful Introspection



Example 9–5 Strongly Typed XSD

```
<schema targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/db/SYS/MOVIES_
CURSORS/MOVIES_QUERY/" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:db="http://xmlns.oracle.com/pcbpel/adapter/db/SYS/MOVIES_CURSORS/MOVIES_
QUERY/" elementFormDefault="qualified">
   <element name="InputParameters">
      <complexType>
         <sequence>
            <element name="EXAMPLE" type="db:SYS.MOVIESOBJ" db:index="1"
db:type="Struct" minOccurs="0" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <element name="OutputParameters">
      <complexType>
```

```
         <sequence>
            <element name="MOVIES" type="db:MOVIES_RowSet" db:index="2"
db:type="RowSet" minOccurs="0" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="MOVIES_RowSet">
      <sequence>
         <element name="MOVIES_Row" minOccurs="0" maxOccurs="unbounded">
            <complexType>
               <sequence>
                  <element name="TITLE" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="50"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="DIRECTOR" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="20"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="STARRING" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="100"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="SYNOPSIS" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="255"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="GENRE" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="70"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="RUN_TIME" type="decimal" db:type="NUMBER"
minOccurs="0" nillable="true"/>
                  <element name="RELEASE_DATE" type="dateTime" db:type="DATE"
minOccurs="0" nillable="true"/>
                  <element name="RATED" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="6"/>
```

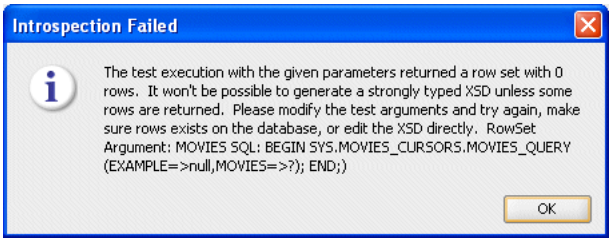
```
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="RATING" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="4"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="VIEWER_RATING" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="5"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="STATUS" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="11"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="TOTAL_GROSS" type="decimal" db:type="NUMBER"
minOccurs="0" nillable="true"/>
                  <element name="DELETED" db:type="VARCHAR2" minOccurs="0"
nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="5"/>
                        </restriction>
                     </simpleType>
                  </element>
                  <element name="SEQUENCENO" type="decimal" db:type="NUMBER"
minOccurs="0" nillable="true"/>
                  <element name="LAST_UPDATED" type="dateTime" db:type="DATE"
minOccurs="0" nillable="true"/>
                  <element name="POLLING_STRATEGY" db:type="VARCHAR2"
minOccurs="0" nillable="true">
                     <simpleType>
                        <restriction base="string">
                           <maxLength value="30"/>
                        </restriction>
                     </simpleType>
                  </element>
               </sequence>
            </complexType>
         </element>
      </sequence>
   </complexType>
   <complexType name="SYS.MOVIESOBJ">
      <sequence>
         <element name="TITLE" db:type="VARCHAR2" minOccurs="0" nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
```

```
               </restriction>
            </simpleType>
         </element>
         <element name="DIRECTOR" db:type="VARCHAR2" minOccurs="0"
nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
               </restriction>
            </simpleType>
         </element>
         <element name="STARRING" db:type="VARCHAR2" minOccurs="0"
nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
               </restriction>
            </simpleType>
         </element>
      </sequence>
   </complexType>
</schema>
```

Proceed to step 5.

b. If no rows are returned, the Introspection Failed dialog is displayed, as shown in Figure 9–54.

Figure 9–54 Introspection Failed Dialog

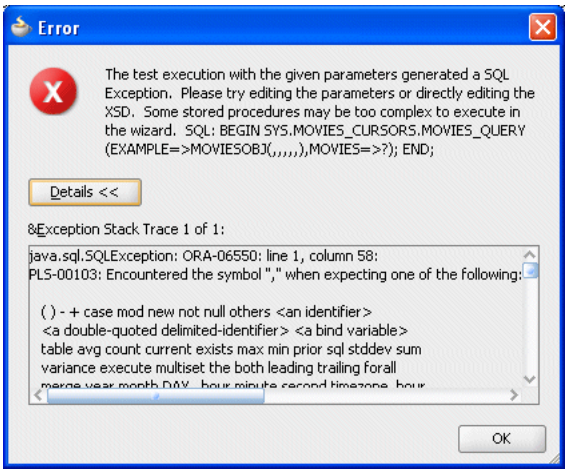


The Adapter Configuration Wizard generates a weakly typed XSD and displays it in the XSD text field by default, overwriting any edits you may have made to a previous version of the XSD.

Go back to step 3 and enter test argument values that returns a row set with at least 1 row.

c. If the stored procedure or function throws an exception, the Introspection Error dialog is displayed, as shown in Figure 9–55.

Figure 9–55 Introspection Error Dialog



The Adapter Configuration Wizard generates a weakly typed XSD and displays it in the XSD text field by default, overwriting any edits you may have made to a previous version of the XSD.

Go back to step 3 and enter test argument values that returns a row set with at least 1 row.

5. Optionally, fine tune the strongly typed XSD by manually editing the schema shown in the **XSD** text filed.
6. Proceed to step 10 in Section 9.7.1.1, "Using Top-Level Standalone APIs".

9.7.7.1.2 Run Time

Suppose you have the following package:

```
CREATE PACKAGE PKG AS
       TYPE REF_CURSOR IS REF CURSOR;
       PROCEDURE TEST(C OUT REF_CURSOR);
END;

CREATE PACKAGE BODY PKG AS
       ROCEDURE TEST(C OUT REF_CURSOR) AS
       BEGIN
           OPEN C FOR SELECT DEPTNO, DNAME FROM DEPT;
       END;
END;
```

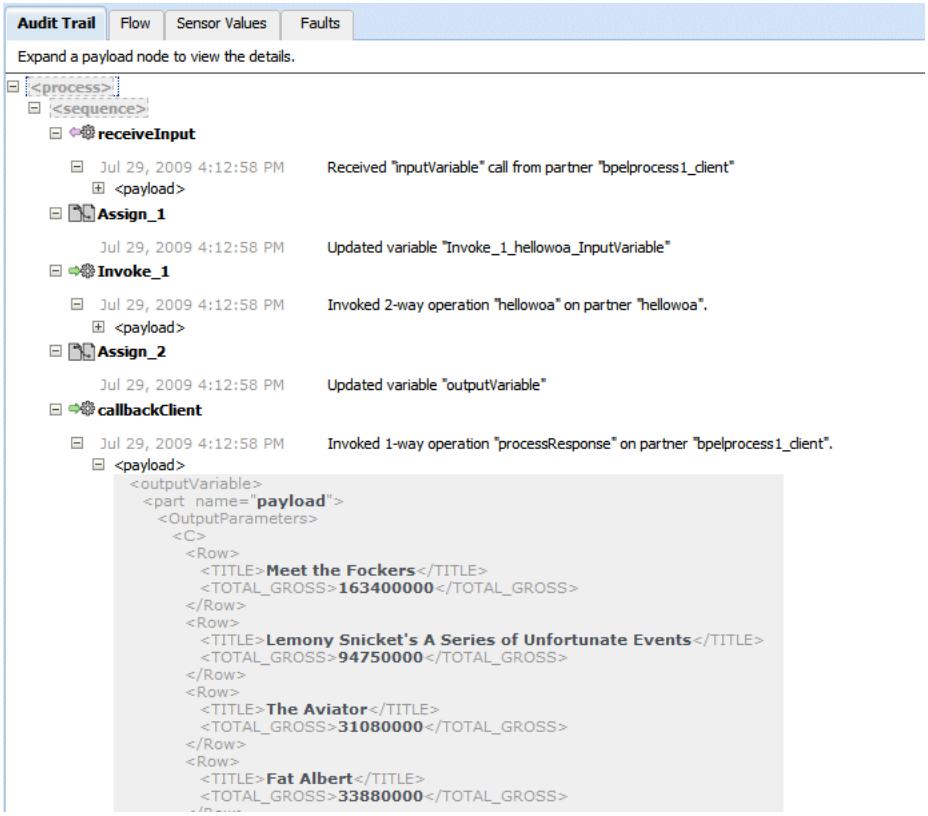
After using the Adapter Configuration Wizard to define a strongly typed XSD, after the procedure executes, the following XML is generated for parameter, `C`:

```
<C>
    <C_Row>
        <DEPTNO>10</DEPTNO>
        <DNAME>ACCOUNTING</DNAME>
    </C_Row>
    <C_Row>
        <DEPTNO>11</DEPTNO>
        <DNAME>DEVELOPMENT</DNAME>
    </C_Row>
    …
</C>
```

Using the Oracle Database Adapter, at run time, it does not matter if the XSD describing the strongly typed ref cursor is inline or imported.

The strongly typed XSD is applied by the SOA runtime and is visible in the Oracle Enterprise Manager Console, where appropriate. For example, Figure 9–56 shows the audit trail for an invoke that returns a ref cursor payload using a strongly typed XSD.

Figure 9–56 Audit Trail for Stongly Typed Payload



9.7.7.2 Row Set Support Using a Weakly Typed XSD

Currently a `REF CURSOR` by nature can support any arbitrary result set, so the XSD generated at design time is weakly typed. By default, the Adapter Configuration Wizard generates a weakly typed XSD for a `REF CURSOR`.

However the XML output from this is hard to use. It is very difficult to write an Xpath expression or XSL based on a weakly typed XSD and column names as attribute values instead of element names.

Although a row set can represent any result set, it is possible to assume for some procedures that it has the same structure each time, and hence can be described with a strongly typed XSD. A strongly typed XSD is almost a necessity to transform the result set to another XSD later on.

If a strongly typed XSD is better suited to your use case, see Section 9.7.7.1, "Row Set Support Using a Strongly Typed XSD".

This section includes the following topics:

- Section 9.7.7.2.1, "Design Time"
- Section 9.7.7.2.2, "Run Time"

For more information, see Section 9.3.3, "Row Set Support Using a Strongly or Weakly Typed XSD".

9.7.7.2.1 Design Time

If the stored procedure or function you select contains an output parameter of type `ResultSet`, you can define a weakly typed XSD for this ref cursor as follows:

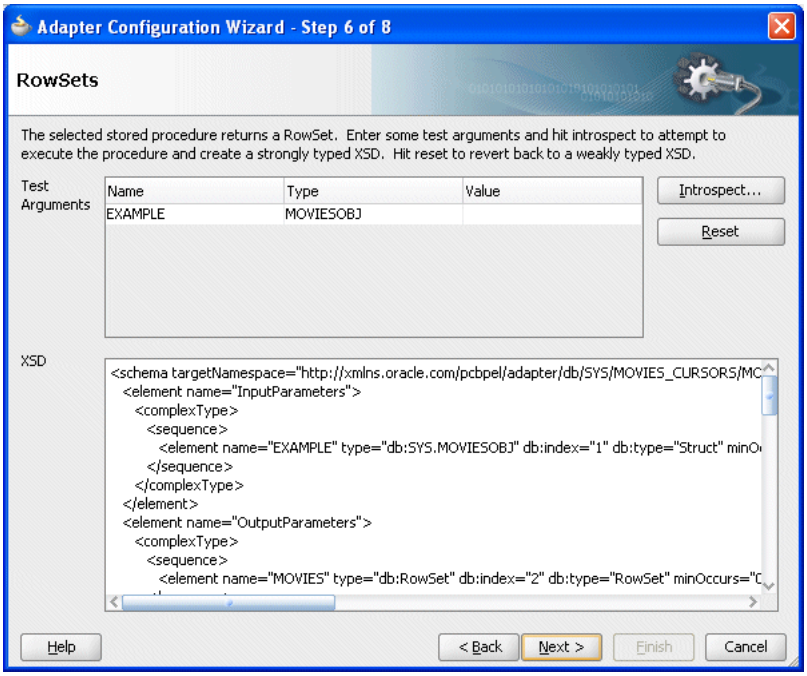
1. Using the Adapter Configuration Wizard, select a stored procedure or function that contains an output parameter of type `ResultSet`.

   See steps 1 through 8 in Section 9.7.1.1, "Using Top-Level Standalone APIs".

2. Click **Next**. The RowSets page is displayed, as shown in Figure 9–57.

   By default, the Adapter Configuration Wizard generates a weakly typed XSD for this ref cursor shown in the **XSD** text field.

Figure 9–57 RowSets Page



3. Optionally, fine tune the weakly typed XSD by manually editing the schema shown in the **XSD** text filed.

4. Proceed to step 10 in Section 9.7.1.1, "Using Top-Level Standalone APIs".

9.7.7.2.2 Run Time

Suppose you have the following package:

```
CREATE PACKAGE PKG AS
       TYPE REF_CURSOR IS REF CURSOR;
       PROCEDURE TEST(C OUT REF_CURSOR);
END;

CREATE PACKAGE BODY PKG AS
       ROCEDURE TEST(C OUT REF_CURSOR) AS
       BEGIN
           OPEN C FOR SELECT DEPTNO, DNAME FROM DEPT;
```

```
       END;
END;
```

The REF_CURSOR is a weakly typed cursor variable because the query is not specified. After the procedure executes, the following XML is generated for parameter, C:

```
<C>
    <Row>
        <Column name="DEPTNO" sqltype="NUMBER">10</Column>
        <Column name="DNAME" sqltype="VARCHAR2">ACCOUNTING</Column>
    </Row>
    <Row>
        <Column name="DEPTNO" sqltype="NUMBER">20</Column>
        <Column name="DNAME" sqltype="VARCHAR2">RESEARCH</Column>
    </Row>
    …
</C>
```

There is a total of four rows, each consisting of two columns, DEPTNO and DNAME.

Ref cursors are represented by Java ResultSets. It is not possible to create a ResultSet programmatically by using APIs provided by the JDBC driver. Therefore, ref cursors may not be passed IN to a stored procedure. They can only be passed as IN/OUT and OUT parameters with one caveat. An IN/OUT ref cursor is treated strictly as an OUT parameter. Because no IN value can be provided for an IN/OUT parameter, a null is bound to that parameter when invoking the stored procedure.

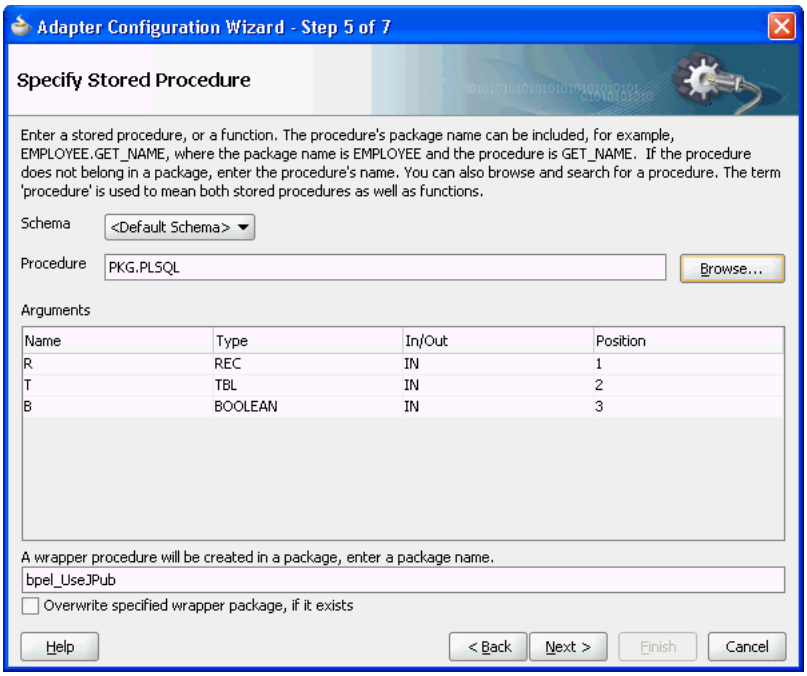
9.7.7.3 Support for PL/SQL Boolean, PL/SQL Record, and PL/SQL Table Types

The Adapter Configuration Wizard provides a mechanism that detects when these types are used and then invokes Oracle JPublisher to generate the necessary wrappers automatically. Oracle JPublisher generates two SQL files, one to create schema objects, and another to drop them. The SQL that creates the schema objects is automatically executed from within the Adapter Configuration Wizard to create the schema objects in the database schema before the XSD file is generated. For example, suppose the following package specification is declared:

```
CREATE PACKAGE PKG AS
  TYPE REC IS RECORD (X NUMBER, Y VARCHAR2 (10));
  TYPE TBL IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
  PROCEDURE PLSQL (R REC, T TBL, B BOOLEAN);
END;
```

Figure 9–58 shows the step in the Adapter Configuration Wizard that is displayed when PROC procedure from PKG package is selected.

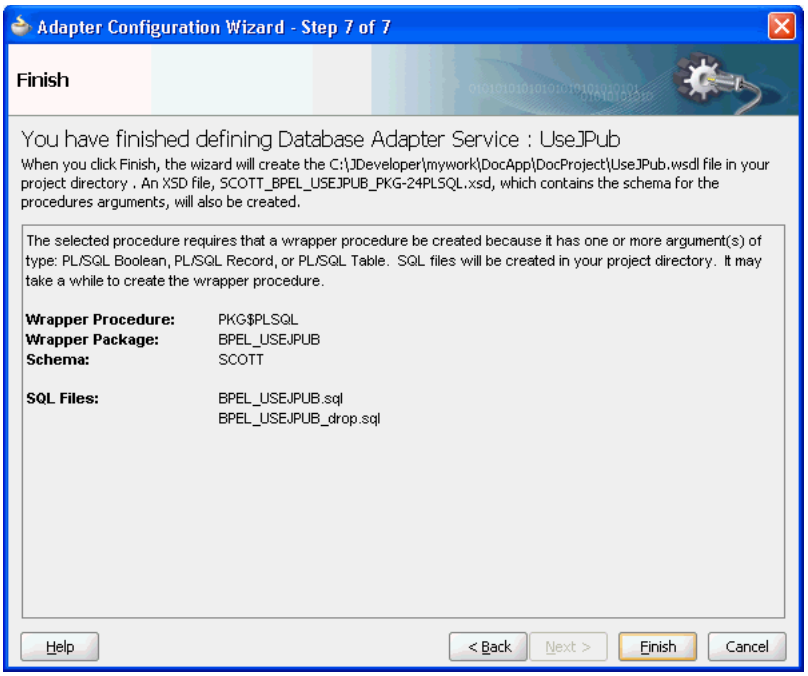
Figure 9–58 Specifying a Stored Procedure in the Adapter Configuration Wizard



As Figure 9–58 shows, the original procedure name is fully qualified, `PKG.PLSQL`. The type of parameter, `R`, is the name of the `RECORD`. The type of `T` is the name of the `TABLE`. The type of `B` is `Boolean`. The name of the wrapper package that is generated is derived from the service name, `bpel_`*`ServiceName`* (for example, `bpel_UseJPub`). This is the name of the generated package that contains the wrapper procedure. You can use the check box to force the Adapter Configuration Wizard to overwrite an existing package when the schema objects are created.

Clicking **Next** twice reveals the **Finish** page of the Adapter Configuration Wizard, as shown in Figure 9–59.

Figure 9–59 Defining a Database Adapter Service: Finish Page



The contents of this page describe what the Adapter Configuration Wizard has detected and what actions are performed when the **Finish** button is clicked. The following summarizes the contents of this page:

1. The name of the generated WSDL is `UseJPub.wsdl`.
2. The name of the JCA file is `UseJPub_db.jca`.
3. Two SQL scripts are created and added to the BPEL process project:
   a. `BPEL_USEJPUB.sql` – Creates the schema objects.
   b. `BPEL_USEJPUB_drop.sql` – Drops the schema objects.
4. The name of the generated XSD is `SCOTT_USEJPUB_PKG-24PLSQL.xsd`.

When you click **Finish**, Oracle JPublisher is invoked to generate the SQL files and load the schema objects into the database. The process of generating wrappers may take quite some time to complete. Processing times for wrappers that are generated in the same package usually require less time after an initial wrapper has been generated for another procedure within the same package.

> **Note:** You must execute `BPEL_XXXX_drop.sql` when re-creating an Oracle Database Adapter. This is likely due to the JPublisher functionality, which uses a cache when generating wrappers.

The following user-defined types are generated to replace the PL/SQL types from the original procedure:

```
SQL> CREATE TYPE PKG_REC AS OBJECT (X NUMBER, Y VARCHAR2 (10));
SQL> CREATE TYPE PKG_TBL AS TABLE OF NUMBER;
```

The naming convention for these types is *`OriginalPackageName_OriginalTypeName`*. `Boolean` is replaced by `INTEGER` in the wrapper procedure.

Acceptable values for the original `Boolean` parameter, now that it is an `INTEGER` are `0` for `FALSE` and any nonFzero `INTEGER` value for `TRUE`. Any value other than `1` is considered false. The generated wrapper procedure uses APIs from the `SYS.SQLJUTL` package to convert from `INTEGER` to `Boolean` and vice-versa.

A new wrapper package called `BPEL_USEJPUB` is created that contains the wrapper for procedure `PLSQL`, called `PKG$PPLSQL`, and conversion APIs that convert from the PL/SQL types to the user-defined types and vice-versa. If the original procedure is a root-level procedure, then the name of the generated wrapper procedure is `TOPLEVEL$`*`OriginalProcedureName`*.

The generated XSD represents the signature of wrapper procedure `PKG$PLSQL` and not the original procedure. The name of the XSD file is URL-encoded, which replaces `$` with `-24`.

Note the naming conventions for the generated artifacts:

- The service name is used in the names of the WSDL and SQL files. It is also used as the name of the wrapper package.
- The name of the generated XSD is derived from the schema name, service name, and the original package and procedure names.
- The name of a SQL object or collection data types are derived from the original package name and the name of its corresponding PL/SQL type.
- The name of the wrapper procedure is derived from the original package and procedure names. `TOPLEVEL$` is used for root-level procedures.

The name of the generated wrapper package is limited to 30 characters. The name of the wrapper procedure is limited to 29 characters. If the names generated by Oracle JPublisher are longer than these limits, then they are truncated.

When the PartnerLink that corresponds with the service associated with the procedure is invoked, then the generated wrapper procedure is executed instead of the original procedure.

9.7.7.3.1 Default Clauses in Wrapper Procedures

If a procedure contains a special type that requires a wrapper to be generated, then the default clauses on any of the parameters are *not* carried over to the wrapper. For example, consider

```
SQL> CREATE PROCEDURE NEEDSWRAPPER (
        >     B BOOLEAN DEFAULT TRUE, N NUMBER DEFAULT 0) IS BEGIN … END;
```

Assuming that this is a root-level procedure, the signature of the generated wrapper procedure is

```
TOPLEVEL$NEEDSWRAPPER (B INTEGER, N NUMBER)
```

The `Boolean` type has been replaced by `INTEGER`. The default clauses on both parameters are missing in the generated wrapper. Parameters of generated wrapper procedures never have a default clause even if they did in the original procedure.

In this example, if an element for either parameter is not specified in the instance XML, then an error occurs stating that an incorrect number of arguments have been provided. The default value of the parameter that is specified in the original procedure is not used.

To address this situation, the generated SQL file that creates the wrapper must be edited, restoring the default clauses to the parameters of the wrapper procedure. The wrapper and any additional schema objects must then be reloaded into the database

schema. After editing the SQL file, the signature of the wrapper procedure is as follows:

```
TOPLEVEL$NEEDSWRAPPER (B INTEGER DEFAULT 1, N NUMBER DEFAULT 0)
```

For `Boolean` parameters, the default value for true is `1`, and the default value for false is `0`.

As a final step, the XSD file generated for the wrapper must be edited. A special attribute must be added to elements representing parameters that now have default clauses. Add `db:default="true"` to each element representing a parameter that now has a default clause. For example,

```
<element name="B" … db:default="true" …/>
<element name="N" … db:default="true" …/>
```

This attribute is used at run time to indicate that if the element is missing from the instance XML, then the corresponding parameter must be omitted from the procedure call. The remaining attributes of these elements remain exactly the same.

9.8 Oracle Database Adapter Use Cases

This describes the Oracle Database Adapter and Oracle Database Adapter - stored procedures use cases.

This section includes the following topics:

- Section 9.8.1, "Use Cases for Oracle Database Adapter"
- Section 9.8.2, "Use Cases for Oracle Database Adapter - Stored Procedures"

9.8.1 Use Cases for Oracle Database Adapter

To obtain Oracle Database Adapter use cases, access the Oracle SOA Sample Code site.

Table 9–20 shows the Oracle Database Adapter samples that are provided with Oracle BPEL PM, and Mediator.

Table 9–20 Oracle Database Adapter Use Cases

Tutorial Name	Description
`Delete`	Illustrates the outbound `delete` operation of the Oracle Database Adapter. An XML record is passed to the operation and the row in the database with the same primary key is deleted.
`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 Oracle 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.)
`InsertWithCatch`	Illustrates the extra steps (based on the `Insert` tutorial) needed to add fault handling to your BPEL process.

Table 9–20 (Cont.) Oracle Database Adapter Use Cases

Tutorial Name	Description
JPublisherWrapper	Illustrates a workaround for using PL/SQL RECORD types. JPublisher is used to create a corresponding OBJECT type whose attributes match the fields of the RECORD, and conversion APIs that convert from RECORD to OBJECT and vice versa. JPublisher also generates a wrapper procedure (or function) that accepts the OBJECT and invokes the underlying method using the conversion APIs in both directions. The invoked methods must be installed in an Oracle database (not Oracle Lite).
MasterDetail	Illustrates how to migrate data from one set of tables to another. The sample uses the Oracle 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 Oracle 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 PM. 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 PM. 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 PM. 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 PM. 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.
QueryByExample	Illustrates the outbound queryByExample operation of the Oracle Database Adapter. A SELECT SQL query is built dynamically based on fields set in an example XML record, and any matching records are returned.
RefCursors	Illustrates how to use a REF CURSOR with a strongly typed or weakly typed XSD. You can use the Adapter Configuration Wizard to create a strongly typed XSD for a row set returned by an Oracle Database stored procedure or function REF CURSOR variable. For more information, see Section 9.3.3, "Row Set Support Using a Strongly or Weakly Typed XSD".
ResultSetConverter	Illustrates a workaround for using a REF CURSOR. The solution involves the use of a Java stored procedure to convert the corresponding java.sql.ResultSet into a collection (either VARRAY or NESTED TABLE) of OBJECTs.

Table 9–20 (Cont.) Oracle Database Adapter Use Cases

Tutorial Name	Description
`SelectAll`	Illustrates the outbound `SelectAll` operation of the Oracle Database Adapter. With no `WHERE` clause, all rows in the `MOVIES` table are returned as XML.
`SelectAllByTitle`	Illustrates the outbound `SelectAllByTitle` operation of the Oracle Database Adapter. The row in the `MOVIES` table with the selected title is returned as XML.
`Update`	Illustrates the outbound `Update` operation of the Oracle 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 9–3 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.

9.8.2 Use Cases for Oracle Database Adapter - Stored Procedures

This section includes the following use cases:

- Section 9.8.2.1, "Creating and Configuring a Stored Procedure in JDeveloper BPEL Designer"
- Section 9.8.2.2, "File To StoredProcedure Use Case"

In addition to the uses cases documented in this section, refer to the sample Oracle Database Adapter use cases available by accessing the Oracle SOA Sample Code site.

Table 9–21 shows the Oracle Database Adapter stored procedure samples that are provided with Oracle BPEL PM, and Mediator.

Table 9–21 Oracle Database Adapter Use Cases - Stored Procedures

Tutorial Name	Description
`JPublisherWrapper`	Illustrates a workaround for using PL/SQL `RECORD` types. JPublisher is used to create a corresponding `OBJECT` type whose attributes match the fields of the `RECORD`, and conversion APIs that convert from `RECORD` to `OBJECT` and vice versa. JPublisher also generates a wrapper procedure (or function) that accepts the `OBJECT` and invokes the underlying method using the conversion APIs in both directions. The invoked methods must be installed in an Oracle database (not Oracle Lite).
`RefCursors`	Illustrates how to use a `REF CURSOR` with a strongly typed or weakly typed XSD. You can use the Adapter Configuration Wizard to create a strongly typed XSD for a row set returned by an Oracle Database stored procedure or function `REF CURSOR` variable. For more information, see Section 9.3.3, "Row Set Support Using a Strongly or Weakly Typed XSD".
`ResultSetConverter`	Illustrates a workaround for using a `REF CURSOR`. The solution involves the use of a Java stored procedure to convert the corresponding `java.sql.ResultSet` into a collection (either `VARRAY` or `NESTED TABLE`) of OBJECTs.

See Table 9–3 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.

9.8.2.1 Creating and Configuring a Stored Procedure in JDeveloper BPEL Designer

This use case describes how to integrate a stored procedure into BPEL Process Manager with JDeveloper BPEL Designer.

This use case includes of the following sections:

- Section 9.8.2.1.1, "Prerequisites"
- Section 9.8.2.1.2, "Creating an Application and an SOA Composite"
- Section 9.8.2.1.3, "Creating the Outbound Oracle Database Adapter Service"
- Section 9.8.2.1.4, "Add an Invoke Activity"
- Section 9.8.2.1.5, "Change the Message Part of the Request Message"
- Section 9.8.2.1.6, "Change the Message Part of the Response Message"
- Section 9.8.2.1.7, "Add a Assign Activity for the Input Variable"
- Section 9.8.2.1.8, "Add an Assign Activity for the Output Variable"
- Section 9.8.2.1.9, "Deploying with JDeveloper"
- Section 9.8.2.1.10, "Creating a DataSource in Oracle WebLogic Server Administration Console"
- Section 9.8.2.1.11, "Monitoring Using the Fusion Middleware Control Console"

9.8.2.1.1 Prerequisites

To perform this use case, you must define the following stored procedure in the SCOTT schema:

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

9.8.2.1.2 Creating an Application and an SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In the Application Navigator of JDeveloper, click **New Application**.

   The Create Generic Application - Name your application page is displayed.

2. Enter `MyHelloApp` in the **Application Name** field, and click **Next**.

   The Create Generic Application - Name your project page is displayed.

3. Enter `HelloProject` in the **Project Name** field.

4. In the Available list in the Project Technologies tab, double-click **SOA** to move it to the Selected list.

5. Click **Next**.

   The Create Generic Application - Configure SOA Settings page is displayed.

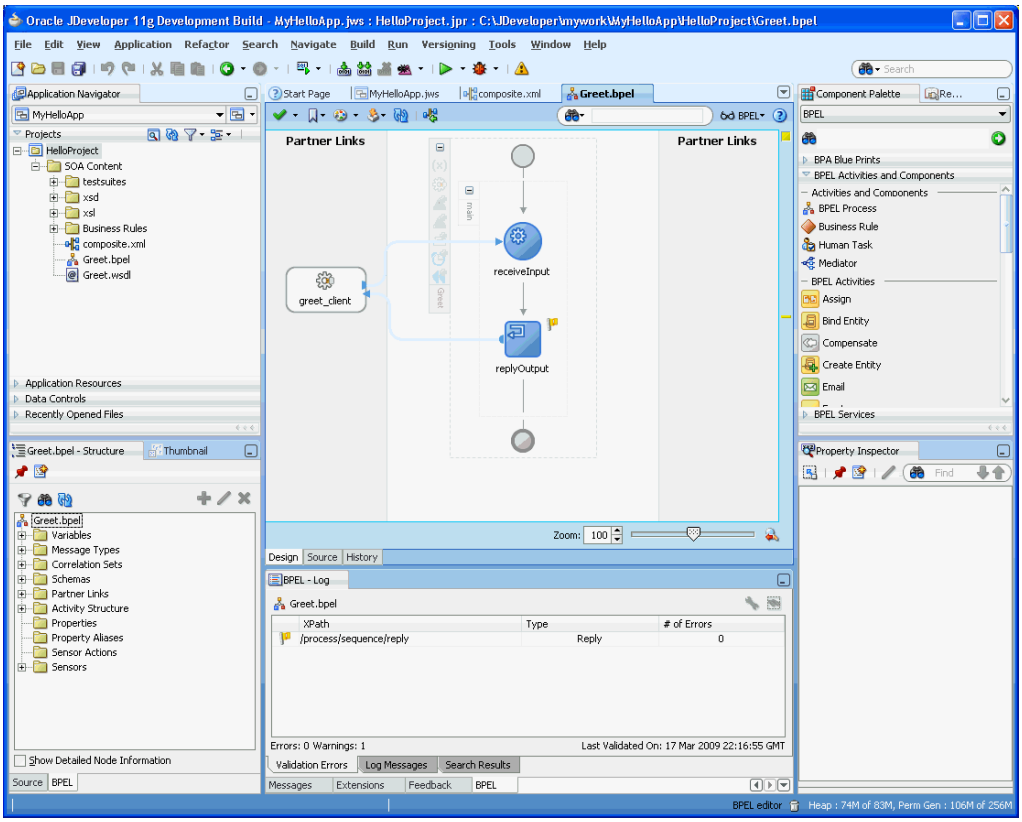
6. Select **Composite With BPEL** in the Composite Template box, and click **Finish**. The Create BPEL Process page is displayed.

7. Enter `Greet` in the **Name** field, and then select **Synchronous BPEL Process** from the Template box.

8. Click **OK**.

   The Greet BPEL process in the HelloProject of MyHelloApp is displayed in the design area, as shown in Figure 9–60.

Figure 9–60 The JDeveloper - Composite.xml



9.8.2.1.3 Creating the Outbound Oracle Database Adapter Service

Perform the following steps to create an outbound Oracle Database Adapter service:

1. Drag and drop **Database Adapter** from the Component Palette to the External References swim lane.

   The Adapter Configuration Wizard Welcome page is displayed.

2. Click **Next**.

   The Service Name page is displayed.

3. Enter `Hello` in the **Service Name** field.

4. Click **Next**.

   The Service Connection page is displayed.

Note: Ensure that you have configured the JNDI name in the `weblogic-ra.xml` file before deploying this application.

For more information, refer to Section 2.18.1, "Creating a Data Source" and Section 2.20, "Recommended Setting for Data Sources Used by Oracle JCA Adapters."

---

5. Click the **Create a New Database Connection** icon.

   The Create Database Connection dialog is displayed.

6. Enter the following details in the Create Database Connection dialog:

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

   b. Select **Oracle (JDBC)** for Connection Type.

   c. Enter the user name and password as `scott/tiger`.

   d. Enter the host name in the **Host Name** field and the JDBC port in the **JDBC Port** field.

   e. Select **SID** and enter the SID name. Alternatively, select **Service Name** and enter the service name.

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

   g. Click **OK**.

   The Connection field is populated with the MyConnection connection and the JNDI field is populated with eis/DB/MyConnection.

7. Click **Next**.

   The Operation Type page is displayed.

8. Select **Call a Stored Procedure or Function**, and then click **Next**.

   The Specify Stored Procedure page is displayed.

9. Click **Browse**. Select `HELLO` in the **Stored Procedures** pane.

   The Arguments tab displays the parameters of the stored procedure and the Source tab displays the source code of the stored procedure.

10. Click **OK**.

    The Specify Stored Procedure page is displayed. The Procedure field is populated with the HELLO stored procedure and the arguments for the HELLO stored procedure are also displayed.

11. Click **Next**.

    The Advanced Options page is displayed.

12. Specify any additional advanced options, and then click **Next**.

    The Adapter Configuration Wizard - Finish page is displayed.

13. Click **Finish**.

    The Create Partner Link dialog box is displayed. The name of the partner link is Hello, which is the same as the service name.

14. Click **OK**.

    The outbound Oracle Database Adapter is now configured and the Greet BPEL process is displayed.

9.8.2.1.4 Add an Invoke Activity

The following are the steps to add an invoke activity:

1. Drag and drop an **Invoke** activity from the Component Palette to the design area between the `receiveInput` activity and the `replyOutput` activity.

2. Double-click the **Invoke** activity.

   The Edit Invoke dialog is displayed.

3. Enter `Input` in the **Name** field.

4. Click the **Automatically Create Input Variable** icon to the right of the Input Variable field in the Invoke box.

   The Create Variable dialog is displayed.

5. Select the default variable name and click **OK**.

   The Input Variable field is populated with the default variable name. The Invoke dialog is displayed.

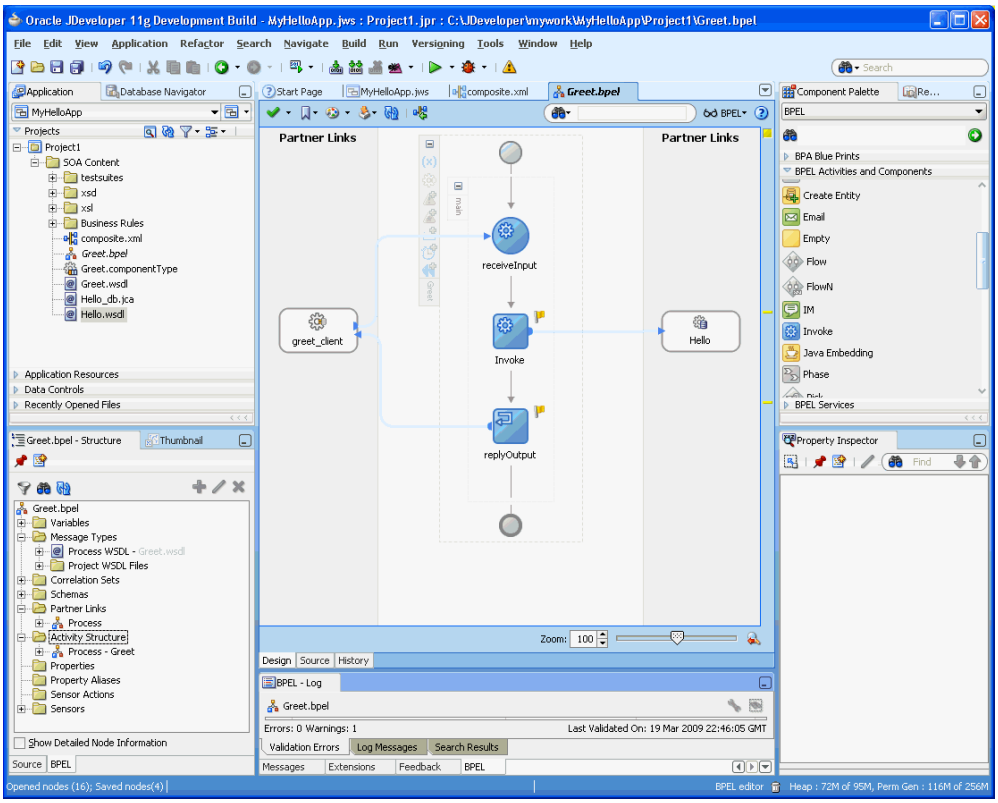
6. Repeat the same procedure to select output variable in the Output Variable field.

   In the Variables section of the Edit Invoke dialog the Input and Output variable names are displayed.

7. Click **OK**.

   A line with a right arrow is connected to the Hello partner link is displayed, as shown in Figure 9–61.

Figure 9–61 The Greet.bpel Page



9.8.2.1.5 Change the Message Part of the Request Message

When the payload of the request matches the InputParameters, then all of the IN parameters is included in the request. The only IN parameter in this example is *name*.

The following are the steps to change the message part for the `GreetRequestMessage` message:

1. In the Structure Pane for the Greet BPEL process, which is beneath the Application pane, expand **Message Types**, then **Process WSDL - Greet.wsdl**, and then **GreetRequestMessage**.
2. Select **payload**, and then click the **Edit** icon.

   The Edit Message Part - payload dialog is displayed.
3. Choose **Element** and then click the **Search** icon.

   The Type Chooser dialog is displayed.
4. Expand **Project Schema Files**, then **SCOTT_HELLO.xsd**, and select **InputParameters**.
5. Click **OK**.

   The Edit Message Part - payload dialog is displayed.
6. Click **OK**.

9.8.2.1.6 Change the Message Part of the Response Message

When the payload of the response matches the OutputParameters, then all of the OUT parameters is included in the response. The only OUT parameter in this example is *greeting*.

The steps for the `GreetResponseMessage` message part are the same as that of GreetRequestMessage with the following exceptions:

1. Expand the **GreetResponseMessage** message type, and then select **payload**.
2. Expand **SCOTT_HELLO.xsd** in the Type Chooser dialog and select **OutputParameters**.
3. Select **OutputParameters**.

9.8.2.1.7 Add a Assign Activity for the Input Variable

The following are the steps to add an Assign activity for the input variable:

1. Drag and drop an **Assign** activity from the Component Palette in between the receiveInput and Greet invoke activities in the design area.
2. Double-click the **Assign** activity.

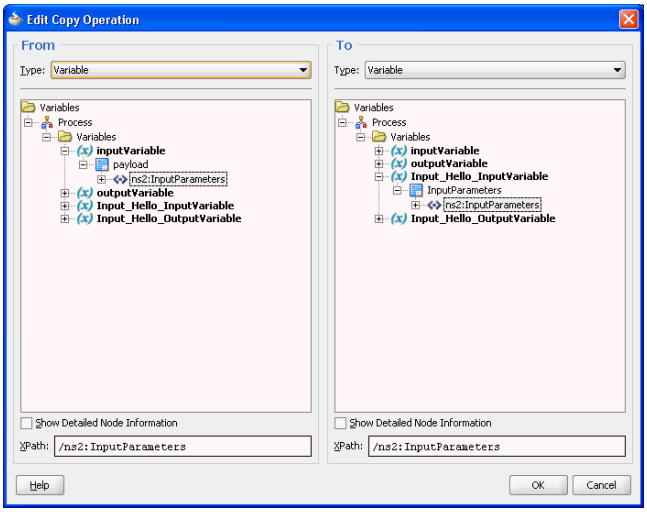
   The Assign dialog is displayed.
3. Click **General** to change the name to `NAME` in the **Name** field.
4. In the **Copy Operation** tab, click the plus icon, and select **Copy Operation** from the list of operations displayed.

   The Create Copy Operation dialog is displayed.
5. In the From area expand **Variables**, **inputVariable**, **payload**, and then select **ns2:InputParameters**.
6. In the To area expand **Variables**, **Input_Hello_InputVariable**, **InputParameters**, and then select **ns2:InputParameters**.
7. Click **OK**.

   You have assigned a value to the input parameter.

   The Assign dialog is displayed, as shown in Figure 9–62. This dialog shows the assign from the inputVariable payload to the Input_Hello_InputVariable payload.

Figure 9–62 The Create Copy Operation Dialog



8. Click **File**, **Save All**.

9.8.2.1.8 Add an Assign Activity for the Output Variable

In the second assign activity, you assign a value to the output parameter.

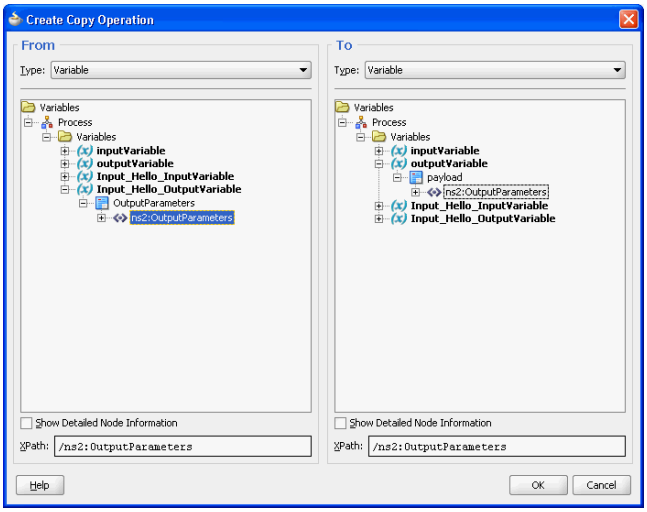
The steps for assigning a value to the output parameter are the same as assigning value to the input parameter with the following exceptions:

1. Drag and drop an **Assign** activity from the Component Palette in between the Greet invoke and replyOutput activities in the design area.
2. Double-click the **Assign** activity.

   The Assign dialog is displayed.
3. Enter `Greeting` in the **Name** field.
4. In the **Copy Operation** tab, click the plus icon, and select **Copy Operation** from the list of operations displayed.

   The Create Copy Operation dialog is displayed.
5. In the From pane expand **Input_Hello_OutputVariable**, **OutputParameters**, and then select **ns2:OutputParameters**, as shown in Figure 9–63.
6. In the To pane expand **outputVariable**, **payload**, and then select **ns2:OutputParameters**, as shown in Figure 9–63

Figure 9–63 The Create Copy Operation Dialog



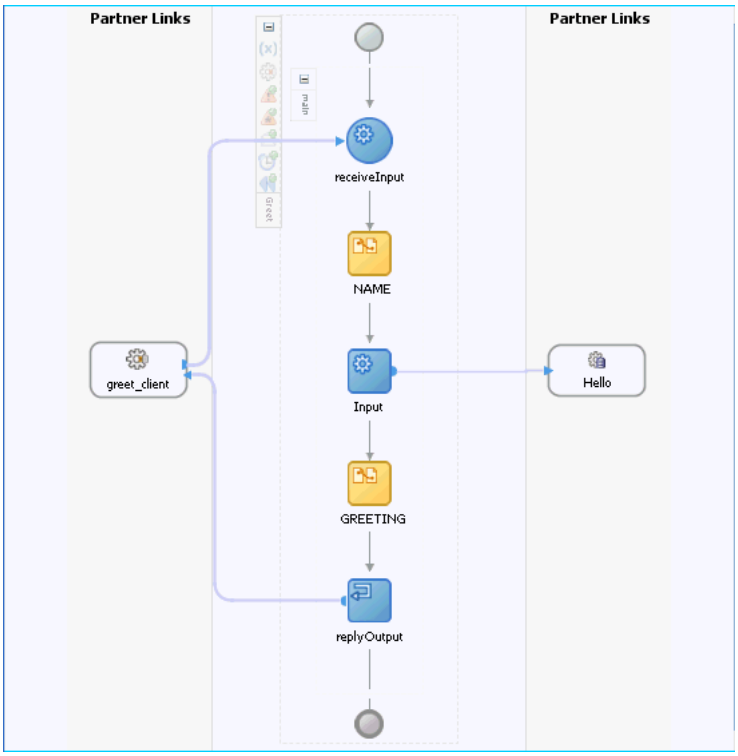
7. Click **OK**.

   You have assigned a value to the output parameter.

8. Click **File**, **Save All**.

   You have completed modeling a BPEL Process. The final BPEL process is displayed, as shown in Figure 9–64.

Figure 9–64 The Final BPEL Process Screen



9.8.2.1.9 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, use the following steps:

1. Create an application server connection using the procedure described in Chapter 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application using the procedure described in Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

9.8.2.1.10 Creating a DataSource in Oracle WebLogic Server Administration Console

Before you can test the `HelloProject` you must create a data source using the Oracle WebLogic Server Administration Console.

The following are the steps:

1. Enter `http://<hostname>:<port>/console` in your Web browser.
2. Enter a user name and password and click **Log In**.

   The administration console is displayed.
3. In the Services area under JDBC click **Data Sources**.

   A summary of JDBC Data Sources is displayed.
4. Click **New**.

   The Create a New JDBC Data Source page is displayed.
5. In the Create a New JDBC Data Source page, enter the following details:
   - `MyDataSource` in the **Name** field.
   - `jdbc/MyDataSource` in the **JNDI Name** field.
   - The Database Type is `Oracle`.
   - The Database Driver is `Oracle's Driver (Thin XA) for Instance Connections; Versions 9.0.1, 9.2.0, 10, 11`.
6. Click **Next**.

   A message stating that no other transaction configuration options are available is displayed.
7. Click **Next**.

   The Create a New Data Source page is displayed.
8. Enter the following details:.
   - **Database Name**: This is usually the `SID`.
   - **Host Name**: Enter the name of the host computer.
   - **Port Number**: Enter the port number.

     The default port is `1521`.
   - **Database User Name**: Enter `SCOTT`
   - **Password**: Enter `TIGER`.
   - **Confirm Password**: Enter `TIGER`.
9. Click **Next**.

A summary of the data source configuration is displayed.

10. Click **Test Configuration**.

    The Messages area indicates that the connection test succeeded.

11. Click **Next**. Select **AdminServer** as the target by selecting the check box.

12. Click **Finish**.

    The summary of JDBC Data Sources now includes the MyDataSource data source that you created in the preceding steps.

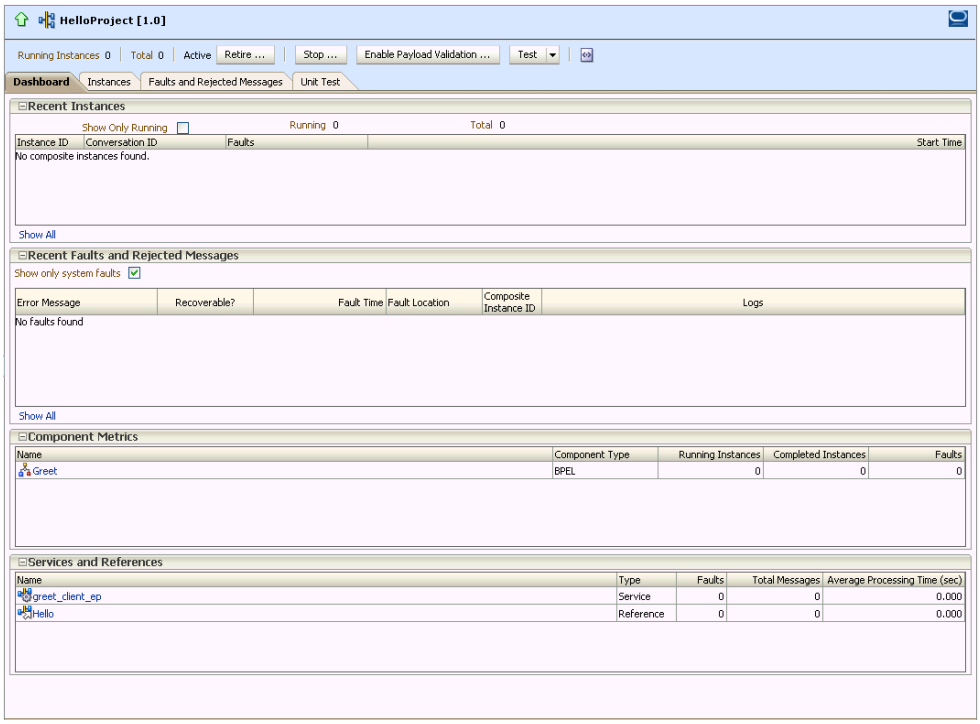
9.8.2.1.11 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. A list of SOA composites is displayed, including the HelloProject[1.0] that you created in the preceding steps.

2. Click the **HelloProject[1.0]** link. The **Dashboard** tab is displayed, as shown in Figure 9–65.

Figure 9–65 The Dashboard Tab of the HelloProject[1.0] Project



3. Click Test. A new browser window is displayed.

4. Enter your name in the **NAME** field that is marked `xsd:string` and then click **Invoke**.

   The browser window displays the Test Result.

5. To view the XML file in readable form, click **Formatted XML**. Figure 9–66 shows the formatted XML file.

Figure 9–66 The Formatted XML File

```
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <env:Header>
  <wsa:ReplyTo>
   <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
  </wsa:ReplyTo>
 </env:Header>
 <env:Body>
  <OutputParameters
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.oracle.com/pcbpel/adapter/db/SCOTT/HELLO/">
   <GREETING
     xmlns="http://xmlns.oracle.com/pcbpel/adapter/db/SCOTT/HELLO/">Hello Michael</GREETING>
  </OutputParameters>
 </env:Body>
</env:Envelope>
```

9.8.2.2 File To StoredProcedure Use Case

This use case illustrates the execution of an Oracle stored procedure. The input to the stored procedure is obtained by reading a file using the File Adapter. The stored procedure executes, populating a table with the data from its input parameter.

To obtain the `adapter-db-101-file2storedprocedure` use case, access the Oracle SOA Sample Code site.

This use case includes the following topics:

- Section 9.8.2.2.1, "Prerequisites"
- Section 9.8.2.2.2, "Creating an Application and an SOA Project"
- Section 9.8.2.2.3, "Creating the Outbound Oracle Database Adapter Service"
- Section 9.8.2.2.4, "Creating an Invoke Activity"
- Section 9.8.2.2.5, "Creating the Inbound File Adapter Service"
- Section 9.8.2.2.6, "Adding a Receive Activity"
- Section 9.8.2.2.7, "Adding an Assign Activity"
- Section 9.8.2.2.8, "Wiring Services and Activities"
- Section 9.8.2.2.9, "Deploying with JDeveloper"
- Section 9.8.2.2.10, "Creating a Data Source"
- Section 9.8.2.2.11, "Adding a Connection-Instance"
- Section 9.8.2.2.12, "Testing using the File Adapter Service and SQL*Plus"
- Section 9.8.2.2.13, "Monitoring Using the Fusion Middleware Control Console"

9.8.2.2.1 Prerequisites

To perform the file to stored procedure use case, the following schema objects and stored procedure must be defined in the `SCOTT/TIGER` schema before modeling the BPEL Composite using JDeveloper.

```
create type address as object
(
  street varchar2(20),
  city   varchar2(15),
  state  char(2),
  zip    char(5)
);
create type customer as object
```

```
(
  fname varchar2(10),
  lname varchar2(10),
  loc   address,
  email varchar2(25),
  phone varchar2(15)
);
create type collection as table of customer;
create table customers
(
  name  varchar2(25),
  loc   varchar2(45),
  email varchar2(25),
  phone varchar2(15)
);
create procedure add_customers(c in collection) as
begin
  for i in c.first .. c.last loop
    insert into customers values (
      c(i).lname || ', ' || c(i).fname,
      c(i).loc.street || ', ' || c(i).loc.city || ', ' || c(i).loc.state || ' ' ||
 c(i).loc.zip,
      c(i).email,
      c(i).phone    );
  end loop;
end;
```

You can define these schema objects and stored procedure using the `adapters-db-101-file2storedprocedure/artifacts/sql/customers.sql` file from the `adapters-db-101-file2storedprocedure` sample by accessing the Oracle SOA Sample Code site.

9.8.2.2.2 Creating an Application and an SOA Project

You must create a JDeveloper application to contain the SOA composite. Use the following steps to create a new application, an SOA project:

1. Open JDeveloper.
2. In the Application Navigator, click **New Application**. The Create Generic Application - Name your Application page is displayed.
3. Enter `File2SPApp` in the **Application Name** field.
4. In the Application Template list, select **Generic Application**.
5. Click **Next**.

   The Create Generic Application - Name your project page is displayed.
6. In the **Project Name** field, enter a descriptive name. For example, `File2SPProject`.
7. In the **Available list** in the **Project Technologies** tab, double-click **SOA** to move it to the Selected list.
8. Click **Next**. The Create Generic Application - Configure SOA Settings page is displayed.
9. Select **Composite With BPEL** from the Composite Template list, and then click **Finish**.

You have created a new application, and an SOA project. This automatically creates an SOA composite.

The Create BPEL Process page is displayed.

10. Enter a name for the BPEL process in the **Name** field. For example, `File2SP`.
11. Select **Define Service Later** in the Template list, and then click **OK**.

    The `File2SP` BPEL process in the `File2SPProject` of `File2SPApp` is displayed in the design area.

9.8.2.2.3 Creating the Outbound Oracle Database Adapter Service

Perform the following steps to create an outbound Oracle Database Adapter service:

1. Drag and drop **Database Adapter** from the Service Adapters list to the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `File2SPService` in the **Service Name** field.
4. Click **Next**.

   The Service Connection page is displayed.
5. Click the **Create a New Database Connection** icon.

   The Create Database Connection dialog is displayed.
6. Enter the following details in the Create Database Connection dialog:
   a. Enter a connection name in the **Connection Name** field. For example, `MyConnection`.
   b. Select **Oracle (JDBC)** for Connection Type.
   c. Enter the user name and password as `scott/tiger`.
   d. Enter the host name in the **Host Name** field and the JDBC port in the **JDBC Port** field.
   e. Select **SID** and enter the SID name. Alternatively, select **Service Name** and enter the service name.
   f. Click **Test Connection**. A success message is displayed in the Status pane.
   g. Click **OK**.

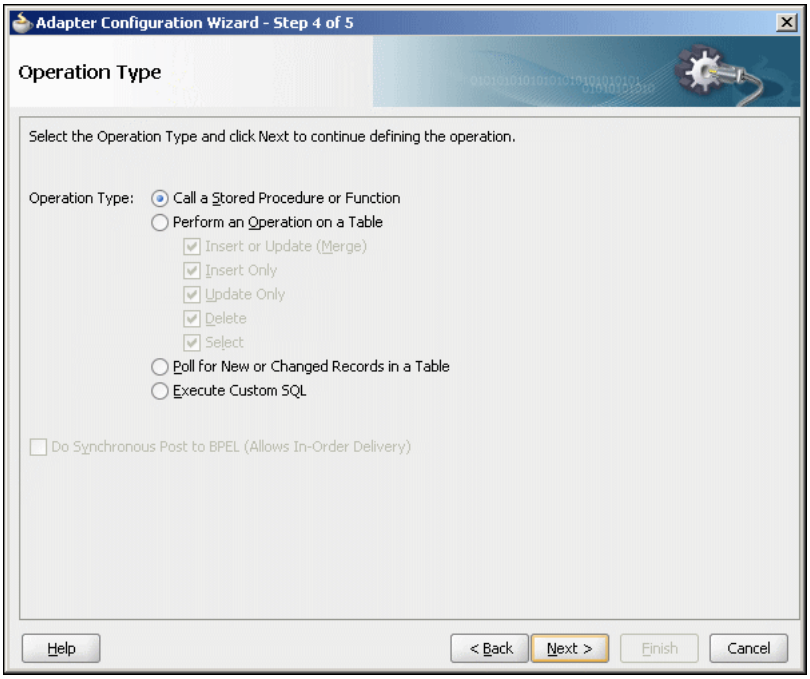
   The Connection field is populated with the MyConnection connection and the JNDI field is populated with eis/DB/MyConnection.
7. Click **Next**.

   The Adapter Interface page is displayed.
8. In the Adapter Interface page, select **Define from operation and schema (specified later)**, and the click **Next**.

   The Operation Type page is displayed.
9. Select **Call a Stored Procedure or Function**, as shown in Figure 9–67, and click **Next**.

   The Specify Stored Procedure page is displayed.

Figure 9–67 The Adapter Configuration Wizard - Operation Type Page



10. Click **Browse**. Select `ADD_CUSTOMERS` in the **Stored Procedures** pane.

    The Arguments tab displays the parameters of the stored procedure and the Source tab displays the source code of the stored procedure.

11. Click **OK**.

    The Specify Stored Procedure page is displayed.

    The Procedure field is populated with the `ADD_CUSTOMERS` stored procedure and the arguments for the `ADD_CUSTOMERS` stored procedure are also displayed.

12. Click **Next**.

    The Advanced Options page is displayed.

13. Specify any additional options, and then click **Next**.

    The Finish page is displayed.

14. Click **Finish**.

    The Create Partner Link dialog is displayed.

    The name of the partner link is File2SPService, which is the same as the service name.

15. Click **OK**.

    The outbound Oracle Database Adapter is now configured and the File2SP BPEL process is displayed.

9.8.2.2.4 Creating an Invoke Activity

You must complete the BPEL process by creating an Invoke activity. This creates the input variables.

The following are the steps to create an Invoke activity:

1. Click **File**, **Save All**.

2. Drag and drop an **Invoke** activity from the Component Palette to the design area.
3. Drag the right arrow on the right of the Invoke activity and connect it to the `File2SPService` partner link.

   The Edit Invoke dialog is displayed.
4. Enter `Invoke` in the **Name** field.
5. Click the **Automatically Create Input Variable** icon to the right of the Input Variable field in the Invoke dialog.

   The Create Variable dialog is displayed.
6. Select the default variable name and click **OK**.

   The Input variable name is displayed in the Variables area of the Edit Invoke dialog.
7. Click **OK**.

   A line with a right arrow connecting to the is `File2SPService` partner link is displayed.

9.8.2.2.5 Creating the Inbound File Adapter Service

Perform the following steps to create an inbound File adapter service. This creates the service that reads input XML from a file directory:

1. Drag and drop the **File Adapter** from the Component Palette to the External References swim lane.

   The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `ReadCustomers` in the **Service Name** field.
4. Click **Next**.

   The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and then click **Next**. The Operation page is displayed.
6. Select **Read File** as the Operation Type and **Read** as the Operation Name. Do not select the other check boxes.
7. Click **Next**.

   The File Directories page is displayed.
8. Select **Physical Path**, and enter a physical path in the **Directory for Incoming Files** field.
9. Select **Process files recursively** and **Delete files after successful delivery**, and the click **Next**.

   The File Filtering page is displayed.
10. Specify **File Wildcards**, enter `customers.xml` in the **Include Files with Name Pattern** field, and then click **Next**.

    The File Polling page is displayed.
11. Specify any value in the **Polling Frequency** field, and click **Next**.

    The Message page is displayed.

12. Click **Browse For Schema File** that is displayed at the end of the URL field.

    The Type Chooser dialog is displayed.

13. Click **Project Schema Files**, **SCOTT_ADD_CUSTOMERS.xsd**, and **InputParameters**.

14. Click **OK**.

    The Messages page is displayed again. The URL is `xsd/SCOTT_ADD_CUSTOMERS.xsd`, and the Schema Element is `InputParameters`.

15. Click **Next**.

    The Finish page is displayed.

16. Click **Finish**.

    This terminates the inbound File Adapter service.

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

18. Click **File**, **Save All**.

9.8.2.2.6 Adding a Receive Activity

The File Adapter Service provides input to the Receive Activity, which then initiates the rest of the BPEL Process.

The following are the steps to add a Receive activity:

1. Double-click **File2SP**. The File2SP.bpel page is displayed.
2. Drag and drop a **Receive** activity from the Component Palette to the design area.
3. Drag the left arrow on the left of the Receive activity and connect it to the `ReadCustomers` partner link.

   The Edit Receive dialog is displayed.

4. Enter `Receive` in the **Name** field.
5. Click the **Automatically Create Input Variable** icon to the right of the Variable field in the Edit Receive dialog.

   The Create Variable dialog is displayed.

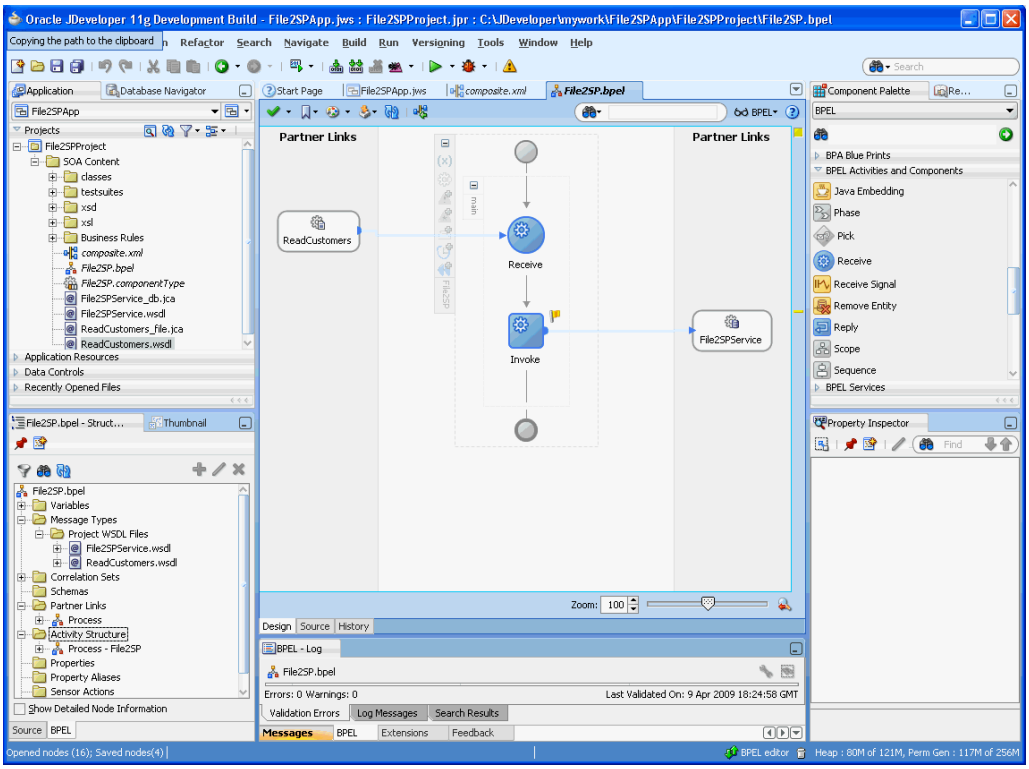
6. Select the default variable name and click **OK**.

   The Variable field is populated with the default variable name.

7. Select **Create Instance**, and click **OK**. The JDeveloper File2SP.bpel page is displayed.

   After adding the Receive activity, the JDeveloper window appears, as shown in Figure 9–68.

Figure 9–68 Adding a Receive Activity



This figure shows the File2SP.bpel page in which a Receive activity is added. On the left side of the File2SP.bpel page is the `ReadCustomers` partnerlink and on the right is the File2SPService partner link.

****

8. Click **File**, **Save All**.

9.8.2.2.7 Adding an Assign Activity

Next, you must assign a value to the input parameter.

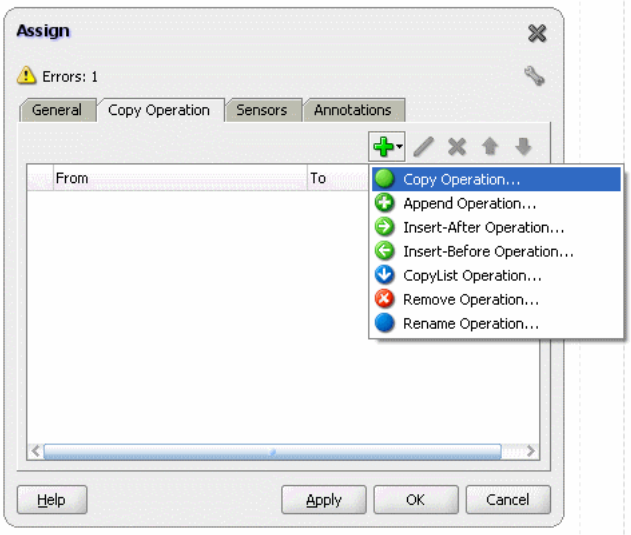
The following are the steps to add an Assign activity:

1. Drag and drop an **Assign** activity from the Component Palette in between the Receive and Invoke activities in the design area.
2. Double-click the **Assign** activity.

   The Assign dialog is displayed.
3. Click **General**, and then `CUSTOMER` in the **Name** field.
4. Click the **Copy Operation** tab.

   The Assign dialog is displayed, as shown in Figure 9–69.

Figure 9–69 The Assign Dialog - Copy Operation Tab



5. Click the icon with the plus sign, as shown in Figure 9–69, and then select **Copy Operation**.

   The Create Copy Operation dialog is displayed.

6. In the **From** area expand **Process**, **Variables**, **Receive_Read_InputVariable** and then body.

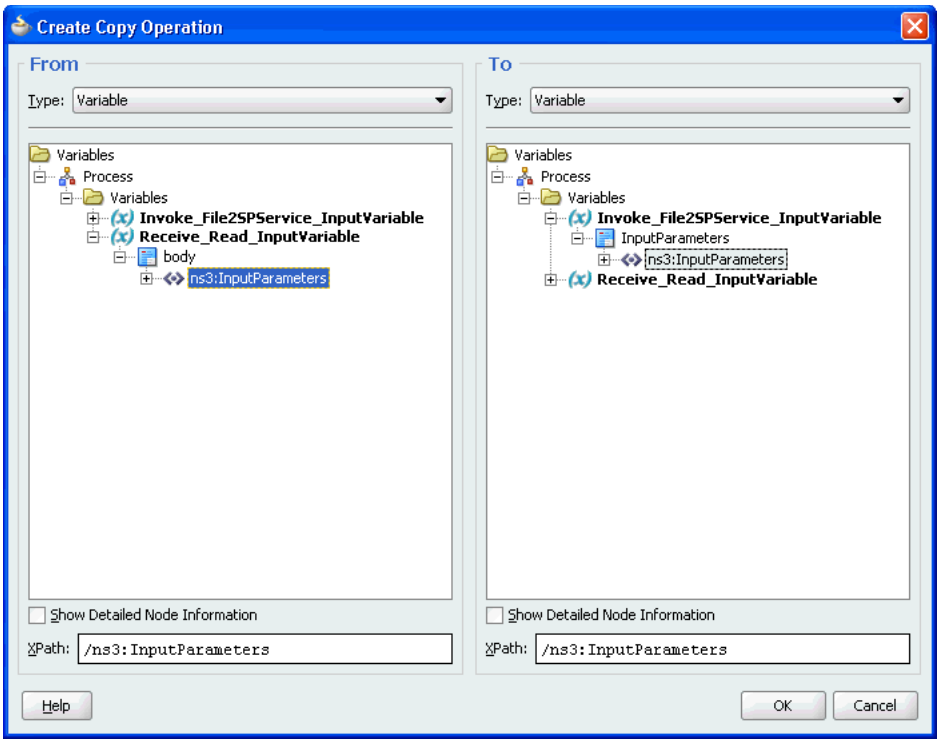
7. Select **ns3:InputParameters**.

8. In the **To** area expand **Process**, **Variables**, **Invoke_File2SPService_InputVariable**, and then **InputParameters**.

9. Select **ns3:InputParameters**.

10. Click **OK**. The Assign dialog is displayed, as shown in Figure 9–70.

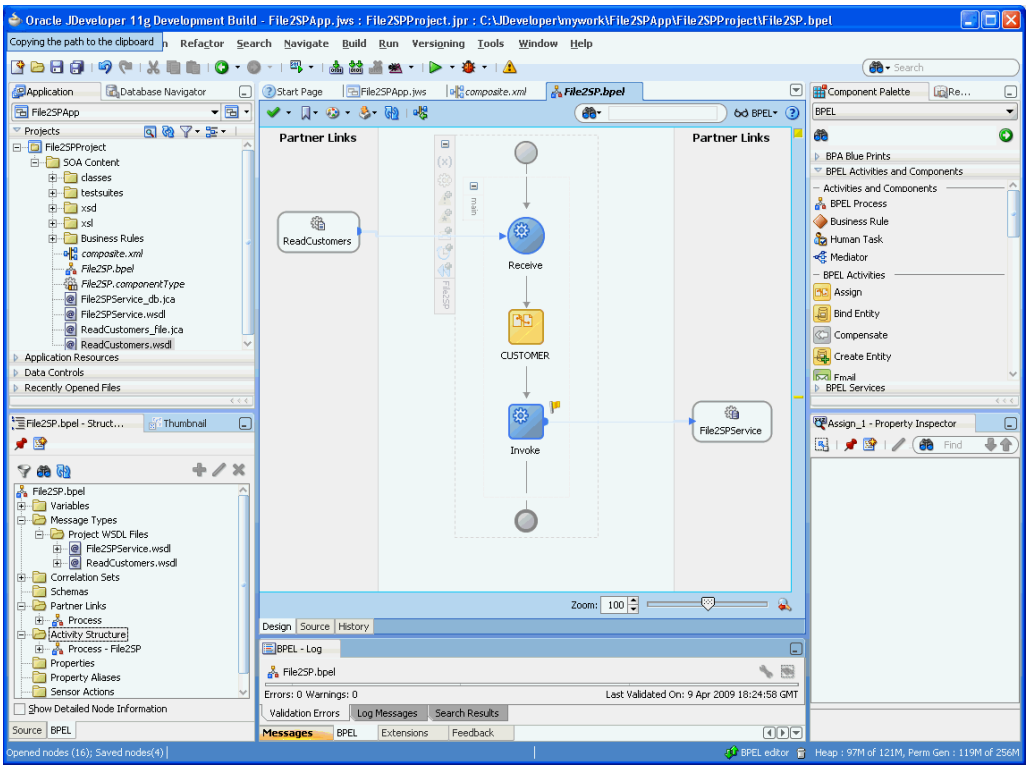
Figure 9–70 The Assign Dialog



11. Click **OK**.

The JDeveloper File2SP.bpel page is displayed, as shown in Figure 9–71.

Figure 9–71 The JDeveloper - File2SP.bpel



12. Click **File**, **Save All**.

9.8.2.2.8 Wiring Services and Activities

You must assemble or wire the three components that you have created: Inbound adapter service, BPEL process, Outbound adapter reference. Perform the following steps to wire components:

1. Drag the small triangle in `ReadCustomer` in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in `File2SPService` in the External References area.
3. Click **File**, **Save All**.

9.8.2.2.9 Deploying with JDeveloper

You must deploy the application profile for the SOA project and the application you created in the preceding steps. To deploy the application profile using JDeveloper, use the following steps:

1. Create an application server connection using the procedure described in Chapter 2.6, "Creating an Application Server Connection for Oracle JCA Adapters."
2. Deploy the application using the procedure described in Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper."

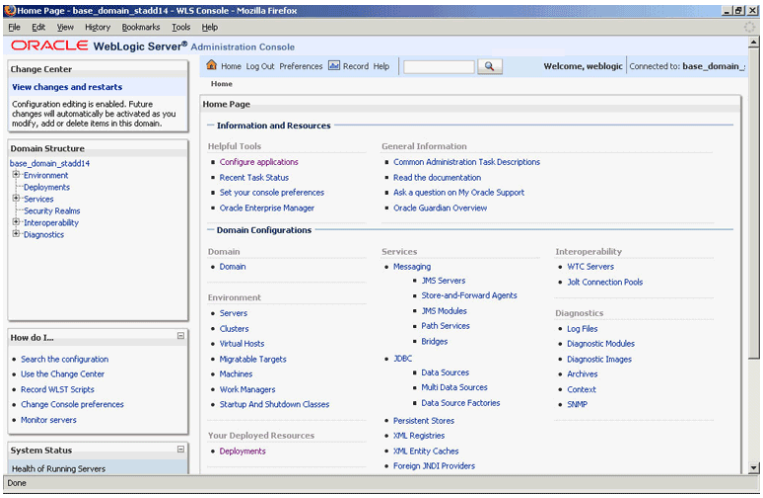
9.8.2.2.10 Creating a Data Source

Before you can test the `File2SPProject` you must create a data source using the Oracle WebLogic Server Administration Console, by using the following steps:

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

   The Home page of the Oracle WebLogic Server Administration Console is displayed, as shown in Figure 9–72.

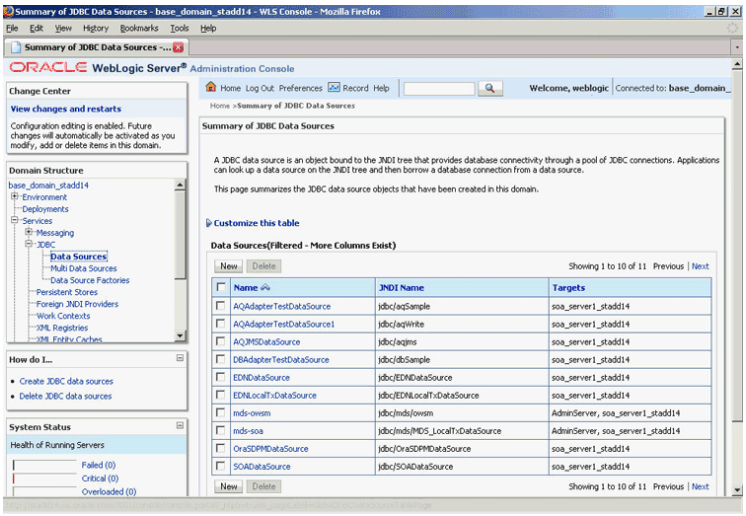
Figure 9–72 Oracle WebLogic Server Administration Console Home Page



3. Under Domain Structure, select **Services**, **JBDC**, and then click **DataSources**.

   The Summary of JDBC Data Sources page is displayed, as shown Figure 9–73.

Figure 9–73 The Summary of JDBC Data Sources Page



4. Click **New**.

   The Create a New JDBC Data Source page is displayed.

5. Enter the following values for the properties to be used to identify your new JDBC data source:

   - Enter `MyDataSource` in the **Name** field.
   - Enter `jdbc/MyDataSource` in the **JNDI Name** field.
   - Retain the default value `Oracle` for **Database Type**.
   - Retain the default value `Oracle's Driver (Thin XA) for Instance Connections; Versions 9.0.1, 9.2.0, 10, 11` for **Database Driver**.

6. Click **Next**.

The Create a New JDBC Data Source - Transaction Options page is displayed. A message stating, "No other transaction configuration options are available." is displayed.

7. Click **Next**.

   The Create a New JDBC Data Source - Connection Properties page is displayed.

8. Enter the following connection properties in the Connection Properties page:
   - Enter a name in the **Database Name** field, which is usually the SID.
   - Enter the host name in the **Host Name** field.
   - Enter the port number in the **Port** field.
   - Enter `SCOTT` in the **Database User Name** field.
   - Enter `TIGER` in the **Password** field.
   - Enter `TIGER` in the **Confirm Password** field.
9. Click **Next**. The Create a New JDBC Data Source - Test Database Connection page is displayed.
10. Click **Test Configuration** to test the database availability and the connection properties you provided. A message stating that the, "Connection test succeeded" is displayed at the top of the Create a New JDBC Data Source - Test Database Connection page.
11. Click **Next**.

    The Create a New JDBC Data Source - Select Targets page is displayed.

12. Select `AdminServer` as target, and then click **Finish**.

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

9.8.2.2.11 Adding a Connection-Instance

The database adapter needs an instance entry, which points to a data source.

The following are the steps to add a connection instance:

1. Search beahome/ for DbAdapter.rar.
2. Unzip the file.
3. Edit `META-INF/weblogic-ra.xml` (and possibly `ra.xml`.), as shown in the following example:

```
<connection-instance>
               <jndi-name>eis/DB/MyConnection</jndi-name>
               <connection-properties>
                   <properties>
                       <property>
                           <name>xADataSourceName</name>
                           <value>jdbc/MyDataSource</value>
                       </property>
                   </properties>
               </connection-properties>
           </connection-instance>
```

4. Use the same database connection name, **MyConnection**, for the JNDI name.

5. Use the same data source name, **MyDataSource**, as the `xADataSourceName`.
6. Jar the file again.
7. Restart the application server.

   You can also create a new database adapter instance using the Oracle WebLogic Server Administration Console.

9.8.2.2.12 Testing using the File Adapter Service and SQL*Plus

You must test the BPEL process by providing input file for the File Adapter. The results of the BPEL process are seen using a simple query from a table. The `customers.xml` file contains the following input:

```
<InputParameters xmlns="http://xmlns.oracle.com/pcbpel/adapter/db/SCOTT/ADD_
CUSTOMERS/">
  <C>
    <C_ITEM>
      <FNAME>John</FNAME>
      <LNAME>Doe</LNAME>
      <LOC>
        <STREET>123 Main Street</STREET>
        <CITY>Anytown</CITY>
        <STATE>CA</STATE>
        <ZIP>12345</ZIP>
      </LOC>
      <EMAIL>john.smith@gmail.com</EMAIL>
      <PHONE>567-123-9876</PHONE>
    </C_ITEM>
    <C_ITEM>
      <FNAME>Jane</FNAME>
      <LNAME>Doe</LNAME>
      <LOC>
        <STREET>987 Sunset Blvd</STREET>
        <CITY>Sometown</CITY>
        <STATE>CA</STATE>
        <ZIP>34567</ZIP>
      </LOC>
      <EMAIL>JaneDoe@yahoo.com</EMAIL>
      <PHONE>567-123-9876</PHONE>
    </C_ITEM>
  </C>
</InputParameters>
```

The following are the steps for testing the BPEL process you created:

1. Place a copy of `customers.xml` in the input directory that you specified when you created the inbound File Adapter Service.
2. The Oracle File Adapter polls the directory for new files. The Receive activity initiates the BPEL process once the file is received by the File Adapter Service.
3. The data for all of the customers is assigned to the InputParameters of the stored procedure.
4. The stored procedure executes. It transforms the data for each customer and then inserts the customer data into a table.
5. Query the table to see the following results:

   ```
   SQL> select * from customers;
   ```

```
NAME                      LOC
------------------------- ---------------------------------------------
EMAIL                     PHONE
------------------------- ---------------
Doe, John                 123 Main Street, Anytown, CA 12345
john.smith@gmail.com      567-123-9876

Doe, Jane                 987 Sunset Blvd, Sometown, CA 34567
JaneDoe@yahoo.com         567-123-9876
```

9.8.2.2.13 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed EM Composite using the Fusion Middleware Control Console. Perform the following steps:

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

   A list of SOA composites is displayed, including `File2SPProject[1.0]` that you created in the preceding steps.

2. Click **File2SPProject[1.0]**.

   The Dashboard is displayed. Note your Instance ID in the Recent Instances area.

3. Click the **Instances** tab.

   A Search dialog is displayed. The default search displays all instances by their Instance ID.

4. Select the **Instance ID** that you noted above.

   A new window opens. It lists any faults (No faults found) and enables you to view the Audit Trail of your instance. Your instance trace is displayed in a new window.

5. The instance tree is expanded from **ReadCustomers** (service) to **File2SP** (BPEL Component) to **File2SPService** (reference).

6. Click **File2SP** BPEL Component.

   The Audit Trail of your process is displayed.

7. Expand the <payload> node to see the input provided to the stored procedure, as shown in Figure 9–74.

Figure 9–74 The Audit Trail Tab

8. Additionally, click the **Flow** tab to view the process flow, as shown in Figure 9–75.

Figure 9–75 Viewing the Process Flow

9.8.3 Database Adapter/Coherence Integration

There is a performance improvement when the Database Adapter is used with Coherence Cache on an Exalogic system. The feature that provides this improvement is called Database Adapter/Coherence Integration.

There are two specific use cases where there is an advantage to using the Database Adapter with Coherence Cache on an Exalogic system. Specifically, performance can be improved when performing the following operations:

- Insert/Update to a database
- Select by primary key

9.8.3.1 Inserts/Updates to a Database

Inserts and updates to a database using the Database Adapter and Coherence cache are improved through the internal use of an *intermediary* Coherence data source, called a Coherence Cache, basically an in-memory database.

In the typical case, you perform insert/delete/update operations directly on the database. To improve performance, these operations can first be performed on this Coherence-fronting in-memory database, called a *write-behind map*, which enables read-write operations using the Cache.

Using such a Coherence map improves the latency of BPEL/OSB processes performing insert/delete/update operations, as these processes can return immediately to the caller without a trip to the database; the actual and intensive work of updating the database is done instead by the Coherence Cache intermediary.

You can use Coherence in this manner when processing large batches of records, which makes record updating more convenient and efficient.

Note: Database Adapter use cases **that do not** leverage Coherence Cache include the following operations: inbound polling, pure SQL invokes, stored procedure calls, and general Selects that return multiple rows.

9.8.3.1.1 Select Optimization

The second use case that Database Adapter/Coherence Integration improves is query performance, specifically in optimizing Select statement use cases. Database Adapter/Coherence Integration provides benefits to the query performance by caching data that might be accessed frequently by many different process instances.

When select optimization is used, to optimize queries, the Database Adapter/Coherence Integration uses a *read-only* Coherence Cache (also called an L2-read cache), which the Database Adapter checks first for a cache hit before proceeding to the database. In other words, queries are optimized by checking to see if the data being queried against is in the Coherence Cache first; if not found there, the database is checked for the same data.

When a Coherence miss occurs, the data is read from the database and loaded into the Coherence Cache. The presumption is that checking the Coherence Cache is faster than executing a query on the database, as the ratio of cache visits to cache misses is typically high.

9.8.3.1.2 Queries that Do Not Benefit from Coherence Database Adapter Integration

Not all queries can benefit from cache visits and hence Coherence Database Adapter Integration.There is no indication if there was a Coherence cache hit on all records that meet a specified query criterion, or if there are additional database records that could have been hit but which were not in cache.

For this reason, the query optimization feature includes a new kind of Database Adapter operation, which is a *Select by Primary Key.* Unlike the existing *Select* and *queryByExample* operations, when using *Select by Primary Key* you can only return a single row. With the primary key selected to return a single row, you are in effect requesting more specific records to be returned from the Coherence Cache, thus improving the performance of the feature against the Cache.

9.8.3.2 Database Adapter/Coherence Integration Architecture

You can choose whether to use Database Adapter/Coherence Integration by making a simple choice among `none`, `read`, or `read-write` in the Operation Type screen of the Database Adapter Wizard. However, it is useful to know some of the background related to the architecture of the Database Adapter/Coherence Integration, as detailed in the following subsections.

For more information on Eclipselink, see http://www.oracle.com/technetwork/middleware/toplink/documentation/index.html

Some background on Coherence can be found at http://www.oracle.com/technetwork/middleware/coherence/overview/index.html

9.8.3.2.1 Current Design of the Database Adapter (No Coherence Cache)

With the current design of the Database Adapter, the Adapter performs selects and inserts to the EclipseLink layer, which directly communicates with the data sourcees without the Coherence Cache.

When you choose `none` in the Cache Usage dropdown on the Operations Type screen in the Database Configuration Wizard, you indicate you do not want to use cache.

9.8.3.2.2 Read-Write Coherence Cache Database Adapter Integration

You can choose to use read-write cache by choosing `read-write` from the Cache-Usage dropdown of the Operation Type screen of the Database Adapter Wizard

Eclipselink is in two layers, with Coherence Cache (a Coherence Cache Store) between the two layers. There is actually only one Eclipselink project, but two copies of that project.

- The top copy of Eclipselink redirects all inserts/selects from the data store to Coherence Cache.
- The bottom copy of Eclipselink handles requests by Coherence Cache to load a particular record by ID from the database, or to store a particular record to the database.

A Select you execute in the read-write scenario might not uniquely identify the rows to retrieve.

Such a case could be a `SELECT *` or `SELECT where total gross > ?`

The write-behind Coherence Cache can only receive requests to load a record by ID. Thus, in either of these cases, if all queries were directed to Coherence Cache, no results would be returned. In this case, the query proceeds to the data source directly, and then the Coherence Cache is updated.

9.8.3.2.3 Read Coherence Cache Database Adapter Integration

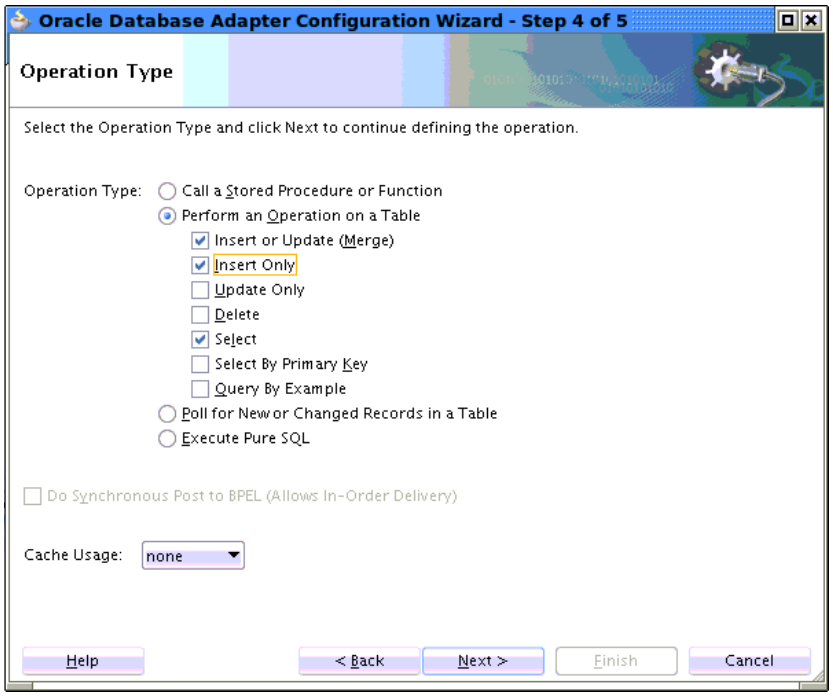
You can choose to use read cache by choosing `read` from the Cache Usage dropdown on the Operation Type screen of the Database Adapter Wizard.

With Read Cache, when the Database Adapter inserts a record to the Database or selects a record from the database, the Coherence Cache is updated. Any query that identifies a row (that is, by specifying primary key) first checks the Coherence cache, possibly saving a trip to the database.) As the Coherence Cache is distributed and can be simply thought of as a hash map, selecting by a specific primary key enables faster lookups through the Coherence Cache Map.

9.8.3.2.4 Enabling No Cache Using the Operations Type Screen

Figure 9–76 shows the No Caching option as it appears on the Operations type screen of the Database Adapter Wizard, with `none` selected.

Figure 9–76 The Database Adapter Configuration Wizzrd Operation Type Screen, with No Caching Selected



All outbound operations are enabled on this screen with the `none` option selected. Once you select this option, and choose `Next` or `Finish`, none of the selected operations contain the property `CacheUsage`. This absence of a property is equivalent to the JCA activation property `CacheUsage` being equal to the value `none`.

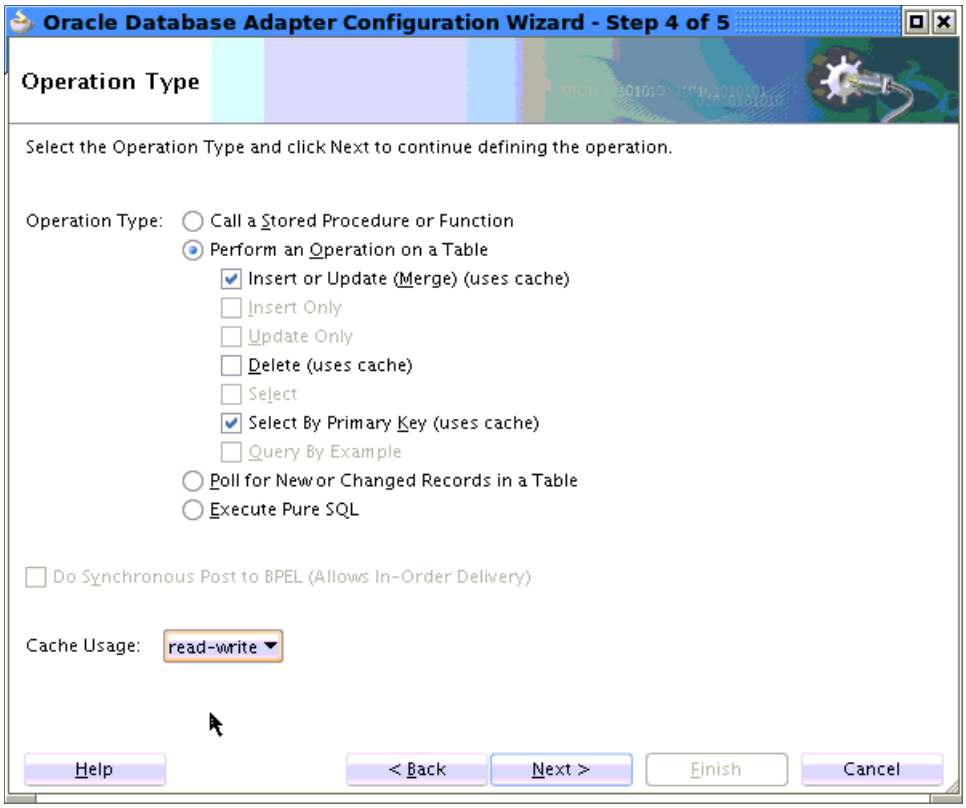
The following options are the only operations pre-selected when you choose the `none` option as the cache usage:

- `Merge`
- `Insert Only`
- `Select`

9.8.3.2.5 Enabling Read-Write Caching Using the Operation Type Screen

You can choose to enable read-write caching through the Operation Type screen. See Figure 9–77. Once you select this option and press `Next` or `Finish`, the JCA property `CacheUsage` value is set to `read-write`.

Figure 9–77 Enabling Read-Write Caching Using the Operation Type Screen



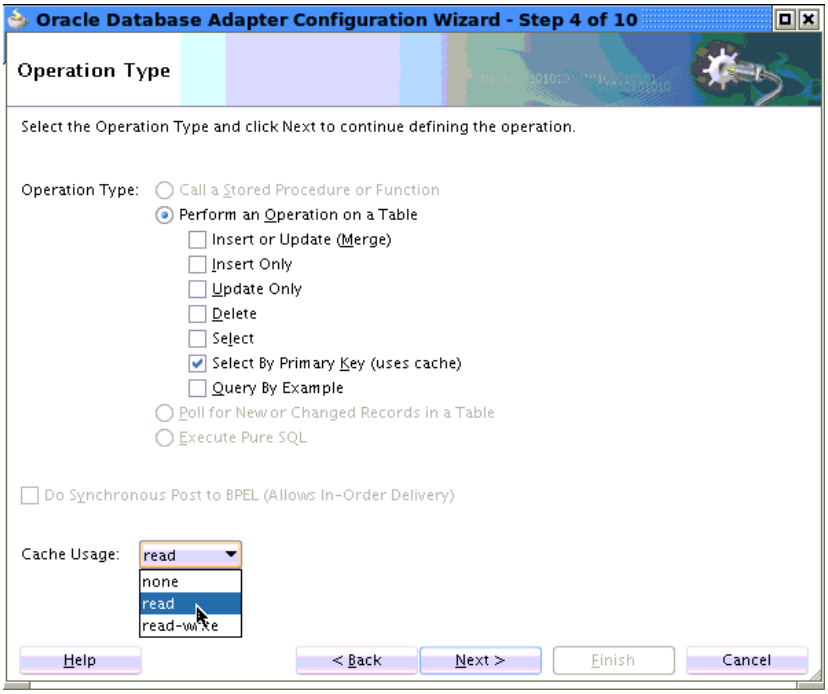
Refer to the following list of operations to understand how they are used on this screen when you choose the `read-write` option from the `Cache Usage` dropdown:

- `Insert or Update (Merge)` are enabled and have the string `uses cache` appended to their label.
- `Insert only` is disabled, as the underlying cache store always performs a merge.
- `Update Only` is disabled, as the underlying cache store always performs a merge.
- `Delete` is selectable but not pre-selected and has the string `uses cache` appended to it when it is selected.
- `Select` is disabled, as this query is converted into a Coherence filter executed on a Coherence map.
- `Query by Example` is diabled, as the Database Adapter/Coherence Integration query is converted into a Coherence filter executed on a Coherence map.
- `Select by Primary Key` has the string `uses cache` appended to the label.

9.8.3.2.6 Enabling Read Caching Using the Operation Type Screen

You can enable read caching using the `Cache Usage` option on the Operation Type screen. See Figure 9–78. Once you select this option on the screen and press `Next` or `Finish`, the JCA property `CacheUsage` value is set to `read`.

Figure 9–78 Enabling Read-Write Caching Using the Operation Type Screen



For the read cache option, only the `Select by Primary Key` operation is pre-selected. `Select by Primary Key` is the only operation that can be meaningfully executed by the Coherence Database Adapter Integration feature through the cache, although other operations can update cache. Because read cache is not intrusive on the cache, any of the operations on the Operation Type on this screen are disabled.

`Select` and `Query By Example` are not disabled, although they do not directly update the cache. The Database Adapter/Coherence Integration feature executes the `Select` against the database, but updates the Coherence Cache with any rows that are returned.

The general operation of read caching is that if any objects returned exist in the Coherence Cache, the *objects in the cache are returned*, rather than the Database Adapter/Coherence Integration feature building a *new copy* from the result set.

This operation improves performance where the master database record has several details; a query on the details does not have to be executed again.

A query behaves the same as a `Select`. This is true, for example with XML data where the primary key is set (and it does not get a cache hit).

9.8.3.2.7 XA Transactions, Read-Write and Read Operations with Coherence/Database Adapter Integration

When using Database Adapter/Coherence Integration, you cannot use **XA transactions with read-write operations.** This is because the Database Adapter, with

Coherence Integration, performs inserts to the Coherence Cache and subsequently to the database, a sequence which breaks the XA transaction contract.

However, you **can** use XA transactions with **read** operations using Database Adapter/Coherence Integration.

Database transactions using the Database Adapter that do not use Database Adapter/Coherence Integration can still use the XA transaction model.

10

Oracle JCA Adapter for MQ Series

This chapter describes how to use the Oracle JCA Adapter for MQ Series (Oracle MQ Series Adapter), which works with Oracle BPEL Process Manager (Oracle BPEL PM) and Oracle Mediator (Mediator) as an external service. The chapter describes JCA Adapter for MQ Series concepts, features, configuration and use cases.

This chapter includes the following sections:

- Section 10.1, "MQ Series Message Queuing Concepts"
- Section 10.2, "Introduction to Native Oracle MQ Series Adapter"
- Section 10.3, "Oracle MQ Series Adapter Features"
- Section 10.4, "Oracle MQ Series Adapter Concepts"
- Section 10.5, "Configuring the Oracle MQ Series Adapter"
- Section 10.6, "Oracle MQ Series Adapter Use Cases"

10.1 MQ Series Message Queuing Concepts

Message queuing is a technique for asynchronous program-to-program communication. It enables application integration by allowing independent applications on a distributed system to communicate with each other. One application sends messages to a queue owned by a queue manager, and another application retrieves the messages from the queue. The communication between applications is maintained even if the applications run at different times or are temporarily unavailable.

The basic concepts of message queuing are described in the following list:

- Messaging

  Messaging is the mechanism that allows two entities to communicate by sending and receiving messages. Messaging can be of two types, synchronous and asynchronous. In *synchronous* messaging, the sender of the message places a message on a message queue and then waits for a reply to its message before resuming its own processing. In *asynchronous* messaging, the sender of the message proceeds with its own processing without waiting for a reply.

- Message

  Messages are structured data sent by one program and intended for another program.

- Message Queue

Message queues are objects that store messages in an application. Applications can put messages to the queues and get messages from the queues. A queue is managed by a queue manager.

- Queue Manager

  A queue manager provides messaging and queuing services to applications through an application programming interface. It provides you with access to the queues and also transfers messages to other queue managers through message channels.

- Message Channel

  A message channel provides a communication path between two queue managers. It connects queue managers. A message channel can transmit messages in one direction only.

- Transmission Queue

  A transmission queue is used to temporarily store messages that are destined for a remote queue manager.

- Message Segment

  If a message is very large, then it can be divided into multiple small messages, called segments. Each segment has a group ID and an offset. All segments of a message have the same group ID. The last segment of the message is marked with a flag.

- Message Group

  A message group consists of a set of related messages with the same group ID. Each message in a message group has a message sequence number. The last message in a message group is marked with a flag.

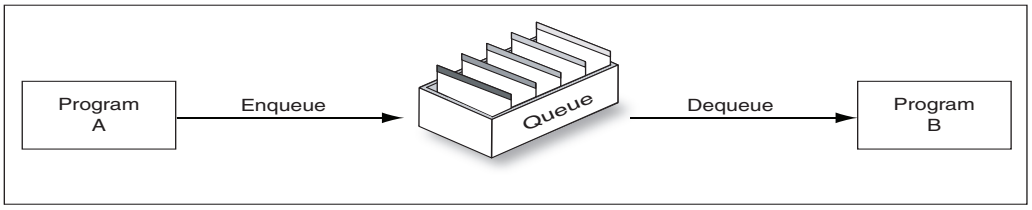
- Cluster

  A cluster is a group of queue managers that are logically associated.

- Enqueue/Dequeue

  To enqueue is to put a message in a queue whereas to dequeue is to get a message from a queue, as shown in Figure 10–1.

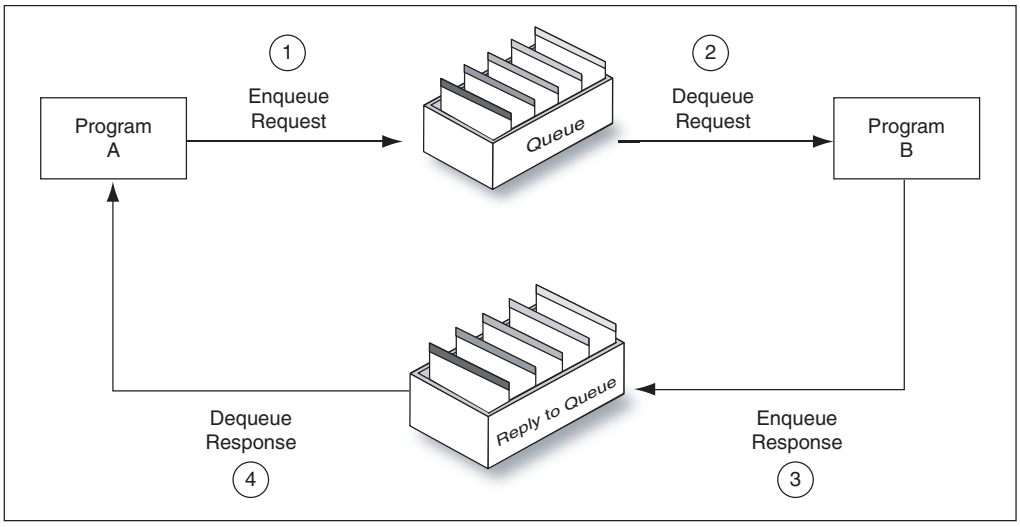
Figure 10–1 Enqueue/Dequeue



- Request/Response

  In a request/response interaction, a program sends a message to another program asking for a reply. The request message contains information about where the reply should be sent. The receiving program sends a reply message in response to the request message. The request/response interaction is shown in Figure 10–2.

Figure 10–2 Request/Response Interaction



For more information about the interaction scenarios supported by the Oracle MQ Series Adapter, see Section 10.4.1.2, "Dequeue Message".

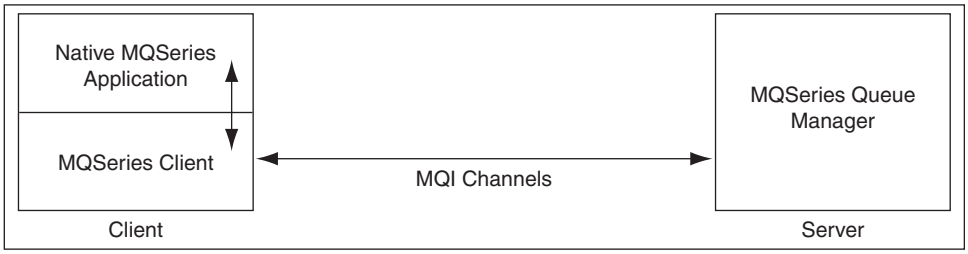
10.1.1 MQ Series Concepts

Messaging and Queuing Series (MQ Series) is a set of products and standards developed by IBM. MQ Series provides a queuing infrastructure that provides guaranteed message delivery, security, and priority-based messaging.

Note: The Oracle MQ Series Adapter is certified on IBM WebSphere MQ V7.0.

The communication process between an MQ Series application and an MQ Series server is shown in Figure 10–3. An MQ Series client enables an application to connect to a queue manager on a remote computer.

Figure 10–3 The MQ Series Communication Process



Every queue in MQ Series belongs to a queue manager. A queue manager has a unique name and provides messaging and queuing services to applications through a Message Queue Interface (MQI) channel. A queue manager also provides access to the queues created on it and transfers messages to other queue managers through message channels.

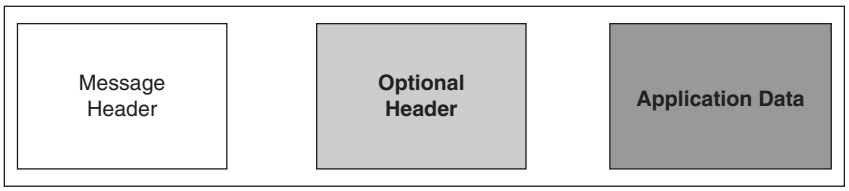
In MQ Series, data is sent in the form of messages. The sending application constructs a message and sends it to a queue by using API calls. The message remains in the

queue until the receiving application is ready to receive it. The receiving application gets the messages from the queue by using API calls.

For sending messages to a remote queue, the remote queue definition must be defined locally. The remote queue definition consists of the destination queue name and the transmission queue name.

Figure 10–4 displays the message structure of an MQ Series message.

Figure 10–4 MQ Series Message



An MQ Series message consists of the following parts, as shown in Figure 10–4:

- Message Header

  The message header contains information such as unique message ID, message type, message priority, and routing information. Every MQ Series message must have a message header.

- Optional Header

  The optional header is required for communication with specific applications, such as the CICS application.

  For more information, see Section 10.4.8, "Integration with CICS".

- Application Data

  This contains the actual data, for example, a record from an indexed or flat file or a row or column from a DB2 table.

10.2 Introduction to Native Oracle MQ Series Adapter

Oracle BPEL Process Manager and Mediator include the Oracle MQ Series Adapter. The Oracle MQ Series Adapter enables applications to connect to MQ Series queue managers and place MQ Series messages on queues or to remove MQ Series messages from queues.

This section contains the following topics:

- Section 10.2.1, "The Need for Oracle MQ Series Adapter"
- Section 10.2.2, "Oracle MQ Series Adapter Integration with Oracle BPEL Process Manager"
- Section 10.2.3, "Oracle MQ Series Adapter Integration with Mediator"

10.2.1 The Need for Oracle MQ Series Adapter

The Oracle MQ Series Adapter provides all native MQ Series functionalities. Although you can configure the Oracle JCA Adapter for JMS (Oracle JMS Adapter) with MQ Series provider, it provides only the JMS functionalities provided by MQ Series and not the native MQ Series functionalities. The following list explains the advantages of Oracle MQ Series Adapter over the Oracle JMS Adapter:

- The Oracle MQ Series Adapter supports Positive Action Notification (PAN) and Negative Action Notification (NAN).
- The Oracle MQ Series Adapter supports report messages such as confirmation on delivery, confirmation on arrival, exception report, and expiry report.
- The Oracle MQ Series Adapter supports sending unwanted or corrupted messages to a dead-letter queue.
- The Oracle MQ Series Adapter provides advanced filter options, such as filtering message belonging to a group.
- The Oracle MQ Series Adapter is faster and easier to use.

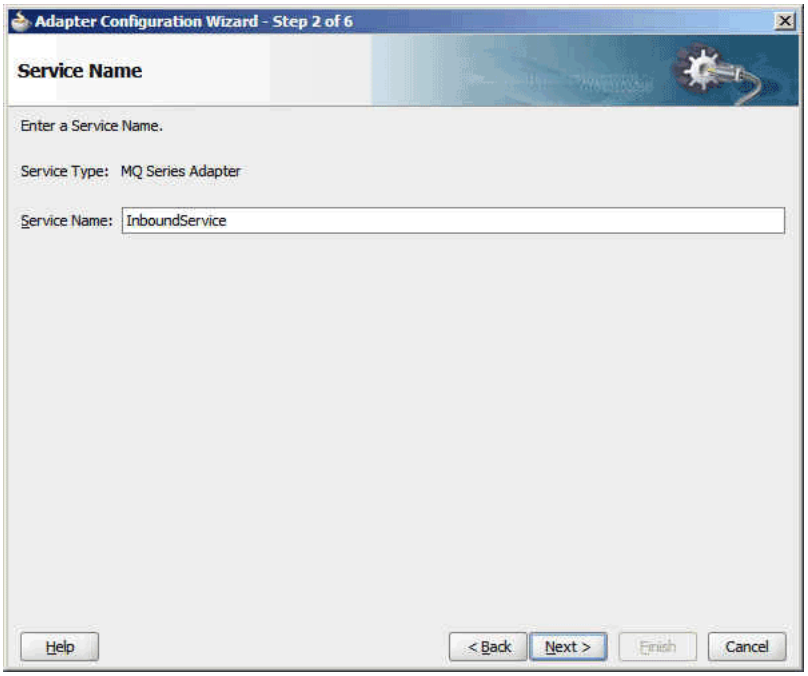
Note: MQ Series version that the Oracle MQ Series Adapter is certified is 6.0.0.0 version, both on Windows and Linux.

10.2.2 Oracle MQ Series Adapter Integration with Oracle BPEL Process Manager

The Oracle MQ Series Adapter is automatically integrated with Oracle BPEL Process Manager. When you create a partner link or an MQ adapter service in Oracle JDeveloper (JDeveloper), the Adapter Configuration Wizard is started.

This wizard enables you to select and configure the Oracle MQ Series Adapter or other Oracle JCA Adapters. The Adapter Configuration Wizard then prompts you to enter a service name, as shown in Figure 10–5.

Figure 10–5 The Service Name Page



When the configuration is complete, a WSDL file of the same name is created in the Application Navigator section of JDeveloper. This WSDL file contains the configuration information you specify with the Adapter Configuration Wizard.

The Operations page of the Adapter Configuration Wizard prompts you to select an operation to perform. Based on your selection, different Adapter Configuration Wizard pages appear and prompt you for configuration information.

Table 10–1 lists the available operations and provides references to sections that describe the information about these operations.

Table 10–1 Supported Operations for Oracle BPEL Process Manager

Operation	See Section...
Enqueue Message	Section 10.4.1.1, "Enqueue Message"
Dequeue Message	Section 10.4.1.2, "Dequeue Message"
Request-Response	Section 10.4.1.3, "Asynchronous Request-Response (Oracle BPEL PM As Client)"
	Section 10.4.1.4, "Synchronous Request-Response (Oracle BPEL PM As Server)"
	Section 10.4.1.5, "Asynchronous Request-Response (Oracle BPEL PM As Server)"
	Section 10.4.1.6, "Synchronous Request-Response (Mediator As Server)"
	Section 10.4.1.7, "Synchronous Request-Response (Oracle BPEL PM As Client)"
	Section 10.4.1.8, "Synchronous Request-Response (Oracle Mediator as Client)"
	Section 10.4.1.9, "Asynchronous Request-Response (Oracle Mediator As Client)"
Outbound Dequeue	Section 10.4.1.10, "Outbound Dequeue Scenario"

10.2.3 Oracle MQ Series Adapter Integration with Mediator

The Oracle MQ Series Adapter is automatically integrated with Mediator. When you create an MQ adapter service in JDeveloper Mediator Designer, the Adapter Configuration Wizard is started.

This wizard enables you to select and configure the Oracle MQ Series Adapter. When the configuration is complete, a WSDL file of the same name is created in the Application Navigator section of JDeveloper. This WSDL file contains the configuration information you specify in the Adapter Configuration Wizard.

The Operations page of the Adapter Configuration Wizard prompts you to select an operation to perform. Based on your selection, different Adapter Configuration Wizard pages appear and prompt you for configuration information. Table 10–2 lists the available operations and provides references to sections that describe the configuration information you must provide.

Table 10–2 Supported Operations for Oracle Mediator

Operation	See Section...
Enqueue Message	Section 10.4.1.1, "Enqueue Message"
Dequeue Message	Section 10.4.1.2, "Dequeue Message"
Request-Response	Section 10.4.1.6, "Synchronous Request-Response (Mediator As Server)"
	Section 10.4.1.8, "Synchronous Request-Response (Oracle Mediator as Client)"
	Section 10.4.1.9, "Asynchronous Request-Response (Oracle Mediator As Client)"
Outbound Dequeue	Section 10.4.1.10, "Outbound Dequeue Scenario"

10.3 Oracle MQ Series Adapter Features

This section explains the following features of the Oracle MQ Series Adapter:

- Section 10.3.1, "RFH Version 2 (RFH2) Header"
- Section 10.3.2, "SSL Enabling"
- Section 10.3.3, "XA Transactions"
- Section 10.3.4, "High Availability"
- Section 10.3.5, "Scalability"
- Section 10.3.6, "Securing Enterprise Information System Credentials"
- Section 10.3.7, "Fault Policy"
- Section 10.3.8, "Inbound Rejection Handler"
- Section 10.3.9.1, "JCA Inbound Retry Mechanism"
- Section 10.3.9.2, "Message Backout Queue"
- Section 10.3.10, "Performance Tuning"

10.3.1 RFH Version 2 (RFH2) Header

The RFH2 header is an extensible header. The RFH2 header enables you to add more header properties to the payload. The RFH2 header carries JMS-specific data that is associated with the message content and can also carry additional information that is not directly associated with JMS.

The RFH2 header consists of two parts, a fixed portion and a variable portion.

10.3.1.1 Fixed Portion

The fixed portion is modeled on the `standard` WebSphere MQ header pattern and consists of the following fields:

StrucId (MQCHAR4)

Structure identifier.

Must be MQRFH_STRUC_ID (value: "RFH ") (initial value).

MQRFH_STRUC_ID_ARRAY (value: "R","F","H"," ") is also defined in the usual way.

Version (MQLONG)

Structure version number.

Must be MQRFH_VERSION_2 (value: 2) (initial value).

StrucLength (MQLONG)

Total length of MQRFH2, including the NameValueData fields.

The value set into StrucLength must be a multiple of 4 (the data in the NameValueData fields may be padded with space characters to achieve this).

Encoding (MQLONG)

Data encoding.

Encoding of any numeric data in the portion of the message following MQRFH2 (the next header, or the message data following this header).

CodedCharSetId (MQLONG)

Coded character set identifier.

Representation of any character data in the portion of the message following MQRFH2 (the next header, or the message data following this header).

Format (MQCHAR8)

Format name.

Format name for the portion of the message following MQRFH2.

Flags (MQLONG)

Flags.

MQRFH_NO_FLAGS =0. No flags set.

NameValueCCSID (MQLONG)

The coded character set identifier (CCSID) for the NameValueData character strings contained in this header. The NameValueData may be coded in a character set that differs from the other character strings that are contained in the header (StrucID and Format).

If the NameValueCCSID field is a 2-byte Unicode CCSID (1200, 13488, or 17584), then the byte order of the Unicode CCSID is the same as the byte ordering of the numeric fields in MQRFH2. (For example, Version, StrucLength, and NameValueCCSID itself.)

The NameValueCCSID field may take only values from Table 10–3:

Table 10–3 Possible Values for NameValueCCSID Field

Value	Meaning
1200	UCS2 open-ended
1208	UTF8
13488	UCS2 2.0 subset
17584	UCS2 2.1 subset (includes the Euro symbol)

10.3.1.2 Variable Portion

The variable portion follows the fixed portion. The variable portion contains a variable number of MQRFH2 folders. Each folder contains a variable number of elements or properties. The related properties are grouped together. The MQRFH2 header can contain the following message service folders:

The <mcd> folder

This contains properties that describe the shape or format of the message. For example, the Msd property identifies the message as being Text, Bytes, Stream. Map, Object, or Null. This folder is always present in JMS MQRFH2.

The <jms> folder

This is used to transport JMS header fields, and JMSX properties that cannot be fully expressed in the MQMD. This folder is always present in a JMS MQRFH2.

The <usr> folder

This is used to transport any application-defined properties associated with the message. This folder is only present if the application has set some application-defined properties.

The <psc> folder

This is used to convey publish/subscribe command messages to the broker. Only one psc folder is allowed in the NameValueData field.

The <pscr> folder

This is used to contain information from the broker, in response to publish/subscribe command messages. Only one pscr folder is present in a response message.

Table 10–4 shows a full list of property names.

Table 10–4 MQRFH2 Folders and Properties Used by JMS

JMS Field Name	Java Type	MQRFH2 Folder name	Property Name	Type/values
JMSDestination	Destination	jms	Dst	string
JMSExpiration	long	jms	Exp	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	String	jms	Cid	string
JMSReplyTo	Destination	jms	Rto	string
JMSTimestamp	long	jms	Tms	i8
JMSType	String	mcd	Type, Set, Fmt	string
JMSXGroupID	String	jms	Gid	string
JMSXGroupSeq	int	jms	Seq	i4
xxx (User Defined)	Any	usr	xxx	any
		mcd	Msd	jms_none
				jms_text
				jms_bytes
				jms_map
				jms_stream
				jms_object

The syntax used to express the properties in the variable portion is as follows:

NameValueLength (MQLONG)

Length, in bytes, of the NameValueData string that immediately follows this length field. It does not include its own length. The value set into NameValueLength is always a multiple of 4. The NameValueData field is padded with space characters to achieve this.

NameValueData (MQCHARn)

A single character string, whose length in bytes is given by the preceding NameValueLength field. It contains a folder holding a sequence of properties. Each

property is a name/type/value triplet, contained within an XML element whose name is the folder name, as follows:

<foldername> triplet1 triplet2 ..... tripletn </foldername>

10.3.2 SSL Enabling

Secure Sockets Layer (SSL) is a protocol for transmitting encrypted data over the Internet or an internal network. SSL works by using public and private keys to encrypt data that is transferred over the SSL connection. Data that has been encrypted with a public key can be decrypted only with the corresponding private key. Conversely, data that has been encrypted with a private key can be decrypted only with the corresponding public key.

MQ Series supports secure communication, with MQ Series clients using SSL. As a part of this functionality, the adapter would provide support to put a message on queue using SSL. To enable Oracle MQ Series Adapter for SSL, the following properties must be provided:

- **SSLEnable**: The true/false value for this property means that the Oracle MQ Series Adapter is SSL enabled/disabled.
- **KeyStoreLocation**: This is the keystore where Oracle MQ Series Adapter has its private keys. This property is required as the adapter must authenticate itself to the MQ Series server.
- **KeyStorePassword**: This password is required to access keystore.
- **TrustStoreLocation**: This is the location where the adapter keeps its trusted certificates information. This information is required when an adapter must authenticate to the MQ Series server.
- **Protocol**: Key Management Algorithm.
- **KeyStoreProviderName**: The name of the keystore provider.
- **KeyStoreType**: Type of the key store.
- **KeyStoreAlgorithm**: Algorithm used by the key store.
- **CipherSuite**: Set CipherSuite to the name matching the CipherSpec set on the `SVRCONN` channel. If set to null (default), then no SSL encryption is performed.
- **SSLPeerName**: A distinguished name pattern. If CipherSuite is set, then you can use this variable to ensure that the correct queue manager is used. If set to null (default), then the DN of the queue manager is not checked. This variable is ignored if `sslCipherSuite` is null.

10.3.3 XA Transactions

Oracle MQ Series Adapter enables transaction support, which along with the inherent data processing, ensures that each modification has a clearly defined outcome, resulting in either success or failure, thus preventing potential corruption of data, executes independently from other changes, and, after completion, leaves underlying data in the same state until another transaction takes place.

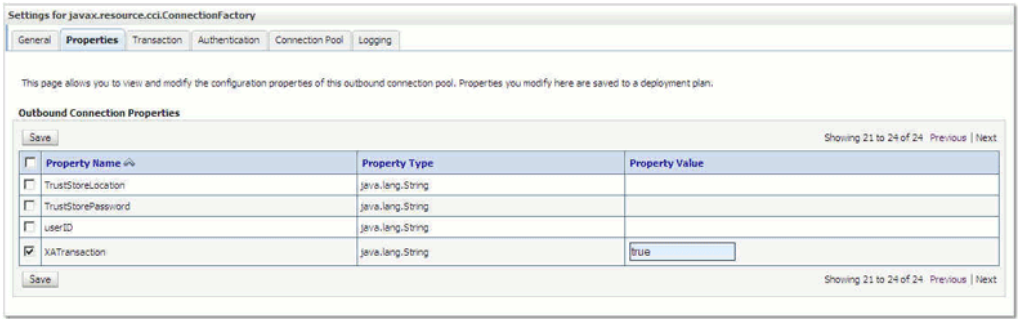
The Oracle MQ Series Adapter supports both inbound and outbound XA transaction. You must set the `XATransaction` property in the Oracle WebLogic Server Administration Console to enable the XA transaction. To enable XA transaction, perform the following steps:

1. Log in to the Oracle WebLogic Server Administration Console using your password credentials.
2. Under Domain Structure, in the left pane, click **Deployments**. The Summary of Deployments page is displayed.
3. Click **MQSeriesAdapter**. The Settings of MQSeriesAdapter page is displayed.
4. Click the **Configuration** tab. The Configuration submenu options are displayed.
5. Click **Outbound Connection Pools**. The Outbound Connection Pool Configuration Table is displayed.
6. Click the + icon next to javax.resource.cci.ConnectionFactory and select **eis/MQ/MQAdapter**. The Outbound Connection Properties page is displayed.

> **Note:** Click **Lock & Edit** to enable the options in the console.

7. Select the **XATransaction** option and click the Property Value row at the end of the XATransaction.
8. Enter `true` in the text field, as shown in Figure 10–6, and click **Save**.

Figure 10–6 Outbound Connection Properties Page



9. Click the **Transaction** tab. The Settings for `javax.resource.cci.ConnectionFactory` page is displayed.
10. Select **XA Transaction** from the Transaction Support list.
11. Click **Save** to save your settings. The Save Deployment Plan Assistant page is displayed.
12. Click **OK**.

You have successfully enabled XA transaction for the Oracle MQ Series Adapter.

In order to use the XA transaction feature for MQ Series with BPEL for synchronous inbound request-reply scenario, you must set the `bpel.config.transaction` parameter to **required**. If this parameter is not set, then it causes the transaction to split at the BPEL boundary and MQ returns `MQRC_SYNCPOINT_NOT_AVAILABLE` error code.

```
<property name="bpel.config.transaction">required
</property>
```

10.3.3.1 XA Recovery

In a scenario involving fail over, such as when the `prepare phase` completes successfully before a middleware fails, messages must be recovered within the adapter without restarting the MQSeries server. You must manually resolve the in-doubt transactions.

To view all in-doubt transactions for a Queue Manager, you must execute the following command at the command prompt:

```
dspmqtrn -m[ourQueueManager]
```

To backout the messages, use the following command:

```
rsvmqtrn -m[ourQueueManager] -b [Transaction],[Number]
```

To commit the messages, use the following command :

```
rsvmqtrn -m[ourQueueManager] -c [Transaction],[Number]
```

Note: You can use the `[Transaction]` and `[Number]` from the output of the `dspmqtrn` command.

10.3.4 High Availability

The Oracle MQ Series Adapter supports the high availability feature for the active-active topology with Oracle BPEL Process Manager (Oracle BPEL PM) and Mediator service engines. It supports this feature for both inbound and outbound operations.

10.3.4.1 Prerequisites for High Availability

Before you configure the Oracle MQ Series Adapter for high availability, you must ensure that the following prerequisites are met:

- Clustered processes must use the same queue.
- Fault-policies and fault-bindings must be created for remote faults to ensure that the adapter acts correctly.

10.3.4.2 High Availability in Inbound/Outbound Operations

The Oracle MQ Series Adapter must ensure that it participates in the XA transaction. For more information about the XA transaction, see Section 10.3.3, "XA Transactions".

10.3.5 Scalability

The Oracle MQ Series Adapter supports the scalability feature for inbound operations only. Oracle MQ Series Adapter provides the parameter to control the number of threads that dequeue the messages from the inbound queue.You must specify the following property in the `.jca` file:

```
InboundThreadCount='N'
```

where, N is the number of threads to span to dequeue the messages from the inbound queue. The default setting is 2.

The Oracle MQ adapter creates the back-endconnections at deployment time, that is, at that time the adapter endpoint starts polling. You can have the application server prewarm the connection pool, which would provide a small marginal advantage,

although connection creation does not otherwise delay the overall deployment task itself.

The example syntax for using InboundThreadCount in the .jca file is:

```
<adapter-config name="ExpressDeathEventListener" adapter="MQSeriesAdapter"
wsdlLocation="ExpressDeathEventListener.wsdl"
<xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  <connection-factory location="eis/MQ/MQAdapter" adapterRef=""/>
   <endpoint-activation portType="Dequeue_ptt" operation="Dequeue"
UITransmissionPrimitive="Dequeue">
   <activation-spec className="oracle.tip.adapter.mq.inbound.ActivationSpecImpl">
      <property name="QueueName" value="BPMPOC_EXPCLAIMQ"/>
      <property name="InboundThreadCount" value="10"/>
   </activation-spec>
  </endpoint-activation>
</adapter-config>
```

10.3.6 Securing Enterprise Information System Credentials

The Oracle MQ Series Adapter supports securing of the Enterprise Information System (EIS) credentials such as the user name and password, whenever it establishes an outbound connection with EIS. You can secure the user name and password for Oracle MQ Series Adapter by using Oracle WebLogic Server container-managed sign-on.

For more information, see Section 4.2.22, "Securing Enterprise Information System Credentials".

10.3.7 Fault Policy

A fault policy file defines fault conditions and their corresponding fault recovery actions. Each fault condition specifies a particular fault or group of faults, which it attempts to handle, and the corresponding action for it. A set of actions is identified by an ID in the fault policy file.

The Oracle MQ Series Adapter supports defining rejection handlers by using fault policies.

For more information about fault policies, see Section 2.21.1.1, "Configuring Rejection Handlers".

10.3.8 Inbound Rejection Handler

The Oracle MQ Series Adapter supports inbound message rejection handling. You can configure the message rejection handler to process translation errors, take corrective action.

For more information about rejection handlers, Section 2.21.1.1, "Configuring Rejection Handlers".

10.3.9 Retry Mechanism

The Oracle MQ Series Adapter supports the following two mechanisms for inbound retry:

- JCA Inbound Retry Mechanism
- Message Backout Queue

The JCA inbound retry mechanism is commonly used by all adapters, in general, whereas the message backout queue mechanism is used only by the Oracle MQ Series Adapter. If you specify the `BackoutQueueName` property in the .jca file, only then the Oracle MQ Series Adapter uses the message backout queue mechanism to retry. By default, the JCA inbound retry mechanism is used for retry.

Note: Both these methods of retry in the Oracle MQ Series Adapter are mutually exclusive operations; the adapter uses one mechanism at a time. If you specify both options, then the Backout Queue option takes precedence.

10.3.9.1 JCA Inbound Retry Mechanism

The Oracle MQ Series Adapter supports a pull model for connecting to the back-end application for receiving events. Connection-related issues are considered recoverable and most inbound adapters keep retrying until the adapters are able to establish connection with the EIS.

In case of Oracle MQ Series Adapter, a message not being able to put to a queue is also retriable.

For more information about retry mechanism, see Section 2.21, "Error Handling".

10.3.9.2 Message Backout Queue

Backout Queue is a queue for putting rejected messages from an inbound queue. The inbound adapter checks for the backout count of the messages and if this count exceeds the `MaximumBcakoutCount` value, then the adapter puts the messages to the specified Backout Queue. This mechanism is used by the Oracle MQ Series Adapter to handle inbound retries for the rejected messages.

If you specify the `BackoutQueueName` property in the `.jca` file, then Oracle MQ Series Adapter uses the message backout count for retries. You can specify the maximum retries using the `MaximumBackoutCount` property. The default value for this property is infinite. If you do not specify the `MaximumBackoutCount` value along with the `BackoutQueueName`, then the adapter retries infinitely. The adapter does not consider `JCA` retries (specified in composite.xml) when the BackOut Queue properties are specified.

The `BackoutRetries` property must be set to specify the number of retries for delivering the message to the Backout Queue with retry interval set using the `BackoutRetryInterval` property. The default value for `BackoutRetries` is `3` and `BackoutInterval` is `5` sec.

If a message gets rejected even after the `MaximumBackoutCount` value is reached, then the adapter puts the message to Backout Queue. If Oracle MQ Series Adapter is cannot put the message to Backout Queue, then the adapter tries till the `BackoutRetries` count with the `BackoutInterval` time delay. If even after the BackoutRetries the adapter cannot put the message to Backout Queue, then the adapter deactivates the endpoint.

You must also specify the name of the Queue Manager of the Backout Queue in the `BackoutQueueManagerName` property. You must not use this property if the BackoutQueue resides on the inbound queue QueueManager.

Note: When using the Backout Queue, consider the following:

- The Backout Queue options cannot be used for translation failures.
- In cases where both JCA and BackOut retries are specified, the BackOut retries takes precedence.

For more information about configuring Backout Queues, see Section 10.6.7, "Configuring a Backout Queue."

10.3.10 Performance Tuning

The Oracle MQ Series Adapter supports performance tuning options.

For more information, see "Oracle MQ Adapter Tuning" in the Oracle Fusion Middleware Performance and Tuning Guide.

10.4 Oracle MQ Series Adapter Concepts

This section explains the following concepts of the Oracle MQ Series Adapter:

- Section 10.4.1, "Messaging Scenarios"
- Section 10.4.2, "Message Properties"
- Section 10.4.3, "Correlation Schemas"
- Section 10.4.4, "Distribution List Support"
- Section 10.4.5, "Report Messages"
- Section 10.4.6, "Message Delivery Failure Options"
- Section 10.4.7, "Message Segmentation"
- Section 10.4.8, "Integration with CICS"
- Section 10.4.9, "Supported Encodings"

10.4.1 Messaging Scenarios

The Oracle MQ Series Adapter supports the following messaging scenarios:

- Section 10.4.1.1, "Enqueue Message"
- Section 10.4.1.2, "Dequeue Message"
- Section 10.4.1.3, "Asynchronous Request-Response (Oracle BPEL PM As Client)"
- Section 10.4.1.4, "Synchronous Request-Response (Oracle BPEL PM As Server)"
- Section 10.4.1.5, "Asynchronous Request-Response (Oracle BPEL PM As Server)"
- Section 10.4.1.6, "Synchronous Request-Response (Mediator As Server)"
- Section 10.4.1.7, "Synchronous Request-Response (Oracle BPEL PM As Client)"
- Section 10.4.1.8, "Synchronous Request-Response (Oracle Mediator as Client)"
- Section 10.4.1.9, "Asynchronous Request-Response (Oracle Mediator As Client)"
- Section 10.4.1.10, "Outbound Dequeue Scenario"

10.4.1.1 Enqueue Message

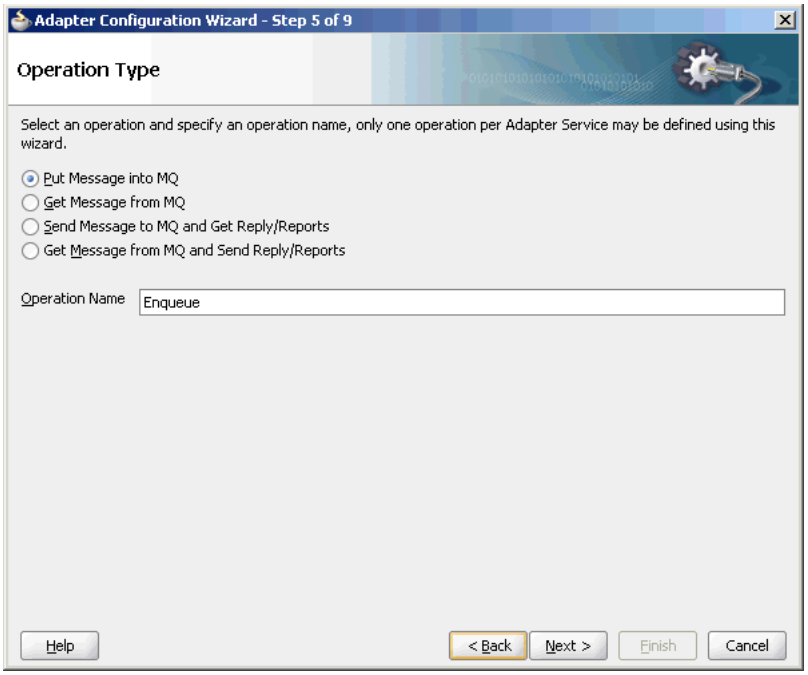
In this scenario, the Oracle MQ Series Adapter connects to a specific queue managed by a queue manager and then writes the message to the queue. For outbound messages sent from Oracle BPEL PM or Mediator, the Oracle MQ Series Adapter performs the following operations:

1. Receives message from Oracle BPEL PM or Mediator.
2. Formats the XML content as specified at design time.
3. Sets the properties of the message, such as priority, expiry, message type, and persistence. These properties are based on the selections that you made in the Adapter Configuration Wizard.

   For more information about message properties, see Section 10.4.2.1, "Messages Types".
4. Sends the message to the queue specified at design time in the Adapter Configuration Wizard.

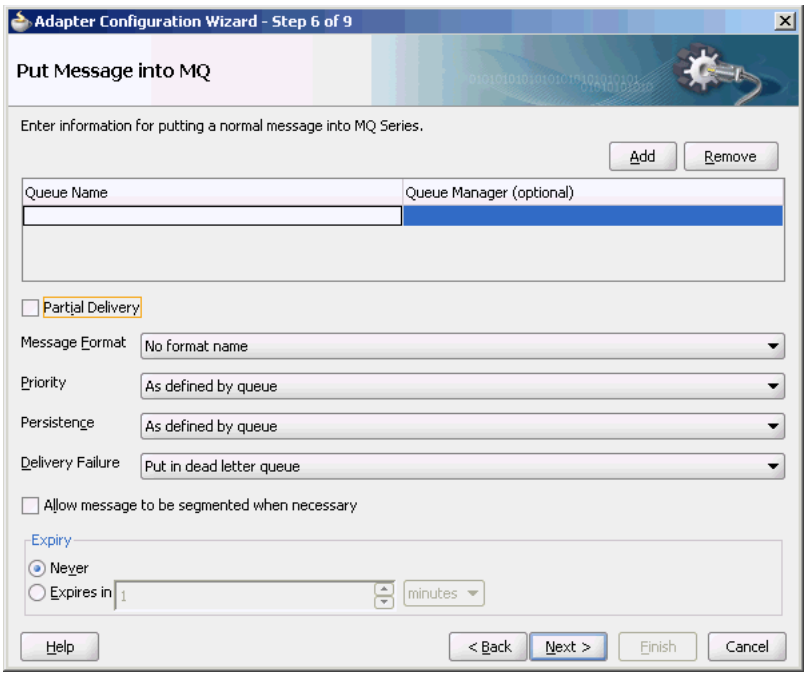
Figure 10–7 displays the operation type that you must select in the Adapter Configuration Wizard.

Figure 10–7 The Adapter Configuration Wizard: Produce Message Selection



The page that appears after selecting the Put Message into MQ operation type is shown in Figure 10–8.

Figure 10–8 Put Message Options



You can specify the following properties in this page:

- **Queue Name**: The name of the queue on which the Oracle MQ Series Adapter enqueues the message. This is a mandatory field.
- **Queue Manager**: The name of the queue manager to which the queue belongs. This field is optional and is necessary when enqueuing message to a remote queue.
- **Partial Delivery**: This is applicable only when you specify multiple queues for outbound operation, which is also known as the Distribution List scenario. Partial Delivery takes either `true` or `false`. If assigned `true`, then even if the delivery of message fails for some queues, it would still go and put the message to the rest of the queues specified in the distribution list. If assigned `false`, it means even if one message fails, then the message is not put to any queue.
- **Message Format**: The format of the message.

  Note: When enqueuing a message, ensure that the various mandatory values, required for a specific format, are specified correctly.

- **Priority**: The priority of the message, ranging from 0 (low) to 9 (high).
- **Persistence**: The persistence of the message. You can also specify the persistence of the message to be taken from the default persistence attribute, as defined by the destination queue.
- **Delivery Failure**: If the delivery of message fails, then either it can be put to a dead letter queue or it can be discarded.
- **Allow Messages to Be Segmented When Necessary**: This is applicable to a message that is big enough for the queue to accommodate. In that case, if you have

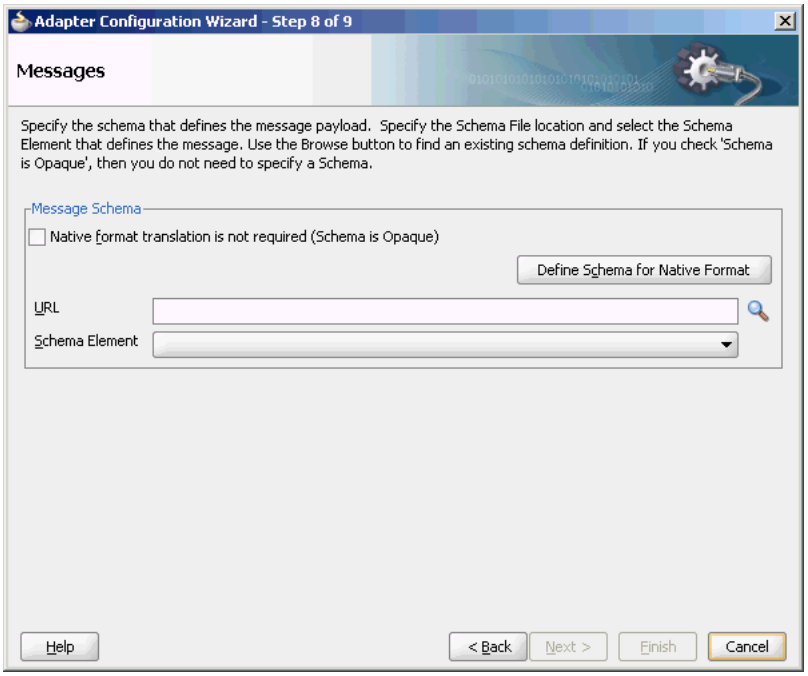
specified that it has to be segmented, then the single message can be broken into that many bytes the queue can take, which results in multiple messages.

- **Expiry**: The expiry time of the message. The message is discarded after the expiry time has elapsed.

For more information about these properties, see Section 10.4.2, "Message Properties".

The next Adapter Configuration Wizard page that appears is the Messages page, as shown in Figure 10–9. This page enables you to select the XML Schema Definition (XSD) file for translation.

Figure 10–9 Messages Page



If native format translation is not required (for example, a JPG or GIF image is being processed), then select the **Native format translation is not required** check box. The file is passed through in base-64 encoding.

XSD files are required for translation. To define a new schema or convert an existing data type description (DTD) or COBOL Copybook, select **Define Schema for Native Format**. This starts the Native Format Builder wizard. This wizard guides you through the creation of a native schema file from file formats, such as delimited by special characters, comma-delimited value (CSV), fixed-length, DTD, and COBOL Copybook. After the native schema file is created, you are returned to this Messages page with the **Schema File URL** and **Schema Element** fields filled in.

For more information, see Section 6.1, "Creating Native Schema Files with the Native Format Builder Wizard".

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

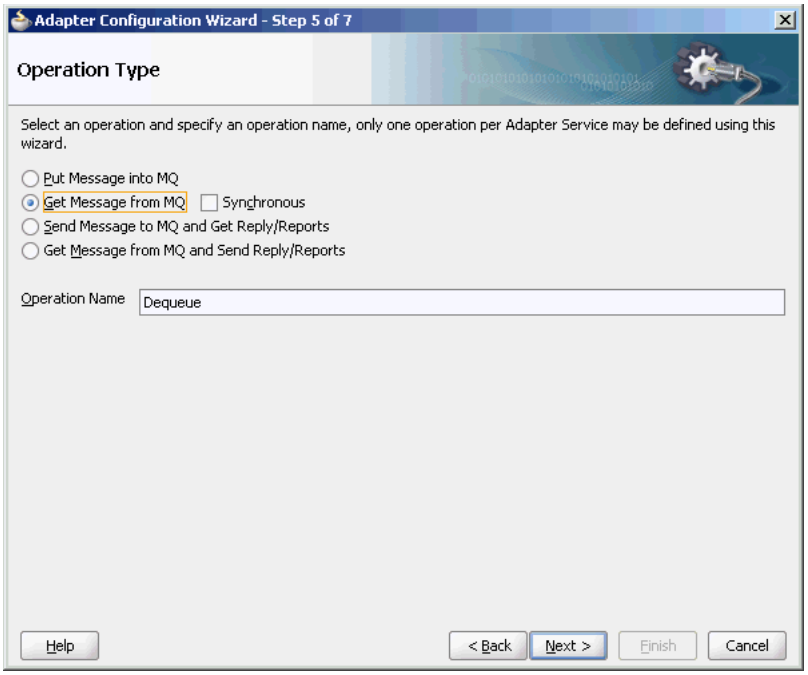
10.4.1.2 Dequeue Message

In this scenario, the Oracle MQ Series Adapter connects to a specific queue managed by a queue manager and then removes the message from the queue. For inbound messages sent to Oracle BPEL PM or Mediator, the Oracle MQ Series Adapter performs the following operations:

1. Connects to the queue specified at design time.
2. Dequeues the message from the queue when a message arrives.
3. Reads and translates the message based on the translation logic defined at design time.
4. Publishes the message as an XML message to Oracle BPEL PM or Mediator.

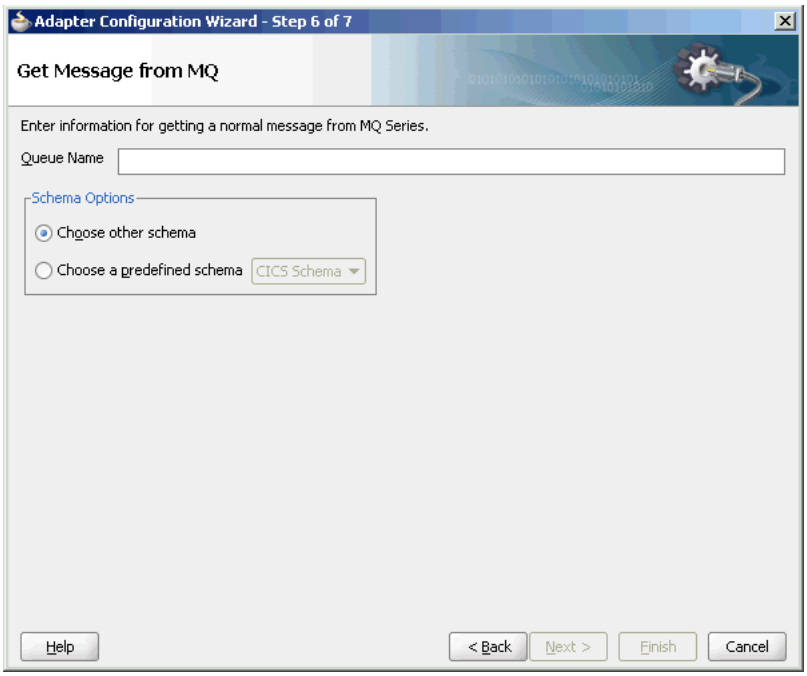
   Figure 10–10 displays the operation type that you must select in the Adapter Configuration Wizard.

Figure 10–10 The Adapter Configuration Wizard: Consume Message Selection



The page that appears after selecting the **Get Message from MQ** operation type is shown in Figure 10–11.

Figure 10–11 Get Message from MQ Page



You can specify the following properties in this page:

- **Queue Name**: The name of the queue from which the Oracle MQ Series Adapter dequeues the message. This is a mandatory field.
- **Schema Options**: This option enables you to specify the schema for the message to be dequeued.
  - **Choose Other Schema**: This option enables you to choose your schema for the message to be dequeued.
  - **Choose a Predefined Schema**: This option enables you to choose a readymade schema that the adapter provides.

The next Adapter Configuration Wizard that appears is the Messages page, as shown in Figure 10–9. This page enables you to select the XSD schema file for translation.

As with specifying the schema for the produce message operation, you can perform the following tasks in this page:

- Specify if native format translation is not required.
- Select the XSD schema file for translation.
- Start the Native Format Builder wizard to create an XSD file from file formats such as CSV, fixed-length, DTD, and COBOL Copybook.

For more information about the Messages page, see Section 10.4.1.1, "Enqueue Message".

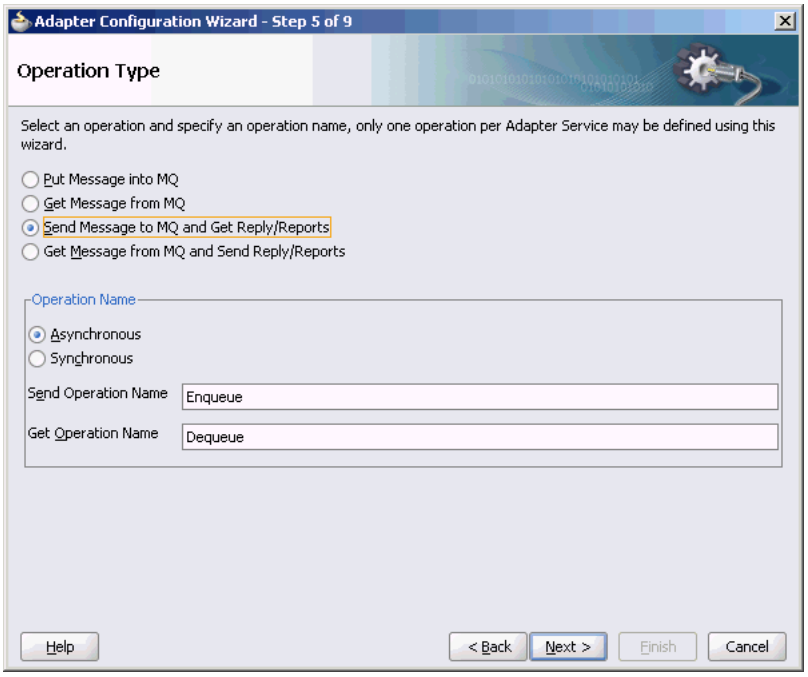
10.4.1.3 Asynchronous Request-Response (Oracle BPEL PM As Client)

In this scenario, the Oracle BPEL PM sends a request message and receives the corresponding response using a non-initiating receive activity. The invoke activity initiates the outbound invocation of the adapter to send the request. The Oracle MQ Series Adapter performs the following operations:

1. Receives message from Oracle BPEL PM.
2. Formats the XML content as specified at design time in the Adapter Configuration Wizard.
3. Sets properties and a correlation schema on the request message.
4. Sends the message to the queue specified at design time. The third-party application receives the message, processes it, generates the response, and then enqueues the response message to the `replyTo` queue specified in the request message. The Correlation ID and Message ID of the response message are generated based on the correlation schema specified in the request message.
5. The Oracle MQ Series Adapter dequeues the message from the `replyTo` queue.
6. Sends the response to the non-initiating receive activity of Mediator. To ensure that response is sent to the correct BPEL instance, correlation schemas are used.

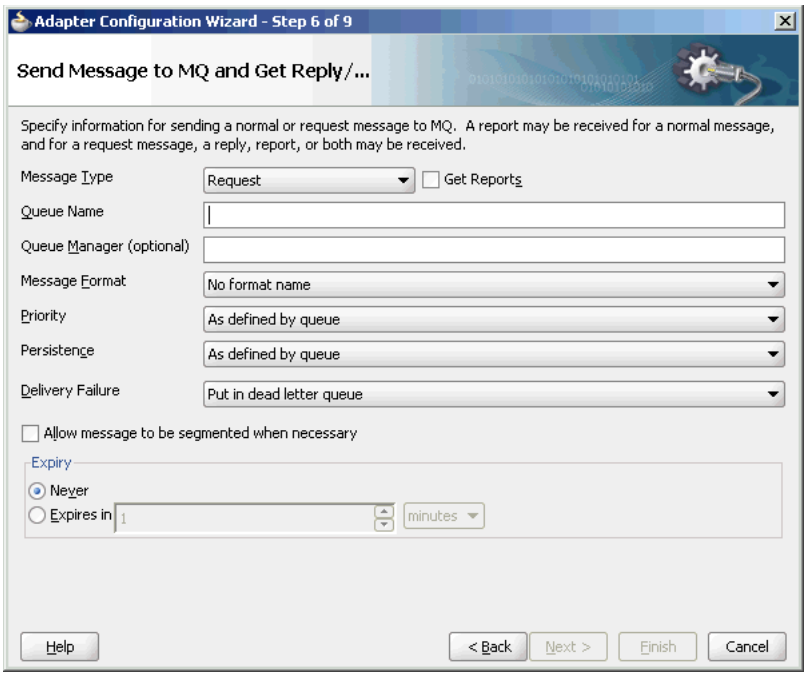
Figure 10–12 displays the operation type that you must select in the Adapter Configuration Wizard.

Figure 10–12 Selecting an Operation Type



The page that appears after selecting the **Send Message to MQ and Get Reply/Reports** operation type is shown in Figure 10–13.

Figure 10–13 Send Message to MQ and Get Reply/Reports Page



You can specify the following properties in this page:

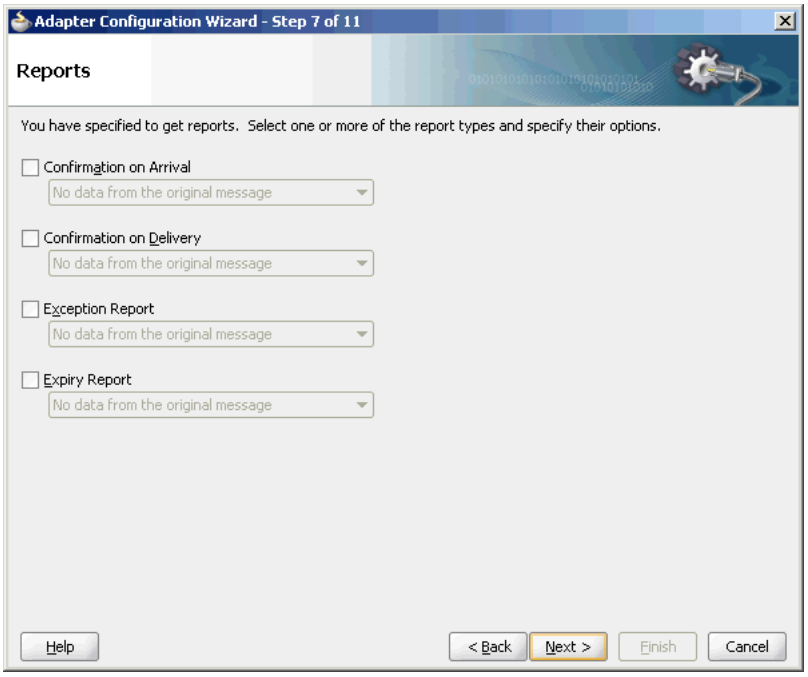
- **Message Type**: The type of the message. You can either send a normal message or a request message.
- **Get Reports**: Select this option if you want any kind of report. You can specify the type of report in the next page, as shown in Figure 10–14.
- **Queue Name**: The name of the queue to which the Oracle MQ Series Adapter enqueues the message. This is a mandatory field.
- **Queue Manager**: The name of the queue manager to which the queue belongs. This field is optional.
- **Message Format**: The format of the message.
- **Priority**: The priority of the message ranging from 0 (low) to 9 (high).
- **Persistence**: The persistence of the message. You can also specify the persistence of the message to be taken from the default persistence attribute, as defined by the destination queue.
- **Delivery Failure**: If the delivery of the message fails, then either it can be put to a dead letter queue or it can be discarded.
- **Allow Messages to Be Segmented When Necessary**: This is applicable to a message that is big enough for the queue to accommodate. In that case, if you have specified that it has to be segmented, then the single message can be broken into that many bytes the queue can take, which results in multiple messages.
- **Expiry**: The expiry time of the message. The message is discarded after the expiry time has elapsed.

For more information about these properties, see Section 10.4.2, "Message Properties" and Section 10.4.5, "Report Messages".

The page that is displayed when you click **Next** in the Send Message to MQ and Get Reply/Reports page can be a Reports page (shown in Figure 10–14) or a Response page (shown in Figure 10–15).

The Reports page, shown in Figure 10–14, is displayed only if you have selected the Get Reports option in the Send Message to MQ and Get Reply/Reports page, as shown in Figure 10–13.

Figure 10–14 Reports Page



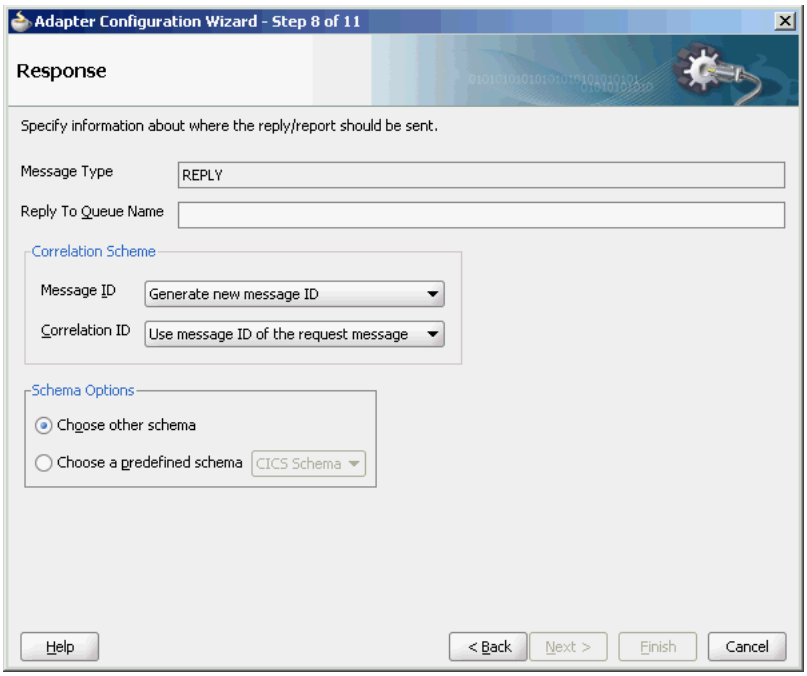
You can select the following types of reports in this page:

- Confirmation on Arrival
- Confirmation on Delivery
- Exception Report
- Expiry Report

For information about these report types, see Section 10.4.5, "Report Messages".

The Response page shown in Figure 10–15 is displayed when you click **Next** in the Reports page.

Figure 10–15 Response Page



You can specify the following properties in the Response page:

- **Reply to Queue Name**: The name of the reply queue name.
- **Correlation Scheme**: The correlation schema that is necessary for the Oracle MQ Series Adapter.

  For information about correlation schemas, see Section 10.4.3, "Correlation Schemas".
- **Schema Options**: This option enables you to specify the schema for the message to be dequeued.
  - **Choose Other Schema**: This option enables you to choose your schema for the message to be dequeued.
  - **Choose a Predefined Schema**: This option enables you to choose a readymade schema that the adapter provides.

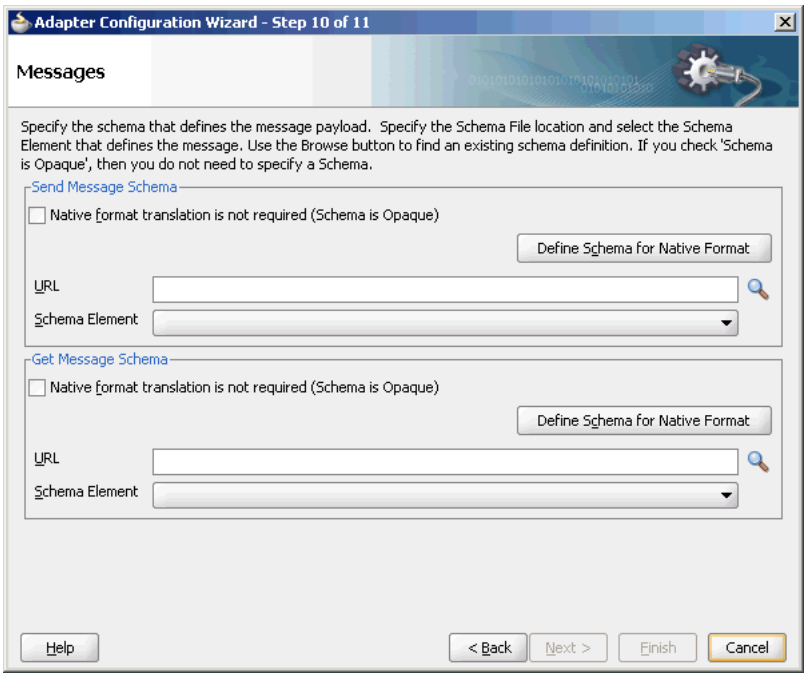
Note: For Oracle MQ Series Adapter in an asynchronous outbound request/reply scenario, properties are differentiated by an `(Enqueue)` or `(Dequeue)` label in Oracle Enterprise Manager Console. For example, `QueueName(Enqueue)` is used for putting a message and `QueueName(Dequeue)` is used for dequeuing the reply.

When using Oracle Enterprise Manager Console to edit Oracle MQ Series Adapter properties in this scenario, note the following:

- If you change the `ReplyToQueueName(Enqueue)` property, you must also change the `QueueName(Dequeue)` property to the same value.
- If you change the `MessageId(Dequeue)` property, you must also change the `MessageId(Enqueue)` property to the same value.
- If you change the `CorrelationId(Dequeue)` property, you must also change the `CorrelationId(Enqueue)` property to the same value.

When you click **Next** in the Response page, a Messages page, shown in Figure 10–16, is displayed. This page enables you to select the XSD schema file for translation for request and as response message.

Figure 10–16 Messages Page



You can perform the following tasks in this page:

- Specify if native format translation is not required.
- Select the XSD schema file for translation.
- Start the Native Format Builder wizard to create an XSD file from file formats such as CSV, fixed-length, DTD, and COBOL Copybook.

For more information about the Messages page, see Section 10.4.1.1, "Enqueue Message".

In the solicit-request-response scenario, the reply message is expected in the reply queue specified with some correlation scheme that is provided through the request message. This reply queue, which is used by a particular process (BPEL/Mediator), should not be used by any other process.

If the same reply queue is used by some other application, then the message might be picked, irrespective of whether the reply message had the proper correlation or not, and eventually the message becomes lost.

10.4.1.4 Synchronous Request-Response (Oracle BPEL PM As Server)

In this scenario, the Oracle BPEL PM receives a request, processes it, and sends the response synchronously by using a reply activity. The Oracle MQ Series Adapter performs the following operations:

1. Dequeues the request message from the queue when the message arrives.
2. Reads and translates the message based on the translation logic defined at design time.
3. Publishes the message as an XML message to Oracle BPEL PM. The Oracle BPEL PM processes the request and sends the response to the Oracle MQ Series Adapter.
4. Receives the response message from the Oracle BPEL PM.
5. Formats the XML content as specified at design time.
6. Sets the properties of the message such as priority, expiry, message type, and persistence. These properties are based on the selections that you made in the Adapter Configuration Wizard.
7. Sends the message to the queue specified at design time in the Adapter Configuration Wizard.

Figure 10–17 shows a sample BPEL process for this scenario.

Figure 10–17 Synchronous Request-Response Oracle BPEL PM As Server Sample

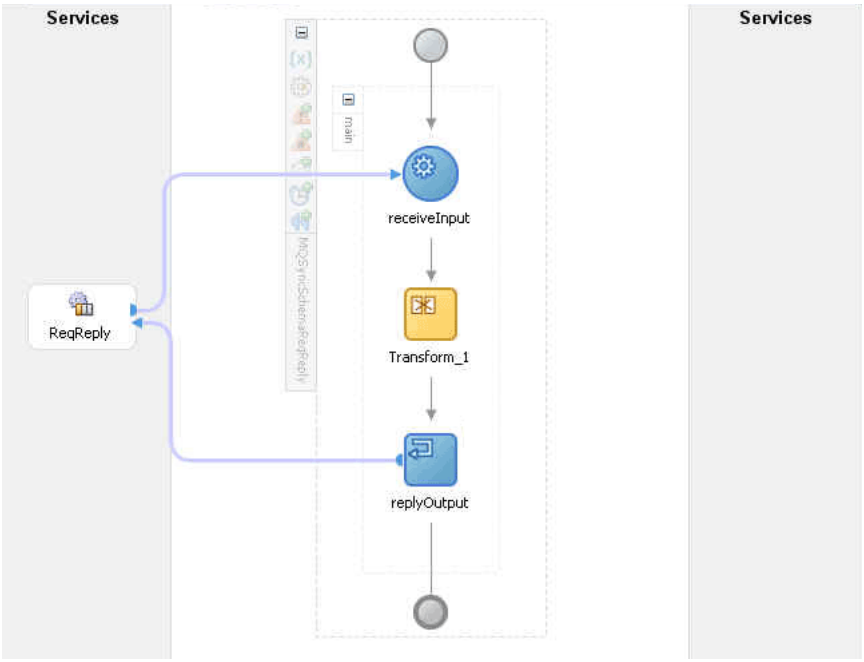
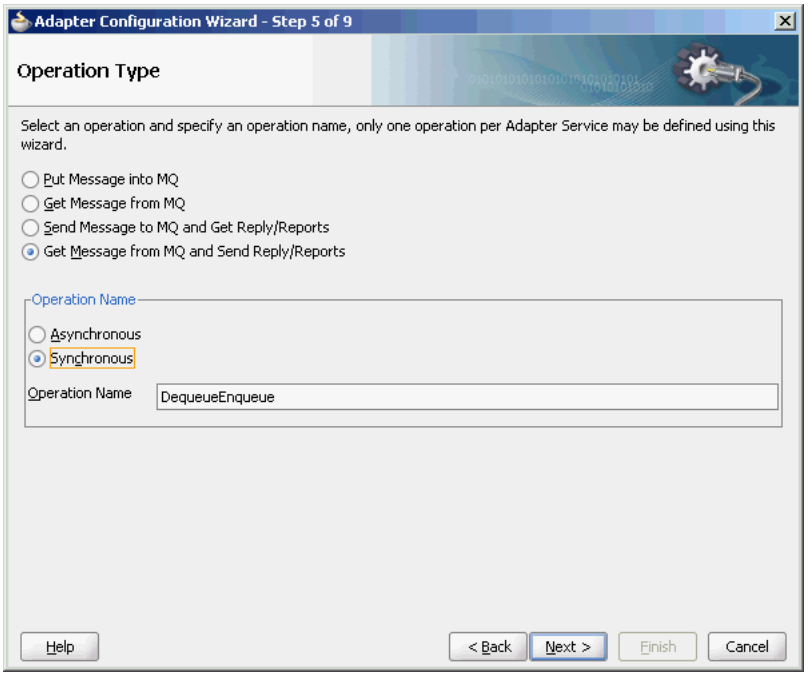


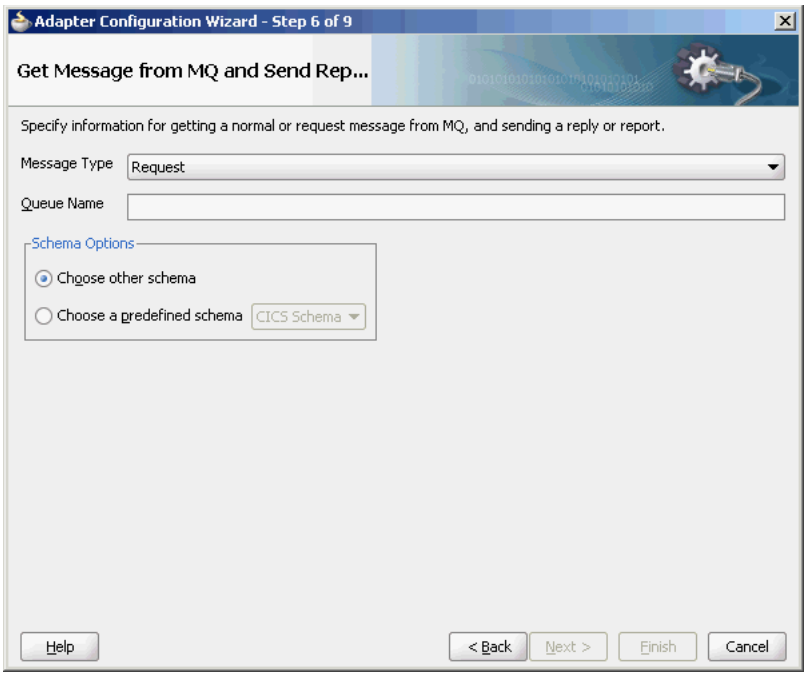
Figure 10–18 displays the operation type that you must select in the Adapter Configuration Wizard.

Figure 10–18 Operation Type Page Selection for Request-Response Synchronous Interaction



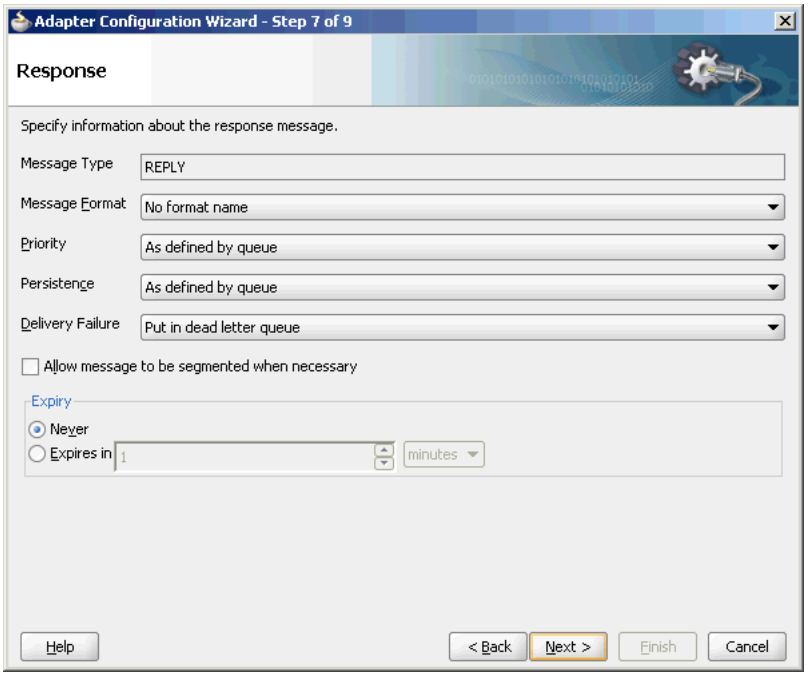
The page that appears after you select the **Get Message from MQ and Send Reply/Reports** operation type is shown in Figure 10–19. Specify the queue name from which the Oracle MQ Series Adapter dequeues the message in this page.

Figure 10–19 Get Message from MQ and Send Reply/Reports Page



When you click **Next** in the Get Message from MQ and send Reply/Reports page, the Response page shown in Figure 10–20 is displayed.

Figure 10–20 Response Page for Synchronous Request-Response



You can specify the following properties in the Response page:

- **Message Type**: The message type of the message to be dequeued. This option affects the return message type.
- **Message Format**: The format of the message.

- **Priority**: The priority of the message.
- **Persistence**: The persistence of the message. You can also specify the persistence of the message to be taken from the default persistence attribute, as defined by the destination queue.
- **Delivery Failure**: If the delivery of the message fails, then either it can be put to a dead letter queue or it can be discarded.
- **Allow Messages to Be Segmented When Necessary**: This is applicable to a message that is big enough for the queue to accommodate. In that case, if you have specified that it has to be segmented, then the single message can be broken into that many bytes the queue can take, which results in multiple messages.
- **Expiry**: The expiry time of the message.

For more information about these properties, see Section 10.4.2, "Message Properties".

Click **Next** in the Response page, the Messages page is displayed, as shown in Figure 10–16. You can perform the following tasks in this page:

- Specify if native format translation is not required.
- Select the XSD schema file for translation.
- Start the Native Format Builder wizard to create an XSD file from file formats such as CSV, fixed-length, DTD, and COBOL Copybook.

For more information about the Messages page, see Section 10.4.1.1, "Enqueue Message".

10.4.1.5 Asynchronous Request-Response (Oracle BPEL PM As Server)

In Oracle BPEL PM initiated request-response interaction, a BPEL process receives a request as an inbound message, processes it, and then sends the response through an invoke activity. For asynchronous request-reply scenario, the Oracle MQ Series Adapter performs the following operations:

1. Dequeues the message from the queue when a message arrives.
2. Reads and translates the message based on the translation logic defined at design time.
3. Publishes the message as an XML message to Oracle BPEL PM. The Oracle BPEL PM processes the request and sends the response to the Oracle MQ Series Adapter.
4. Receives messages from Oracle BPEL PM.
5. Formats the XML content as specified at design time.
6. Sets the properties of the message, such as priority, expiry, message type, and persistence. These properties are based on the selections that you made in the Adapter Configuration Wizard.
7. Sends the message to the queue specified at design time in the Adapter Configuration Wizard.

Figure 10–21 shows a sample BPEL process for this scenario.

Figure 10–21 Asynchronous Request-Response Oracle BPEL PM As Server Sample

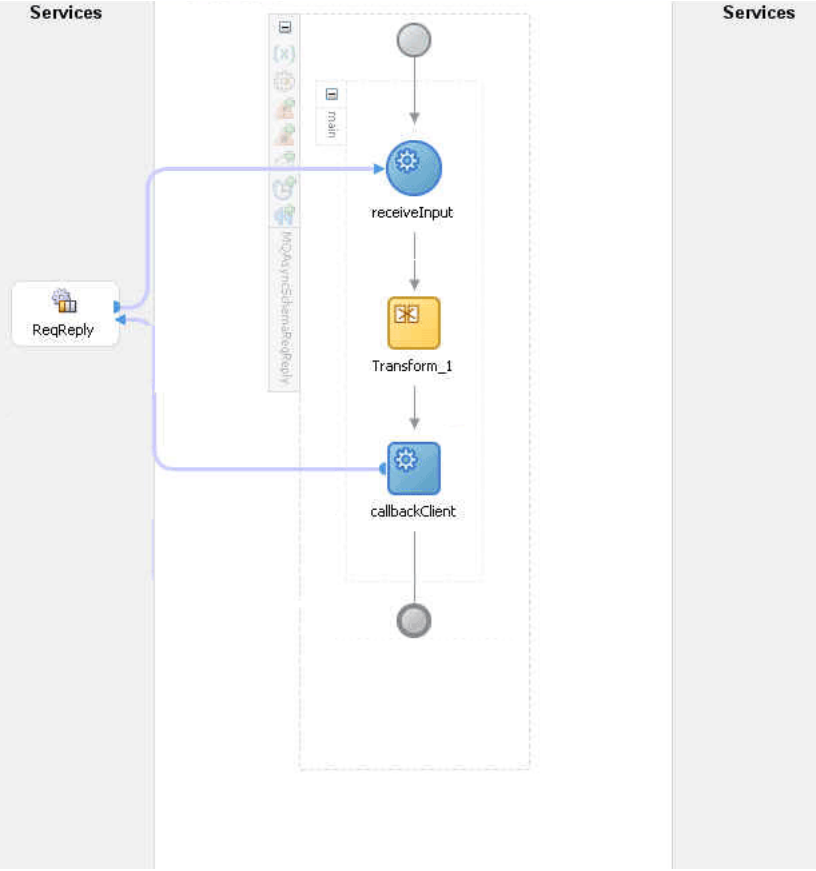
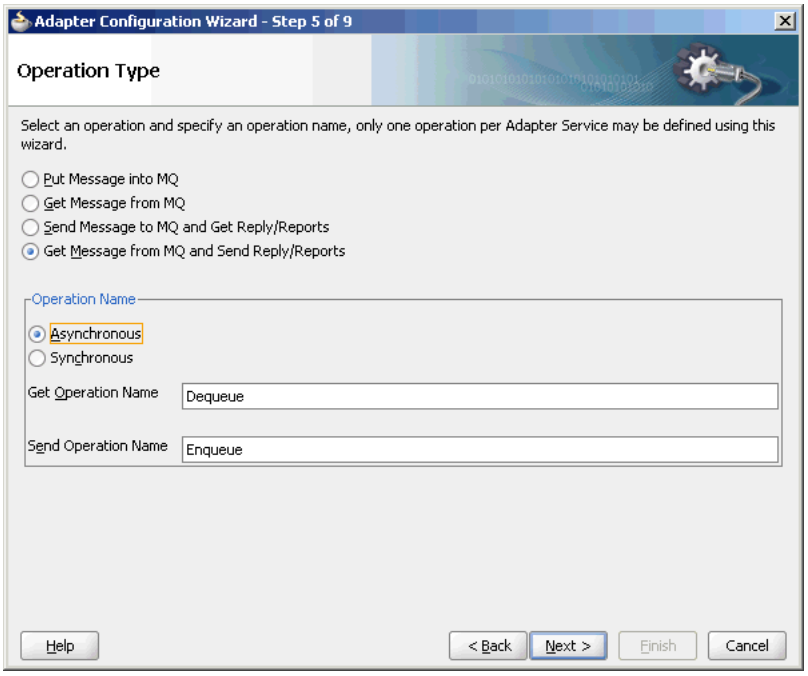


Figure 10–22 displays the operation type that you must select in the Adapter Configuration Wizard.

Figure 10–22 Operation Type Page Selection for Request-Response Asynchronous Interaction



The page that appears after selecting the Get Message from MQ and send Reply/Reports operation type is shown in Figure 10–19. Specify the queue name from which the Oracle MQ Series Adapter dequeues the message in this page.

When you click Next in the Get Message from MQ and send Reply/Reports page, the Response page shown in Figure 10–20 is displayed.

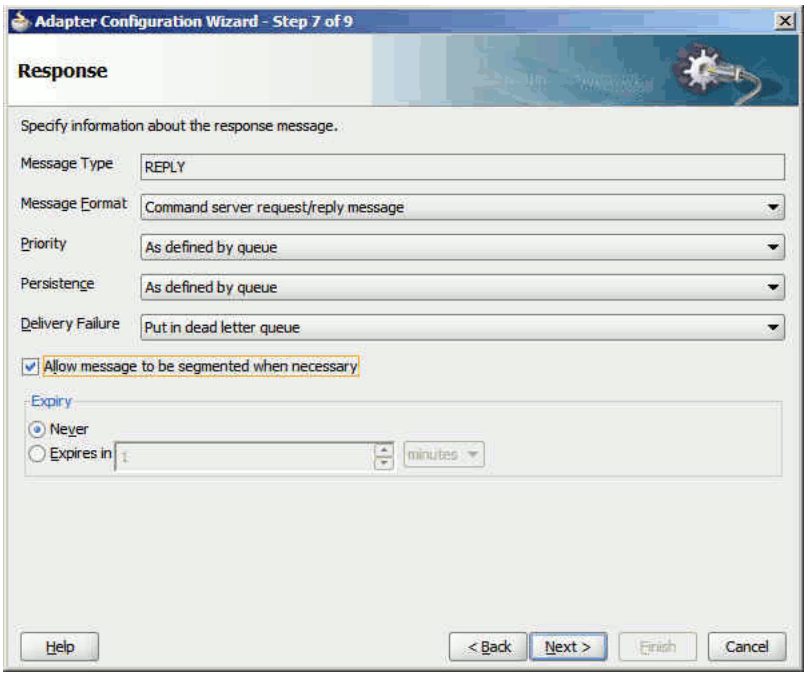
You can specify the following properties in the Response page:

- **Message Type**: The message type of the message to be dequeued. This option affects the return message type.
- **Message Format**: The format of the message.
- **Priority**: The priority of the message.
- **Persistence**: The persistence of the message. You can also specify the persistence of the message to be taken from the default persistence attribute, as defined by the destination queue.
- **Delivery Failure**: If the delivery of the message fails, then either it can be put to a dead letter queue or it can be discarded.
- **Allow Messages to Be Segmented When Necessary**: This is applicable to a message that is big enough for the queue to accommodate. In that case, if you have specified that it has to be segmented, then the single message can be broken into that many bytes the queue can take, which results in mulitple messages.
- **Expiry**: The expiry time of the message.

For more information about these properties, see Section 10.4.2, "Message Properties".

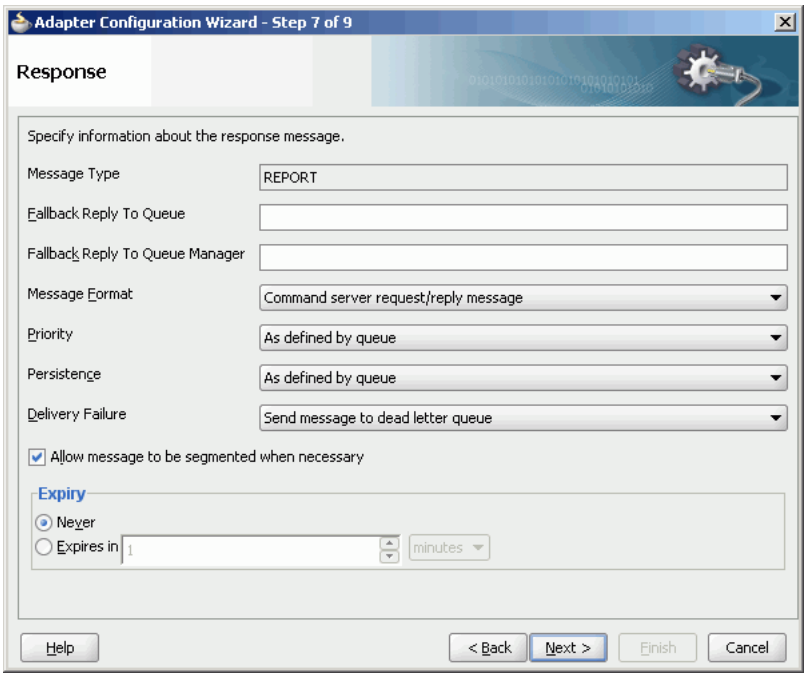
The page that is displayed when you click Next in the Get Message to MQ and Send Reply/Reports page is a Response page (shown in Figure 10–23 and Figure 10–24) but with two different set of options.

Figure 10–23 Response Page (Request Message Type Selected)



The Response page shown in Figure 10–24 is displayed only if you have selected the **Normal** option in Message Type field in the Get Message to MQ and Send Reply/Reports page.

Figure 10–24 Response Page (Normal Message Type Selected)



You can specify the following properties in the Response page:

- (Optional) **Fallback Reply to Queue**: Enter a response fallback queue name. The response message is enqueued to the queue specified with the replyToQueue

property of the request message. However, if the replyToQueue property is not set on the request message, then entering a name here ensures that the process does not fail to enqueue the response.

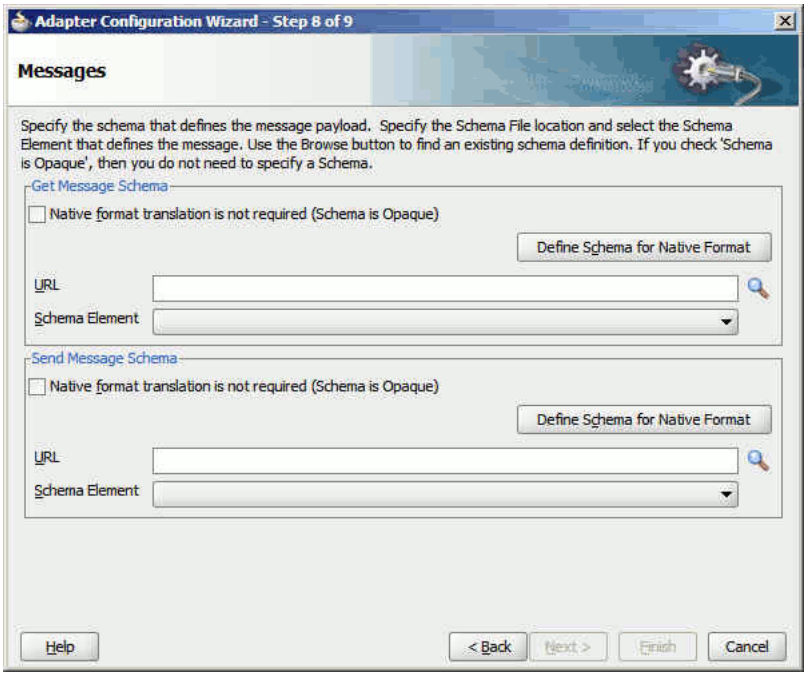
- (Optional) **Fallback Reply to Queue Manager**: Enter a secondary queue name. This name is used when the primary queue manager that was established when you specified the JNDI connection name cannot access the queue name entered in the Queue Name field. This is similar to the functionality described in the Fallback Reply to Queue field.

  To specify the other properties in this Response page, see properties mentioned for Figure 10–23.

When you click **Next** in the Response page, the Messages page shown in Figure 10–25 is displayed. You can perform the following tasks in this page:

- Specify if native format translation is not required.
- Select the XSD schema file for translation.
- Start the Native Format Builder wizard to create an XSD file from file formats such as CSV, fixed-length, DTD, and COBOL Copybook.

Figure 10–25 Messages Page



For more information about the Messages page, see Section 10.4.1.1, "Enqueue Message".

In asynchronous request-reply interaction, you must map the following properties from the inbound message header to the outbound message header:

- `MsgID`: Refers to the message ID.
- `CorrelID`: Refers to the correlation ID of a message.
- `CorrelationScheme`: Refers to a combination of both the msgid and the correlid of the request message.

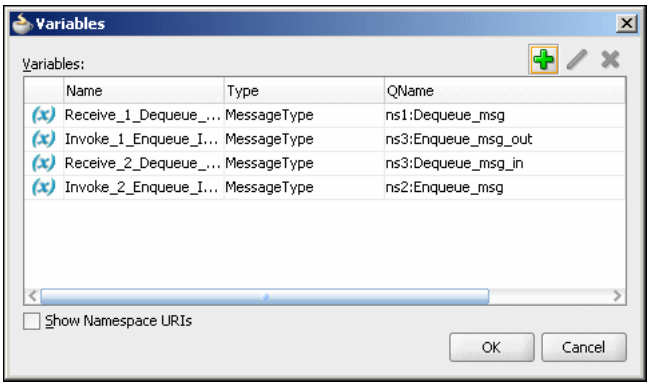
  For more information, see Section 10.4.3, "Correlation Schemas".

- `ReplyToQ` : Refers to the name of the response queue name.
- `ReplyToQueueManager`: Refers to the name of the response queue manager.

You can use the Assign activity to map these properties.

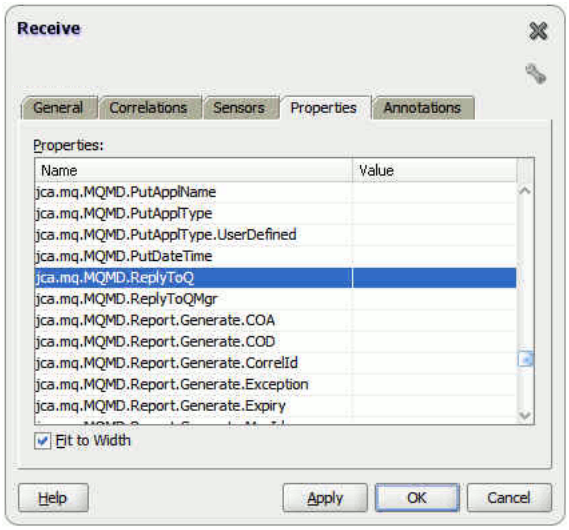
1. Create a BPEL process and double-click to open the BPEL Designer page.
2. In the vertical menu that appears, click the Variables icon that appears as (x) grayed out. The Variables dialog is displayed, as shown in Figure 10–26.

Figure 10–26 The Variables Dialog



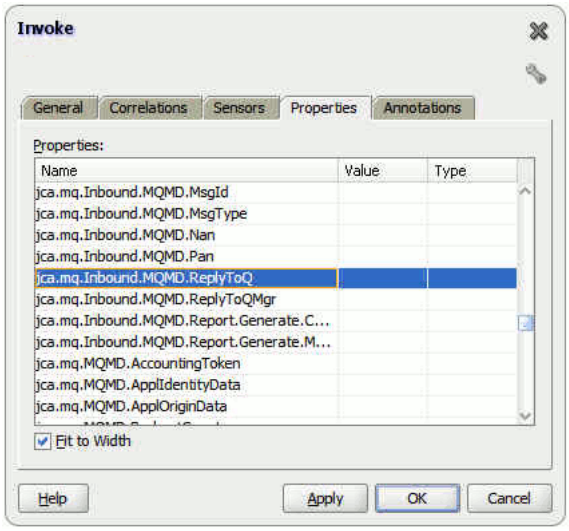
3. Capture the inbound header messages into these variables, as shown in Figure 10–27 and Figure 10–28.

Figure 10–27 The Receive Dialog



4. Assign the variables captured in Step 2 for the Outbound Reply message, as shown in Figure 10–28 and Figure 10–23.

Figure 10–28 The Invoke Dialog



10.4.1.6 Synchronous Request-Response (Mediator As Server)

In this scenario, the Mediator receives a request, processes it, and sends the response synchronously. The Oracle MQ Series Adapter performs the following operations:

1. Dequeues the request message from the queue when the message arrives.
2. Reads and translates the message based on the translation logic defined at design time.
3. Publishes the message as an XML message to Mediator. The Mediator processes the request and sends the response to the Oracle MQ Series Adapter.
4. Receives the response message from the Mediator.
5. Formats the XML content as specified at design time.
6. Sets the properties of the message such as priority, expiry, message type, and persistence. These properties are based on the selections that you made in the Adapter Configuration Wizard.
7. Sends the message to the queue specified at design time in the Adapter Configuration Wizard.

Figure 10–19 displays the operation type that you must select in the Adapter Configuration Wizard.

From this page onwards, all the pages are similar to the pages explained in Section 10.4.1.4, "Synchronous Request-Response (Oracle BPEL PM As Server)".

> **Note:** The asynchronous request-response pattern is not supported for Mediator.

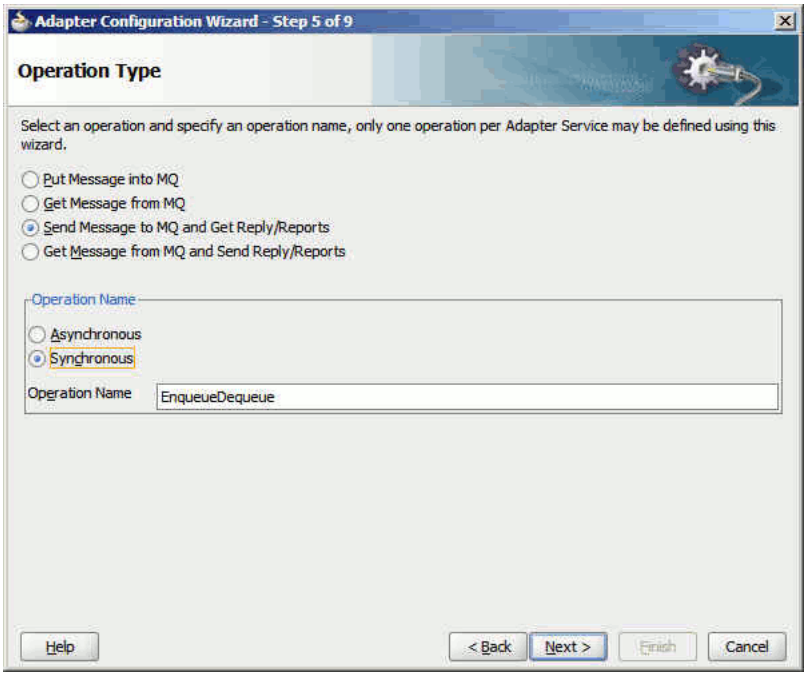
10.4.1.7 Synchronous Request-Response (Oracle BPEL PM As Client)

The Oracle MQ Series Adapter supports the outbound synchronous-solicit-request-response scenario. In this scenario, the adapter enqueues a normal/request message in a queue and expects the report/reply synchronously. The report/reply message arrives in the `ReplyToQueueName` queue of the normal/request message.

Note: Outbound synchronous-solicit-responses must be executed in non-XA modes as the request message does not get enqueued when it is participating in a global transaction.

Figure 10–29 displays the operation type that you must select in the Adapter Configuration Wizard.

Figure 10–29 The Operation Type Dialog



The page that appears after selecting the Send Message to MQ and Get Reply/Reports operation type is shown in Figure 10–13.

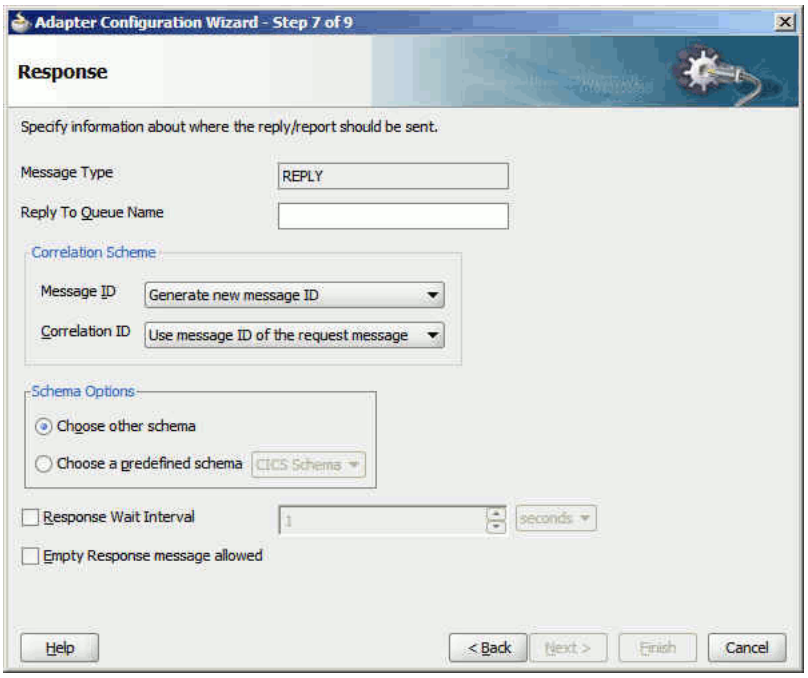
You can specify the following properties in this page:

- **Message Type**: The type of the message. You can either send a normal message or a request message.
- **Queue Name**: The name of the queue to which the Oracle MQ Series Adapter enqueues the message. This is a mandatory field.
- **Queue Manager**: The name of the queue manager to which the queue belongs. This field is optional and is necessary when enqueuing message to a remote queue.
- **Message Format**: The format of the message.
- **Priority**: The priority of the message ranging from 0 (low) to 9 (high).
- **Persistence**: The persistence of the message. You can also specify the persistence of the message to be taken from the default persistence attribute, as defined by the destination queue.
- **Delivery Failure**: If delivery of the message fails, then either it can be put to a dead letter queue or it can be discarded.

- **Allow Messages to Be Segmented When Necessary**: This is applicable to a message that is big enough for the queue to accommodate. In that case, if you have specified that it has to be segmented, then the single message can be broken into that many bytes the queue can take, which results in multiple messages.
- **Expiry**: The expiry time of the message. The message is discarded after the expiry time has elapsed.

Click Next in the Send Message to MQ and Get Reply/Reports page, the Response page, as shown in Figure 10–30, is displayed.

Figure 10–30 The Response Page



For the Synchronous Request-Response scenario, you must also edit the following properties in the Response page:

- **Reply to Queue Name**: The name of reply queue name.
- **Correlation Scheme**: The correlation schema that must be used by the Oracle MQ Series Adapter.

  For more information about correlation schemas, see Section 10.4.3, "Correlation Schemas".
- **Schema Options**: This option enables you to specify the schema for the message to be dequeued.
  - **Choose Other Schema**: This option enables you to choose your schema for the message to be dequeued.
  - **Choose a Predefined Schema**: This option enables you to choose a readymade schema that the adapter provides.
- **Response Wait Interval**: The permitted value for this property is any interval value (>= 0). This is the time in milliseconds during which the adapter waits for the report/reply to arrive in `replyToQueueName`. By default, the value of this property is 0 milliseconds. You can change this value, but the value must be less than that of the timeout interval for the outbound activity. If the report/reply

message does not arrive in the stipulated time, then the adapter throws an exception. This property is not mandatory.

Note: The `ResponseWaitInterval` value must be less than the timeout interval for the outbound activity. If the `ResponseWaitInterval` value exceeds the outbound activity timeout, then the adapter can behave ambiguously.

10.4.1.8 Synchronous Request-Response (Oracle Mediator as Client)

The Oracle MQ Series Adapter also supports the outbound synchronous-solicit-request-response scenario. In this scenario, the adapter enqueues a normal/request message in a queue and expects the report/reply synchronously. The report/reply message arrives in the Reply to Queue Name queue of the normal/request message.

The Synchronous Request-Response scenario for Oracle Mediator as client is same as the Synchronous Request-Response for Oracle BPEL as client. For more information about the Synchronous Request-Response scenario, see Section 10.4.1.7, "Synchronous Request-Response (Oracle BPEL PM As Client)".

10.4.1.9 Asynchronous Request-Response (Oracle Mediator As Client)

In this scenario, Oracle Mediator sends a request message and receives the corresponding response from the Mediator callback handler. Oracle Mediator sends an outbound invocation to send the request. The Oracle MQ Series Adapter performs the following operations:

1. Receives message from Oracle Mediator.
2. Formats the XML content as specified at design time in the Adapter Configuration Wizard.
3. Sets properties and a correlation schema on the request message.
4. Sends the message to the queue specified at design time. The third-party application receives the message, processes it, generates the response, and then enqueues the response message to the `replyTo` queue specified in the request message. The Correlation ID and Message ID of the response message is generated based on the correlation schema specified in the request message.
5. The Oracle MQ Series Adapter dequeues the message from the `replyTo` queue.
6. Sets the properties of the message such as priority, expiry, message type, and persistence. These properties are based on the selections that you made in the Adapter Configuration Wizard.
7. Sends the response to the non-initiating receive activity of the BPEL process. To ensure that response is sent to the correct BPEL instance, correlation schemas are used.

Figure 10–12 displays the operation type that you must select in the Adapter Configuration Wizard.

The page that appears after selecting the Send Message to MQ and Get Reply/Reports operation type is shown in Figure 10–13.

You can specify the following properties in this page:

- **Message Type**: The type of the message. You can either send a normal message or a request message.

- **Queue Name**: The name of the queue to which the Oracle MQ Series Adapter enqueues the message. This is a mandatory field.
- **Message Format**: The format of the message.
- **Queue Manager**: The name of the queue manager to which the queue belongs. This field is optional and is necessary when enqueuing message to a remote queue.
- **Priority**: The priority of the message ranging from 0 (low) to 9 (high).
- **Persistence**: The persistence of the message. You can also specify the persistence of the message to be taken from the default persistence attribute, as defined by the destination queue.
- **Delivery Failure**: If delivery of the message fails, then either it can be put to a dead letter queue or it can be discarded.
- **Allow Messages to be Segmented When Necessary**: This is applicable to a message that is big enough for the queue to accommodate. In that case, if you have specified that it has to be segmented, then the single message can be broken into that many bytes the queue can take, which results in mutliple messages.
- **Expiry**: The expiry time of the message. The message is discarded after the expiry time has elapsed.

For more information about these properties, see Section 10.4.2, "Message Properties" and Section 10.4.5, "Report Messages".

The page that is displayed when you click Next in the Send Message to MQ and Get Reply/Reports page can be a Reports page (shown in Figure 10–14) or a Response page (shown in Figure 10–15).

The Reports page shown in Figure 10–14 is displayed only if you have selected the Get Reports option in the Send Message to MQ and Get Reply/Reports page shown in Figure 10–13.

The Response page shown in Figure 10–15 is displayed, irrespective of whether you select the Request or Normal option. The only difference is that if you select the Request option, then `REPLY` is displayed in the Message Type field of the Response page. On the other hand, if you select the Normal option, then `REPORTS` is displayed in the Message Type field of the Response page.

You can select the following types of reports in Figure 10–14:

- Confirmation on Arrival
- Confirmation on Delivery
- Exception Report
- Expiry Report

For information about these report types, see Section 10.4.5, "Report Messages".

The Response page, shown in Figure 10–15, is displayed when you click Next in the Reports page.

You can specify the following properties in the Response page:

- **Reply to Queue Name**: The name of reply queue name.
- **Correlation Scheme**: The correlation schema that is used by the Oracle MQ Series Adapter.

For information about correlation schemas, see Section 10.4.3, "Correlation Schemas".

- **Schema Options**: This option enables you to specify the schema for the message to be dequeued.
  - **Choose Other Schema**: This option enables you to choose your schema for the message to be dequeued.
  - **Choose a Predefined Schema**: This option enables you to choose a readymade schema that the adapter provides.

Note: For Oracle MQ Series Adapter in an asynchronous outbound request/reply scenario, properties are differentiated by an `(Enqueue)` or `(Dequeue)` label in Oracle Enterprise Manager Console. For example, `QueueName(Enqueue)` is used for putting a message and `QueueName(Dequeue)` is used for dequeuing the reply.

When using Oracle Enterprise Manager Console to edit Oracle MQ Series Adapter properties in this scenario, note the following:

- If you change the `ReplyToQueueName(Enqueue)` property, you must also change the `QueueName(Dequeue)` property to the same value.
- If you change the `MessageId(Dequeue)` property, you must also change the `MessageId(Enqueue)` property to the same value.
- If you change the `CorrelationId(Dequeue)` property, you must also change the `CorrelationId(Enqueue)` property to the same value.

When you click Next in the Response page, a Messages page shown in Figure 10–16 is displayed. This page enables you to select the XSD schema file for translation for request and as response message.

For more information about the Messages page, see Section 10.4.1.1, "Enqueue Message".

10.4.1.10 Outbound Dequeue Scenario

The outbound dequeue scenario dequeues a single message from a queue using the outbound Oracle MQ Series Adapter by using the Get Message from MQ option in the Operation Type page of the Adapter Configuration Wizard. To enable the outbound dequeue option, you must select the Synchronous option, as shown in Figure 10–29.

Click Next in the Send Message to MQ and Get Reply/Reports page, the Response page, as shown in Figure 10–30, is displayed. You must set the following properties in the Response page:

- **QueueName**: This is the name of the MQ Series queue from which the message is dequeued. This property is mandatory.
- **Response Wait Interval**: This is the time (in milliseconds) that the adapter waits if the message is not in the queue. The default value for this property is 0 milliseconds. This property is not mandatory. The permitted value for this property is any integer value (>=0). The value of this property must be less than that of the timeout for outbound activity.

Note: The `ResponseWaitInterval` value must be less than the timeout interval for the outbound activity. If the `ResponseWaitInterval` value exceeds the outbound activity timeout, then the adapter can behave ambiguously.

- **Message Id**: This property sets the message filter option based on the `messageId`. This property is not mandatory. The value provided for this property must be a hexadecimal-encoded value for some `messageId`.
- **Correlation Id**: This property sets the message filter option based on the `correlationId`. This property is not mandatory. The value provided for this property must be a hexadecimal-encoded value for some `correlationId`.

Note: You can filter messages based on the Message Id and Correlation Id property through headers.

10.4.2 Message Properties

The Oracle MQ Series Adapter supports the following message properties:

- Section 10.4.2.1, "Messages Types"
- Section 10.4.2.2, "Message Format"
- Section 10.4.2.3, "Message Expiry"
- Section 10.4.2.4, "Message Priority"
- Section 10.4.2.5, "Message Persistence"

10.4.2.1 Messages Types

The Oracle MQ Series Adapter supports the following four types of messages:

- Normal Message

  A normal message is sent by one program to another program without expecting any response.

- Request Message

  A request message is sent by one program to another program requesting a response.

- Reply Message

  A reply message is sent by a program in response to a request message.

- Report Message

  A report message is sent by a receiving program to a sending program as confirmation of successful or unsuccessful delivery of a message. A report message can be generated for any of the message types, normal message, request message, or reply message.

  For more information about acknowledgment messages supported by the Oracle MQ Series Adapter, see Section 10.4.5, "Report Messages".

10.4.2.2 Message Format

You can specify the format for an outgoing message through the Adapter Configuration Wizard, as shown in Figure 10–8. The following message formats are supported:

- No format name (Default)
- Command server request/reply message
- Type 1 command reply message
- Type 2 command reply message
- Dead letter header
- Event message
- User-defined message in programmable command format
- Message consisting entirely of characters
- Trigger message
- Transmission queue header

10.4.2.3 Message Expiry

You can specify the expiry time for an outgoing message by using the Adapter Configuration Wizard, as shown in Figure 10–8. The queue manager discards the message after the expiry time of a message has elapsed.

If a message has expiration notification set, then a notification is generated when the message is discarded. The notification is sent to the queue specified in the `replyToQueue` parameter. By default, `NEVER` is set for the expiry field.

10.4.2.4 Message Priority

You can specify the priority of an outgoing message through the Adapter Configuration Wizard, as shown in Figure 10–8. A priority can be in the range of 0 (low) to 9 (high). You can also specify the priority of the message to be taken from the default priority attribute, as defined by the destination queue. By default, `AS_Q_DEF` is set as message priority.

10.4.2.5 Message Persistence

You can specify the persistence of an outgoing message through the Adapter Configuration Wizard, as shown in Figure 10–8. If message persistence is not set, then a message is lost when the queue manager restarts or there is a system failure. If you set persistence for a message to `true`, then it means that the message does not get lost even if there is system failure or the queue manager is restarted. You can also specify the persistence of the message to be taken from the default priority attribute, as defined by the destination queue. The Adapter writes persistent messages to log files and queue data files. If a queue manager is restarted after a failure, it recovers these persistent messages from these files.

Note: You can specify all these message properties at run time through message headers. You can use the assign activity to assign values to these properties.

10.4.3 Correlation Schemas

Mapping a response to a request in a request-reply interaction requires correlation. Each MQ Series request message contains a message ID and a correlation ID. When an application receives a request message from Oracle BPEL PM, it checks for the correlation schema defined for the response message. Based on the correlation schema, the application generates the message ID and correlation ID of the response message.

The response page of the Adapter Configuration Wizard shown in Figure 10–15 enables you to specify the correlation schema for the response message.

The Message ID box shown in Figure 10–15 provides the following options for the message ID of the response message:

- Generate a new message ID for the response message.
- Use the message ID of the request message.

Similarly, the Correlation ID box shown in Figure 10–15 provides the following options for the correlation ID of the response message:

- Use the message ID of the request message
- Use the correlation ID of the request message

10.4.4 Distribution List Support

The Oracle MQ Series Adapter enables you to enqueue a message to multiple queues.

When you select the Put Message Into MQ option in the Operation Type page and multiple queues, then the `DistributionList` parameter is automatically added to the JCA file.

10.4.5 Report Messages

The Oracle MQ Series Adapter enables you to set various types of acknowledgment messages on an outgoing message. These acknowledgment messages are known as report messages. A report message is generated, only if the criteria for generating that report message is met. When enqueuing a message on a queue, you can request for more than one type of report message. When you request for a report message, you must specify the queue name to which the report message is sent. This queue is known as `replyTo` queue. A report message can be generated by a queue manager, a message channel, or an application.

The Oracle MQ Series Adapter supports the following message reports:

- Confirmation on Arrival

  The Confirmation on Arrival (COA) message indicates that the message has been delivered to the target queue manager. A COA message is generated by the queue manager. This message report can be selected in the Reports page of the Adapter Configuration page shown in Figure 10–14.

- Confirmation on Delivery

  A Confirmation on Delivery (COD) message indicates that the message has been retrieved by the receiving application. A COD message is generated by the queue manager. This message report can be selected in the Reports page shown in Figure 10–14.

- Exception Report

An exception report is generated when a message cannot be delivered to the specified destination queue. Exception reports are generated by the message channel. This message report can be selected in the Reports page of the Adapter Configuration page shown in Figure 10–14.

- Expiry Report

  An expiry report indicates that the message was discarded because the expiry time specified for the message elapsed before the message was retrieved. An expiry report is generated by a queue manager. This message report can be selected in the Reports page of the Adapter Configuration page shown in Figure 10–14.

- Positive Action Notification

  A Positive Action Notification (PAN) indicates that a request has been successfully processed. It means that the action requested in the message has been performed successfully. This type of report is generated by the application.

- Negative Action Notification

  A Negative Action Notification (NAN) indicates that a request has not been successfully serviced. It means that the action requested in the message has not been performed successfully. This type of report is generated by the application.

You can specify whether all these report messages except PAN and NAN should contain the complete original message, a part of the original message, or no part of the original message. You can select any of the following options in the Adapter Configuration Wizard:

- No data from the original message
- The first 100 bytes of data in the original message
- The entire original message

10.4.6 Message Delivery Failure Options

The Message Delivery Failure options are supported only for remote queues and not for normal queues. The Oracle MQ Series Adapter enables you to specify the action that should be taken in case a message could not be delivered to the destination queue. You can specify any of:

- Place message on a dead letter queue

  This is the default action. A message is placed on a dead-letter queue if it cannot be delivered to the destination queue. A report message is generated if requested by the sender.

- Discard message

  This indicates that the message should be discarded if it cannot be delivered to the destination queue. A report message is generated if requested by the sender.

You can specify these options by selecting the Put Message To MQ option in the Adapter Configuration Wizard.

10.4.7 Message Segmentation

The Oracle MQ Series Adapter supports message segmentation for both inbound and outbound interactions. Segmentation is required when the size of a message is greater than the message size allowed for a queue. A physical message is divided into two or more logical messages. All logical messages have the same group ID and a sequence number, and an offset.

In the inbound interaction, the segmentation is inherently supported by the Oracle MQ Series Adapter. The Oracle MQ Series Adapter dequeues all logical messages in the order of sequence number and then publishes the single message as XML to Oracle BPEL PM or Mediator.

The Allow Messages to Be Segmented When Necessary option enables you to segment messages for outbound interactions. This option appears in the Response page of the Adapter Configuration Wizard.

The message is segmented based on whether the size of the message is larger than the maximum limit set on the queue.

10.4.8 Integration with CICS

The Oracle MQ Series Adapter provides support for sending and receiving messages from the CICS server. In the inbound direction, an inbound message from the CICS server is dequeued in the same way as a normal message. In the outbound direction, the message should be in the CICS format. A sample schema file for the outbound CICS message format is shown in the following example:

```
<?xml version="1.0" ?><schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://xmlns.oracle.com/pcbpel/nxsd/cics_mqcih"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"

        xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
        nxsd:version="NXSD"

        nxsd:encoding="UTF8"
        nxsd:stream="bytes"
        nxsd:byteOrder="bigEndian"

        xmlns:nxsd_extn="http://xmlns.oracle.com/pcbpel/nxsd/extensions"

<element name="MSGForMQCICSBridge">
    <complexType>
      <sequence>
        <element name="MQCIH">
          <complexType>
            <sequence>
              <!--
              MQCHAR4   StrucId;
              Structure identifier
              -->
              <element name="StrucId" type="string"
               nxsd:style="fixedLength" nxsd:length="4" nxsd:padStyle="tail"/>

              <!--
              MQLONG    Version;
              Structure version number 1 or 2
              -->
              <element name="Version" type="string"
               nxsd:style="integer" nxsd_extn:octet="4"
               nxsd_extn:align="0" nxsd_extn:sign="unticked" />
              <!--
              MQLONG    StrucLength;
              Length of MQCIH structure V1=164 V2=180
              -->
              <element name="StrucLength" type="string"
               nxsd:style="integer" nxsd_extn:octet="4"
```

```
 nxsd_extn:align="0" nxsd_extn:sign="unticked" />

<!--
MQLONG    Encoding;
Reserved
-->
<element name="Encoding" type="string"
 nxsd:style="integer" nxsd_extn:octet="4"
 nxsd_extn:align="0" nxsd_extn:sign="unticked" />

<!--
MQLONG    CodedCharSetId;
Reserved              -->
<element name="CodedCharSetId" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />

<!--
MQCHAR8   Format;
MQ Format name
-->
<element name="Format" type="string"
nxsd:style="fixedLength" nxsd:length="8" />

<!--
MQLONG    Flags;
Reserved
-->
<element name="Flags" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"

 nxsd_extn:sign="unticked" />

<!--
MQLONG   ReturnCode;
Return code from bridge
-->
<element name="ReturnCode" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />

<!--
MQLONG   CompCode;
MQ completion code or CICS EIBRESP
-->
<element name="CompCode" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />

<!--
MQLONG   Reason;
MQ reason or feedback code, or CICS EIBRESP2
-->
<element name="Reason" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />

<!--
MQLONG   UOWControl;
Unit-of-work control
```

```
-->
<element name="UOWControl" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />

<!--
MQLONG   GetWaitInterval;
Wait interval for MQGET call issued by bridge
-->
<element name="GetWaitInterval" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="ticked" />

<!--
MQLONG   LinkType;
Link type
-->
<element name="LinkType" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
nxsd_extn:sign="unticked" />

<!--
MQLONG   OutputDataLength;
Output commarea data length
-->
<element name="OutputDataLength" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="ticked" />

<!--
MQLONG   FacilityKeepTime;
Bridge facility release time
-->
<element name="FacilityKeepTime" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />

<!--
MQLONG   ADSDescriptor;
Send/receive ADS descriptor
-->
<element name="ADSDescriptor" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />

<!--
MQLONG   ConversationalTask;
Whether task can be conversational
-->
<element name="ConversationalTask" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />
             <!--
MQLONG   TaskEndStatus;
Status at end of task
-->
<element name="TaskEndStatus" type="string"
 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
 nxsd_extn:sign="unticked" />
```

```
<!--
MQBYTE   Facility[8];
BVT token value. Initialise as required.
-->
<element name="Facility" type="string"
 nxsd:style="integer" nxsd_extn:octet="8" nxsd_extn:align="0"
 nxsd _extn:sign="unticked" />

<!--
MQCHAR4  Function;
MQ call name or CICS EIBFN function name
-->
<element name="Function" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  AbendCode;
Abend code
-->
<element name="AbendCode" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR8  Authenticator;
Password or passticket
-->
<element name="Authenticator" type="string"
nxsd:style="fixedLength" nxsd:length="8" />

<!--
MQCHAR8  Reserved1;
Reserved
-->
<element name="Reserved1" type="string"
 nxsd:style="fixedLength" nxsd:length="8" />

<!--
MQCHAR8  ReplyToFormat;
MQ format name of reply message
-->
<element name="ReplyToFormat" type="string"
 nxsd:style="fixedLength" nxsd:length="8" />

<!--
MQCHAR4  RemoteSysId;
Remote sysid to use
-->
<element name="RemoteSysId" type="string"
nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  RemoteTransId;
Remote transid to attach
-->
<element name="RemoteTransId" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  TransactionId;
```

```
Transaction to attach
-->
<element name="TransactionId" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  FacilityLike;
Terminal emulated attributes
-->
<element name="FacilityLike" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  AttentionId;
AID key
-->
<element name="AttentionId" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  StartCode;
Transaction start code
-->
<element name="StartCode" type="string"
nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  CancelCode;
Abend transaction code
-->
<element name="CancelCode" type="string"
 nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR4  NextTransactionId;
Next transaction to attach
-->
<element name="NextTransactionId" type="string"
nxsd:style="fixedLength" nxsd:length="4" />

<!--
MQCHAR8  Reserved2;
Reserved
-->
<element name="Reserved2" type="string"
nxsd:style="fixedLength" nxsd:length="8" />
<!--
MQCHAR8  Reserved3;
Reserved
-->
<element name="Reserved3" type="string"
nxsd:style="fixedLength" nxsd:length="8" />

<!--
MQLONG   CursorPosition;
Cursor position
-->
<element name="CursorPosition" type="string"
nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
nxsd_extn:sign="unticked" />
```

```
                <!--
                MQLONG   ErrorOffset;
                Error offset
                -->
                <element name="ErrorOffset" type="string"
                 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
                 nxsd_extn:sign="unticked" />

                <!--
                MQLONG   InputItem;
                Input item
                -->
                <element name="InputItem" type="string"
                 nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
                 nxsd_extn:sign="unticked" />

                <!--
                MQLONG   Reserved4;
                Reserved
                -->
                <element name="Reserved4" type="string"
                nxsd:style="integer" nxsd_extn:octet="4" nxsd_extn:align="0"
                nxsd_extn:sign="unticked" />
              </sequence>
            </complexType>
          </element>

          <!--
          Application data
          -->
          <element name="ApplicationData" type="string"
                                       fixed="Nothing" />

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

10.4.9 Supported Encodings

By default, Oracle MQ Series Adapter supports a list of encodings. It displays a list of MQ Series message encodings and Java encoding, and also the mapping between the MQ Series message encoding and Java encoding. The list of supported encodings for Oracle MQ Series Adapter is as follows:

- ibm037
- ibm437
- ibm500
- ibm819
- Unicode
- UTF8
- ibm273
- ibm277

- ibm278
- ibm280
- ibm284
- ibm285
- ibm297
- ibm420
- ibm424
- ibm737
- ibm775
- ibm813
- ibm838
- ibm850
- ibm852
- ibm855
- ibm856
- ibm857
- ibm860
- ibm861
- ibm862
- ibm863
- ibm864
- ibm866
- ibm868
- ibm869
- ibm870
- ibm871
- ibm874
- ibm875
- ibm912
- ibm913
- ibm914
- ibm915
- ibm916
- ibm918
- ibm920
- ibm921
- ibm922

- ibm930
- SJIS
- ibm933
- ibm935
- ibm937
- ibm939
- ibm942
- ibm948
- ibm949
- ibm950
- EUCJIS
- ibm964
- ibm970
- ibm1006
- ibm1025
- ibm1026
- ibm1089
- ibm1097
- ibm1098
- ibm1112
- ibm1122
- ibm1123
- ibm1124
- Cp1250
- Cp1251
- Cp1252
- Cp1253
- Cp1254
- Cp1255
- Cp1256
- Cp1257
- Cp1258
- ibm1381
- ibm1383
- JIS
- KSC5601
- ibm33722813

- GB18030

You can add support for the other standard Java encodings that are not provided in above list, as follows:

1. Extract the `MQSeriesAdapter.jar` file from the `MQSeriesAdapter.rar` file.
2. Extract the `mq.properties` file from the `MQSeriesAdapter.jar` file.
3. Add the entry in the `mq.properties` file. For each new encoding, you must add two lines (properties) to the `mq.properties` file. One line for the MQ Series encoding to the corresponding Java encoding and other line for the Java encoding to the corresponding MQ Series encoding.

   For example, to add support for the following `ibm037` Java encoding:ibm037 (Java encoding)<->37 (MQ Series message encoding), you must add the following two lines to the `mq.properties` file:

   ```
   oracle.tip.adapter.mq.encoding.37=ibm037
   ```

   ```
   oracle.tip.adapter.mq.encoding.ibm037=37
   ```

10.5 Configuring the Oracle MQ Series Adapter

The prerequisites for using the Oracle MQ Series Adapter are:

- IBM WebSphere MQ server should be installed and running.
- A queue manager and a server connection channel should be created.

> **Note:** You must create queues based on the requirement of the application.

To configure the Oracle MQ Series Adapter, perform the following:

- Adding com.ibm.mq.jar to the Oracle MQ Series Adapter Classpath
- Adding JNDI Entry
- Enabling Binding Mode for Connections

10.5.1 Adding com.ibm.mq.jar to the Oracle MQ Series Adapter Classpath

The steps in this section should be performed once, before using the Oracle MQ Series Adapter. To add the `com.ibm.mq.jar` property to the classpath for the Oracle MQ Series Adapter, copy the `com.ibm.mq.jar` file to the *<DOMAIN_HOME>/lib* folder.

10.5.2 Adding jar Files to the Oracle MQ Series Adapter Classpath: MQ Series 6 and 7

The steps in this section should be performed once, before using the Oracle MQ Series Adapter.

To add correct jar properties to the classpath for the Oracle MQ Series 6 Adapter, copy the following jars to *<DOMAIN_HOME>/lib* folder

- `com.ibm.mq.jar`
- `com.ibm.mqetclient.jar` (for use with XA)

To add correct jar properties to the classpath for the Oracle MQ Series 7 Adapter, copy the following jars to *<DOMAIN_HOME>/lib* folder

- com.ibm.mq.commonservices.jar
- com.ibm.mq.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.headers.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mqetclient.jar (for use with XA)

In addition, if you are using the Oracle MQ Series 7 Adapter , the new Sharing Conversation property of the Server Connection Channel has to be to set to zero.

10.5.3 Adding JNDI Entry

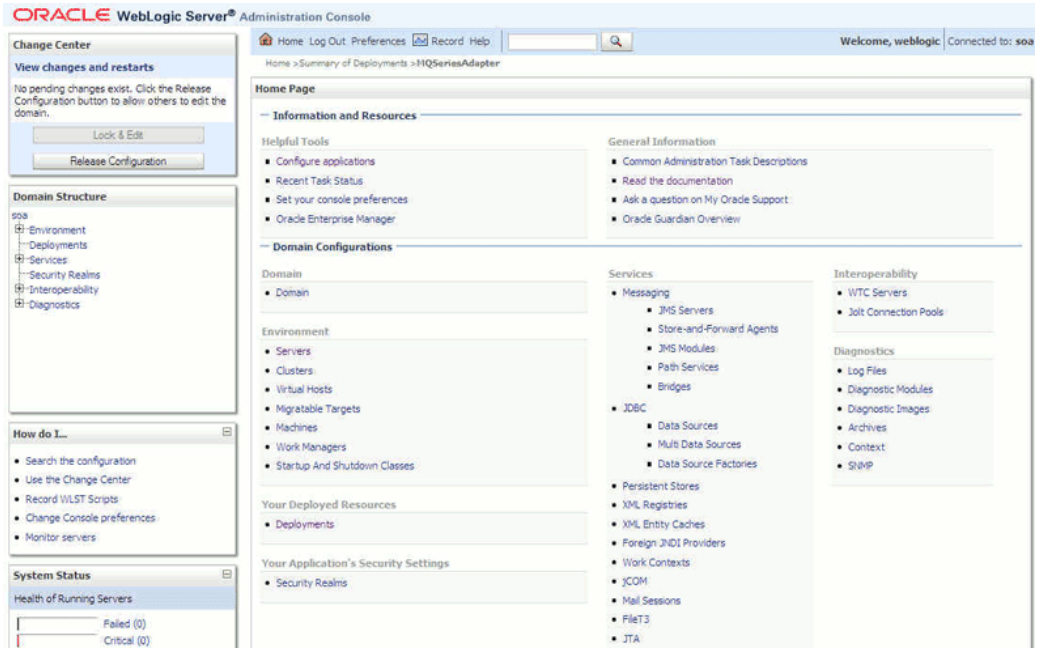
You can add a new jndi entry in the Oracle WebLogic Server Administration Console by following these steps:

1. Log in to the following URL using the username/password to open the Oracle WebLogic Server Administration Console:

   http://<localhost>:port/console

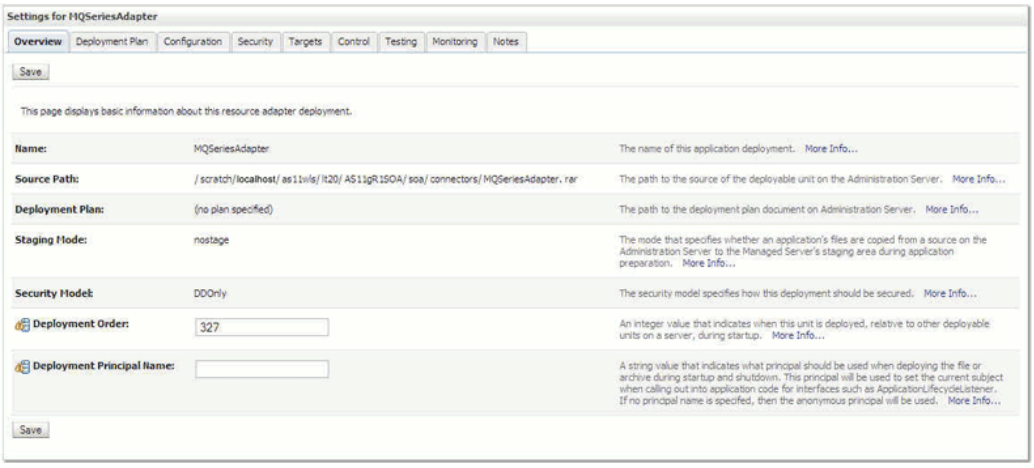
   The Home page is displayed, as shown in Figure 10–31.

Figure 10–31 Oracle WebLogic Administration Console Home Page



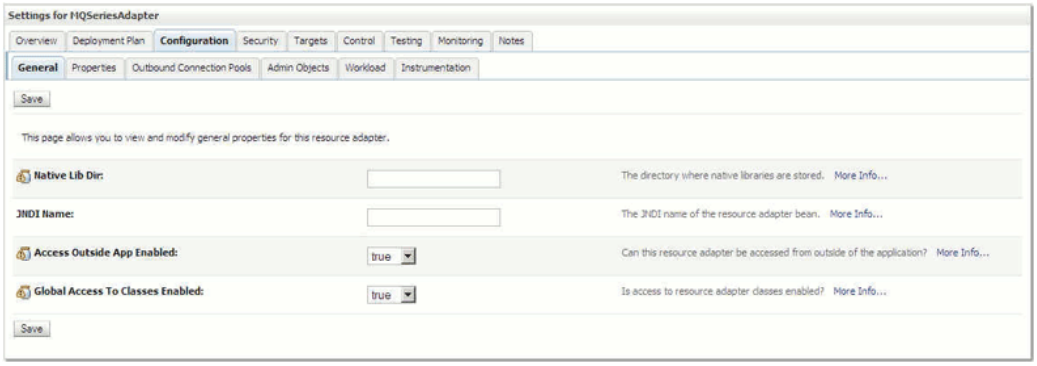
2. Under Domain Structure, in the left pane, click **Deployments**. The Summary of Deployments page is displayed.
3. Click **MQSeriesAdapter**. The Settings of MQSeriesAdapter page is displayed, as shown in Figure 10–32.

Figure 10–32 Settings of MQSeriesAdapter Page



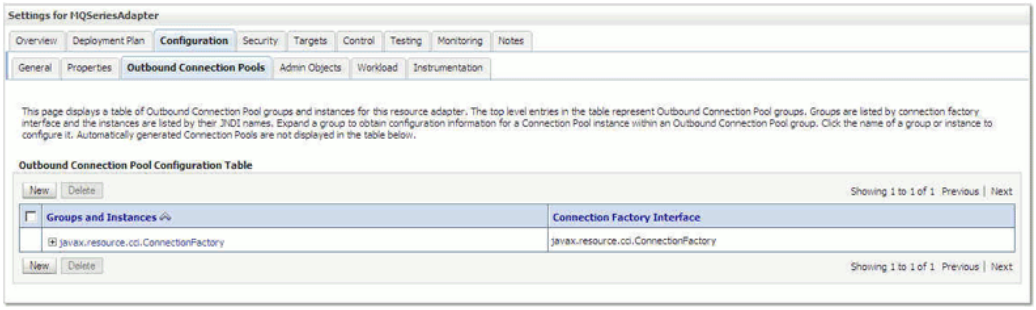
4. Click the **Configuration** tab. The Configuration submenu options are displayed, as shown in Figure 10–33.

Figure 10–33 Settings of MQSeriesAdapter Page - Configuration Submenu Options



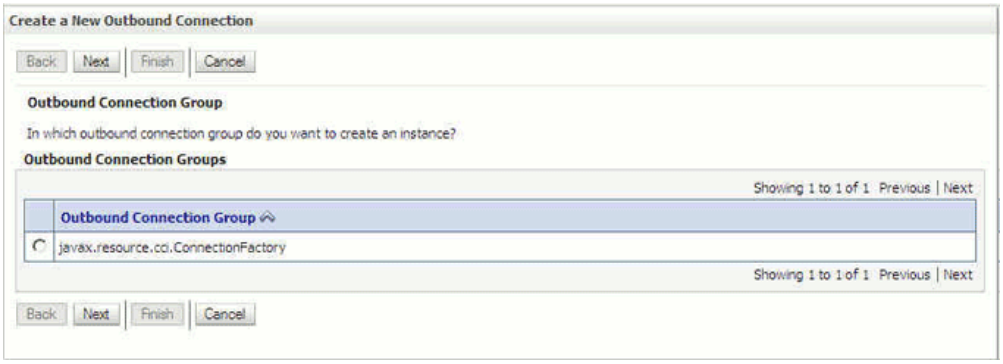
5. Click **Outbound Connection Pools**. The Outbound Connection Pool Configuration Table is displayed, as shown in Figure 10–34.

Figure 10–34 Outbound Connection Pool Configuration Table



6. Click **New**. The Create a New Outbound Connection page is displayed, as shown in Figure 10–35.

Figure 10–35 Create a New Outbound Connection Page



7. Select the **javax.resource.cci.ConnectionFactory** option, and click **Next**.
8. Enter a value in the JNDI Name field, for example `eis/MQ/MQAdapter`, as shown in Figure 10–36.

Figure 10–36 Create a New Outbound Connection Page - JNDI Name

9. Click **Finish**. The Save Deployment Plan Assistant page is displayed.
10. Click **OK**. You have successfully created a JNDI name.

10.5.4 Enabling Binding Mode for Connections

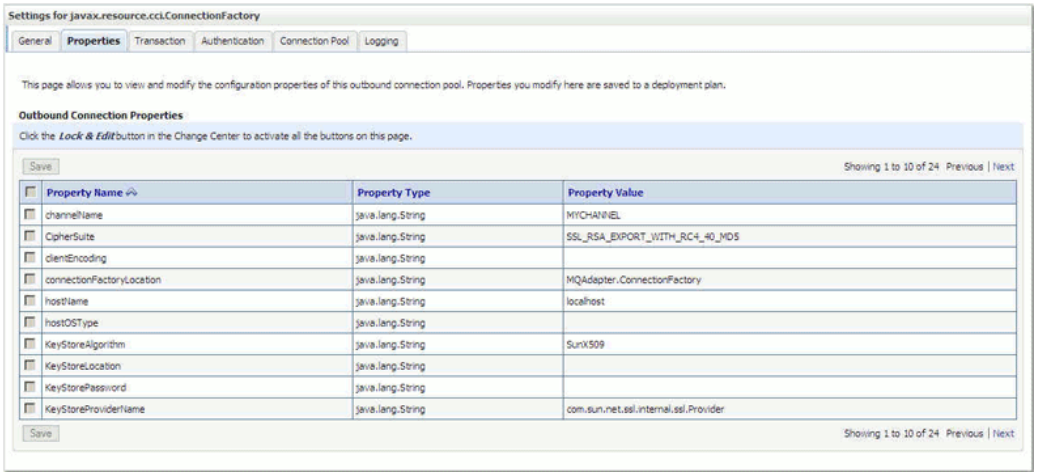
You can enable binding mode for connections for the Oracle MQ Series Adapter by modifying a few properties in the Oracle WebLogic Server Administration Console:

To enable binding mode, perform the following steps:

1. Log in to the Oracle WebLogic Server Administration Console using your password credentials.
2. Under Domain Structure, in the left pane, click **Deployments**. The Summary of Deployments page is displayed.
3. Click **MQSeriesAdapter**. The Settings of MQSeriesAdapter page is displayed.
4. Click the **Configuration** tab. The Configuration submenu options are displayed.
5. Click **Outbound Connection Pools**. The Outbound Connection Pool Configuration Table is displayed.
6. Click the + icon next to javax.resource.cci.ConnectionFactory. A list of JNDIs are displayed.

7. Select, **eis/MQ/MQAdapter**, the JNDI that you created in the Section 10.5.3, "Adding JNDI Entry". The Outbound Connection Properties page is displayed, as shown in Figure 10–37, with a list of 24 properties.

Figure 10–37 Outbound Connection Properties Page



8. Set the following parameters as mentioned below:
   - **hostName**: This value should always be blank.
   - **portNumber**: This value should contain some unused port numbers. For example, 44888.
   - **channelName**: This value should always be blank.
   - **queueManagerName**: This value is a valid queue manager name.

You have enabled the binding mode for connections for the Oracle MQ Series Adapter.

10.6 Oracle MQ Series Adapter Use Cases

This section contains the following topics:

- Section 10.6.1, "Dequeue Enqueue"
- Section 10.6.2, "Inbound Synchronous Request-Reply"
- Section 10.6.3, "Inbound-Outbound Synchronous Request-Reply"
- Section 10.6.4, "Asynchronous-Request-Reply"
- Section 10.6.5, "Outbound Dequeue"
- Section 10.6.6, "Using RFH2 Headers"
- Section 10.6.7, "Configuring a Backout Queue"

10.6.1 Dequeue Enqueue

This use case is the end-to-end demonstration of how MQ Adapter dequeues a message and enqueues the same message after transformation from the MQ Series queue. This section contains the following topics:

- Section 10.6.1.1, "Prerequisites"
- Section 10.6.1.2, "Designing the SOA Composite"

- Section 10.6.1.3, "Creating an Inbound Adapter Service"
- Section 10.6.1.4, "Creating an Outbound Adapter Service"
- Section 10.6.1.5, "Wiring Services and Activities"
- Section 10.6.1.6, "Deploying with JDeveloper"
- Section 10.6.1.7, "Monitoring Using the Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console)"

10.6.1.1 Prerequisites

To perform the dequeue enqueue use case, you need the following files from the `artifacts.zip` file contained in the `Adapters-101MQAdapterDequeueEnqueue` sample:

- `artifacts/schemas/address-csv.xsd`
- `artifacts/schemas/address-fixedLength.xsd`
- `artifacts/input/data.txt`

You can obtain the `Adapters-101MQAdapterDequeueEnqueue` sample by accessing the Oracle SOA Sample Code site.

You must also create the following queues:

- `test_in`
- `test_out`

10.6.1.2 Designing the SOA Composite

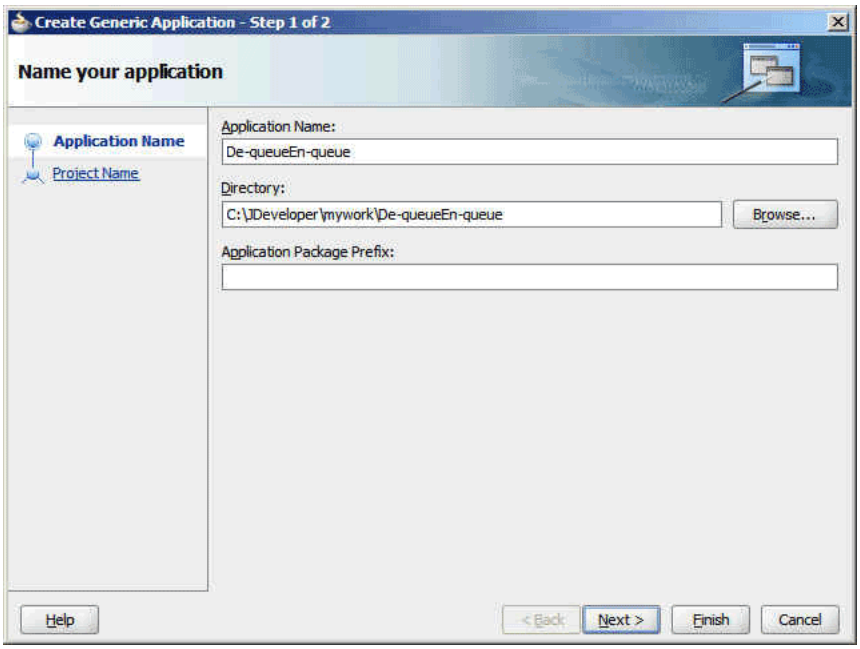
You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.

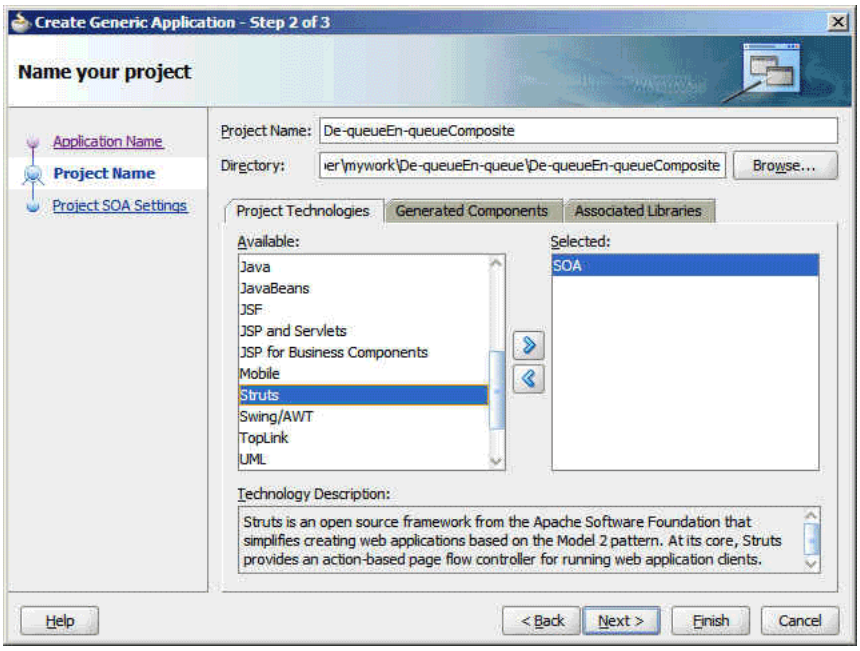
2. Expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.
4. In the Name Your Application screen, enter `De-queueEn-queue` in the **Application Name** field, as shown in Figure 10–38, and then click **Next**. The Name Your Project screen is displayed.

Figure 10–38 The Name Your Application Page



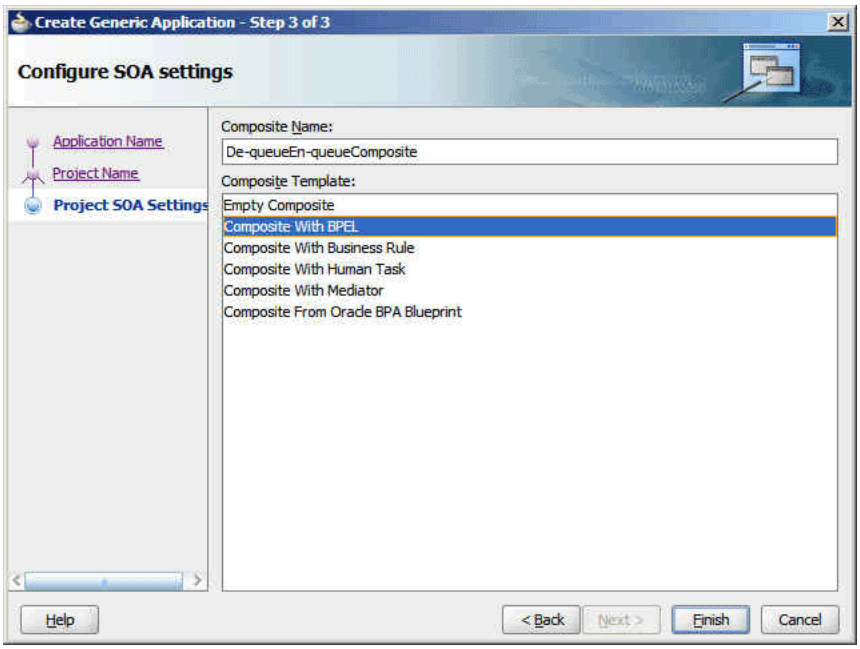
5. In the **Project Name** field, enter `De-queueEn-queueComposite` and from the **Available** list, select **SOA** and click the right-arrow button, as shown in Figure 10–39.

Figure 10–39 The Name Your Project Page



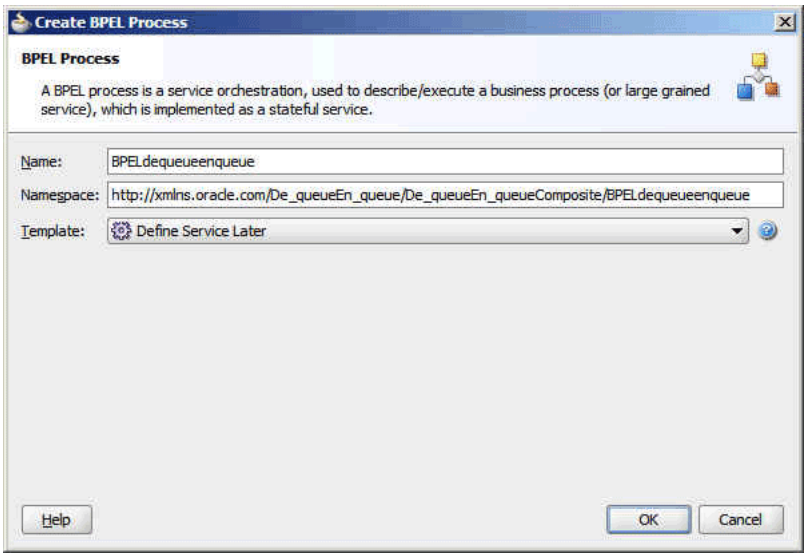
6. Click **Next**. The Configure SOA Settings screen is displayed.
7. In the Composite Template list, select **Composite With BPEL**, as shown in Figure 10–40, and then click **Finish**. The Create BPEL Process dialog is displayed.

Figure 10–40 The Configure SOA Settings Page



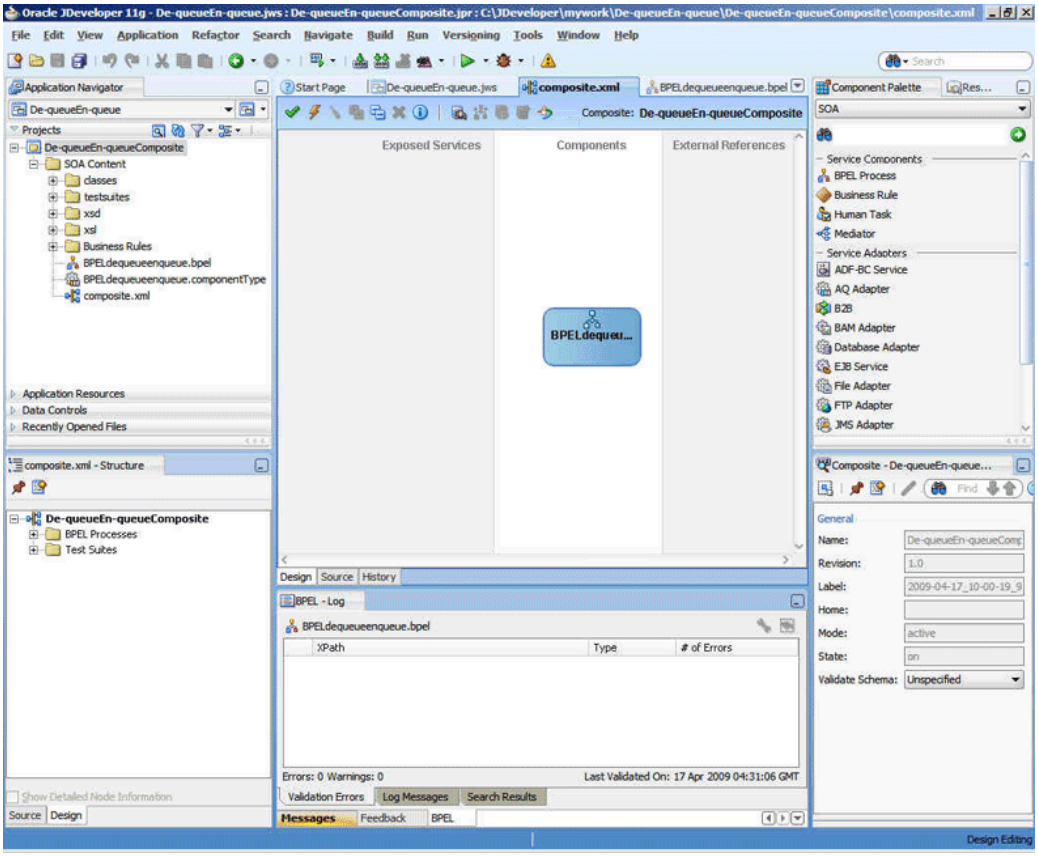
8. Enter `BPELdequeueenqueue` in the **Name** field, and select **Define Service Later** from the Template box, as shown in Figure 10–41.

Figure 10–41 The Create BPEL Process Dialog



9. Click **OK**. The De-queueEn-queue application and the De-queueEn-queue project appears in the design area, as shown in Figure 10–42.

Figure 10–42 The JDeveloper - Composite.xml



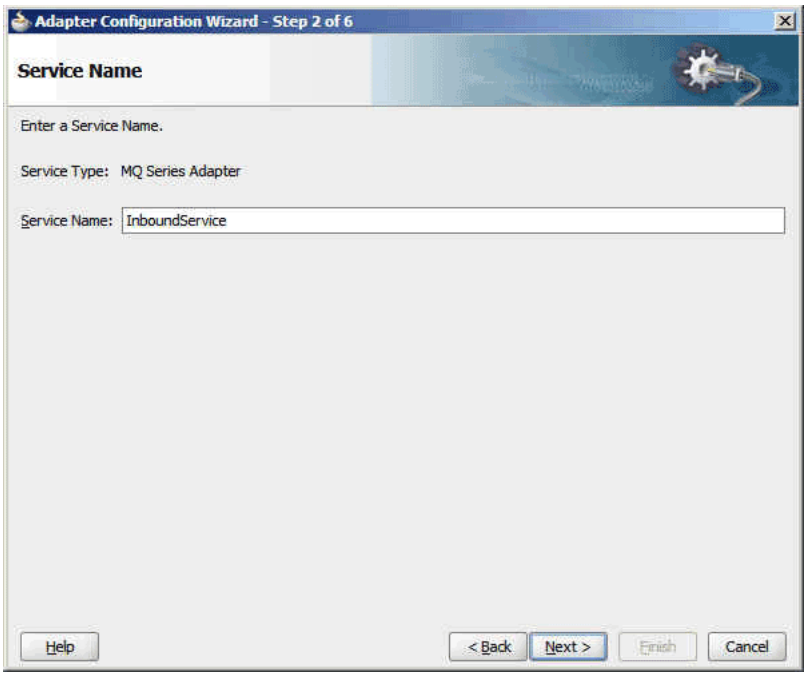
10. Copy the `address-csv.xsd` and `address-fixedLength.xsd` files to the xsd folder in your project (see Section 10.6.1.1, "Prerequisites" for the location of these files).

10.6.1.3 Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message from a queue:

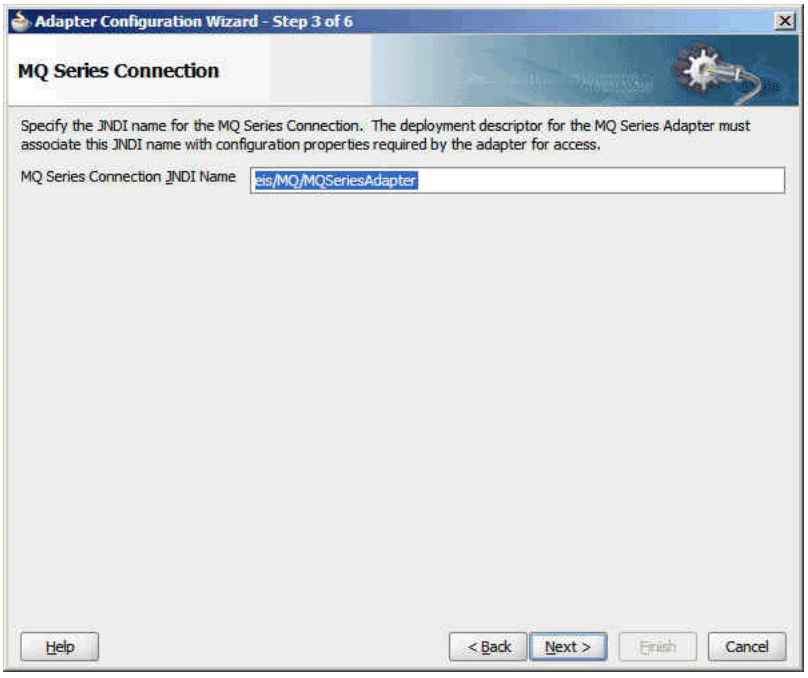
1. Drag and drop **MQ Adapter** from the Component Palette into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `InboundService` in the **Service Name** field, as shown in Figure 10–43, and click **OK**. The MQ Series Connection page is displayed.

Figure 10–43 The Service Name Page



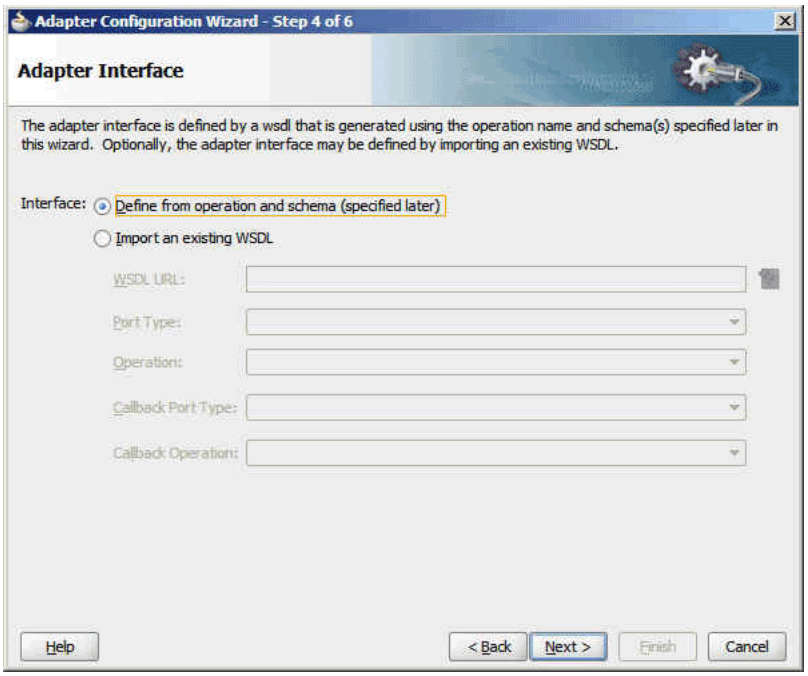
4. Accept the default JNDI name for the MQ Series connection, as shown in Figure 10–44, and click **Next**. The Adapter Interface page is displayed.

Figure 10–44 The MQ Series Connection Page



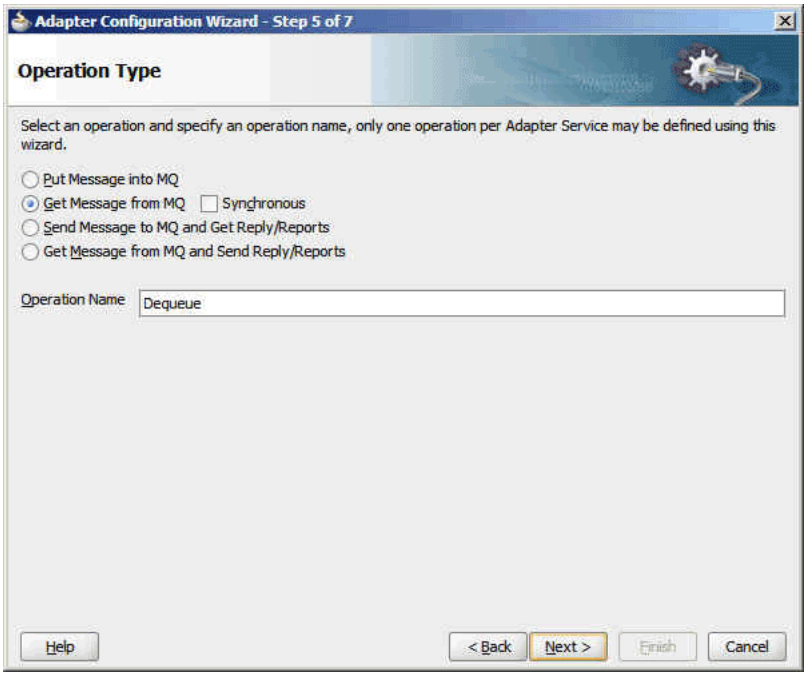
5. Select **Define from operation and schema (specified later)**, as shown in Figure 10–45, and click **Next**. The Operation Type page is displayed.

Figure 10–45 The Adapter Interface Page



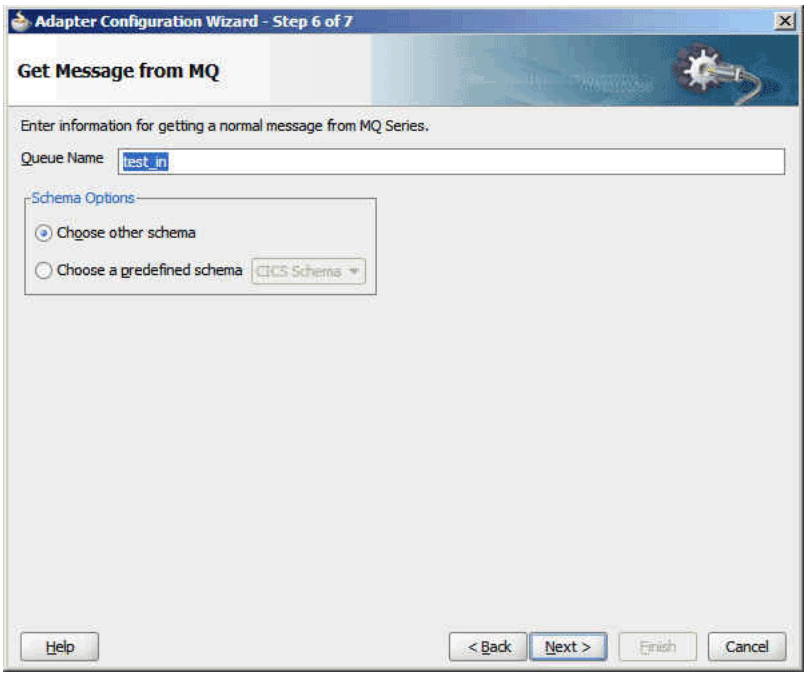
6. Select **Get Message from MQ**, as shown in Figure 10–46, and click **Next**. The Get Message from MQ page is displayed.

Figure 10–46 The Operation Type Page



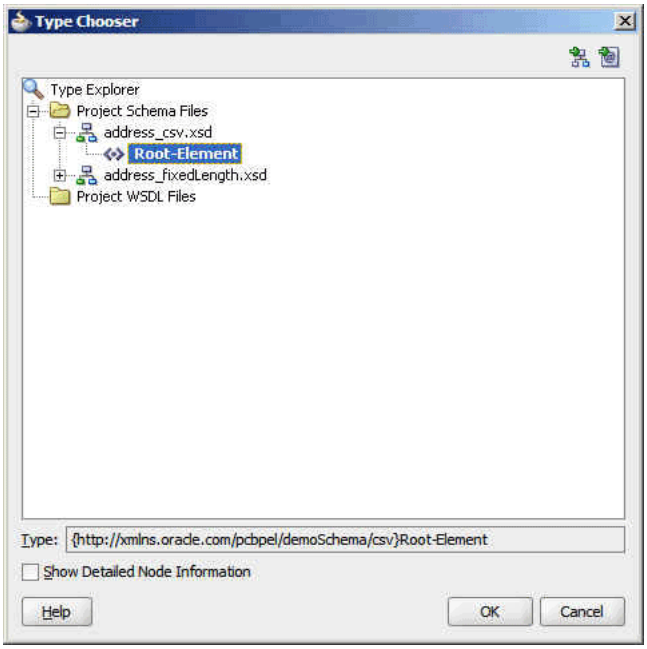
7. Enter `test_in` in the **Queue Name** field, as shown in Figure 10–47, and click **Next**. The Messages page is displayed.

Figure 10–47 The Get Message From MQ Page



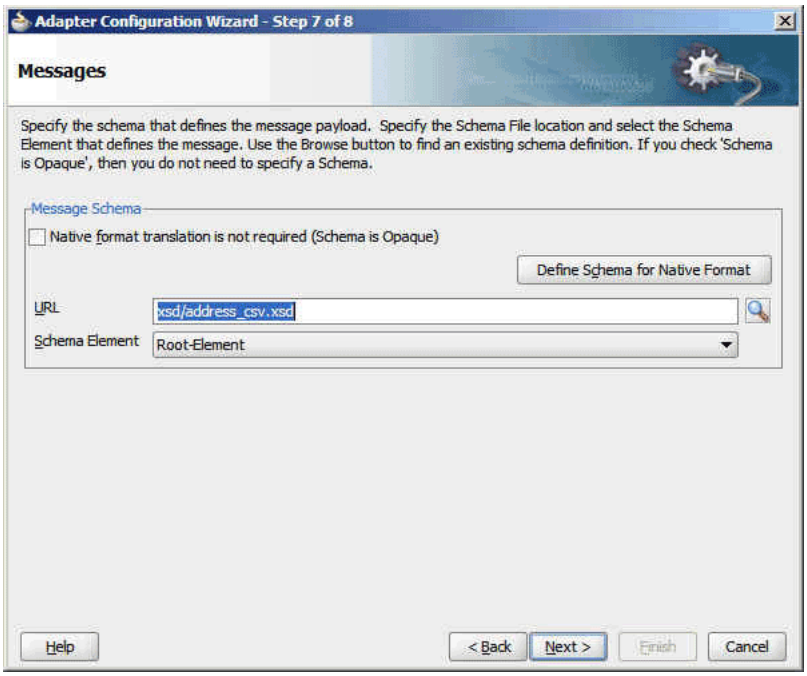
8. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
9. Select **Project Schema Files**, **address-csv.xsd**, and then **Root-Element**, as shown in Figure 10–48.

Figure 10–48 The Type Chooser Dialog



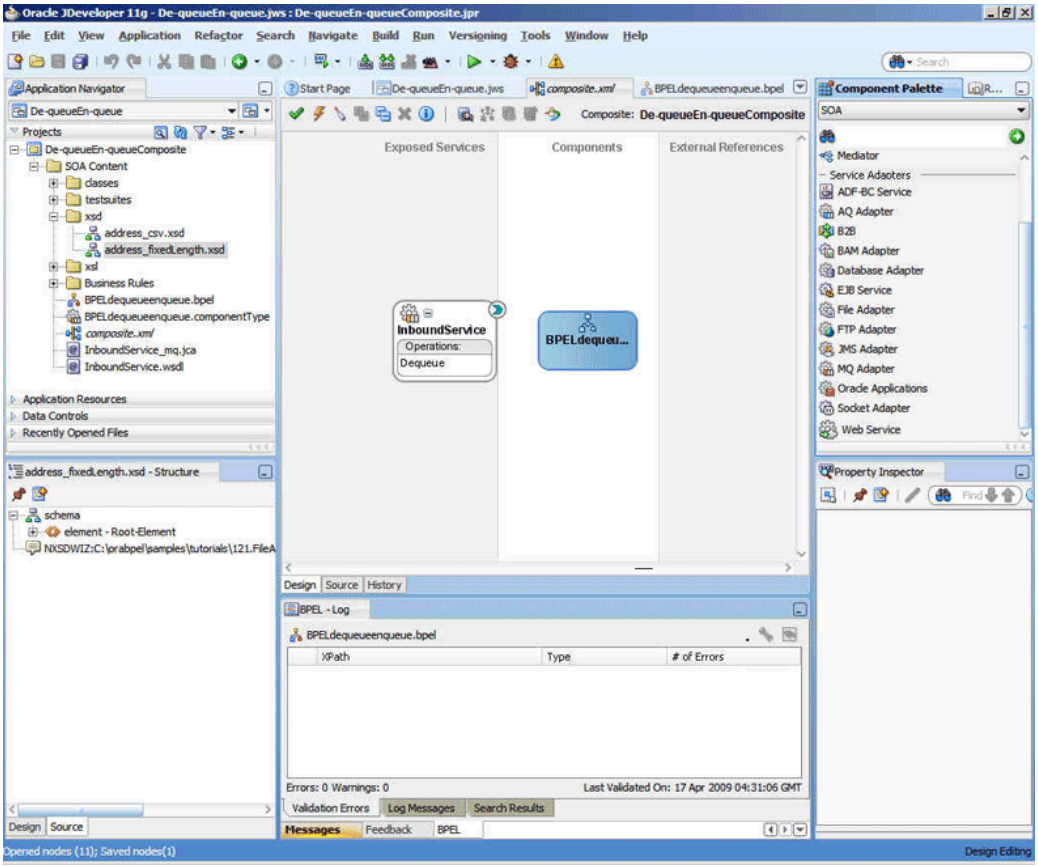
10. Click **OK**. The address-csv.xsd file appears in the URL field in the Messages page, as shown in Figure 10–49.

Figure 10–49 The Messages Page



11. Click **Next**. The Finish page is displayed.
12. Click **Finish**. You have now configured the inbound adapter service, and the composite.xml page is displayed, as shown in Figure 10–50.

Figure 10–50 The JDeveloper Page - Composite.xml Page

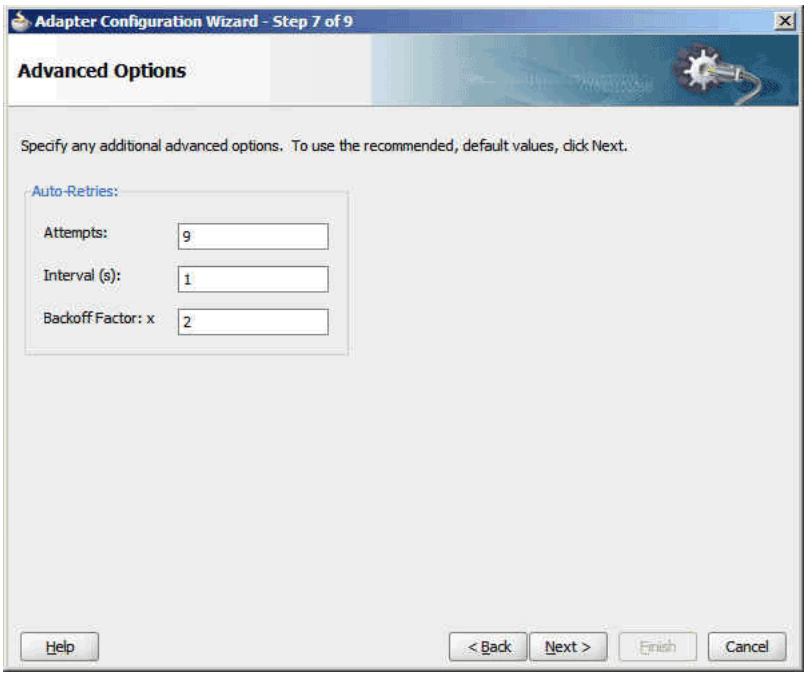


10.6.1.4 Creating an Outbound Adapter Service

Perform the following steps to create an adapter service that enqueues the messages.

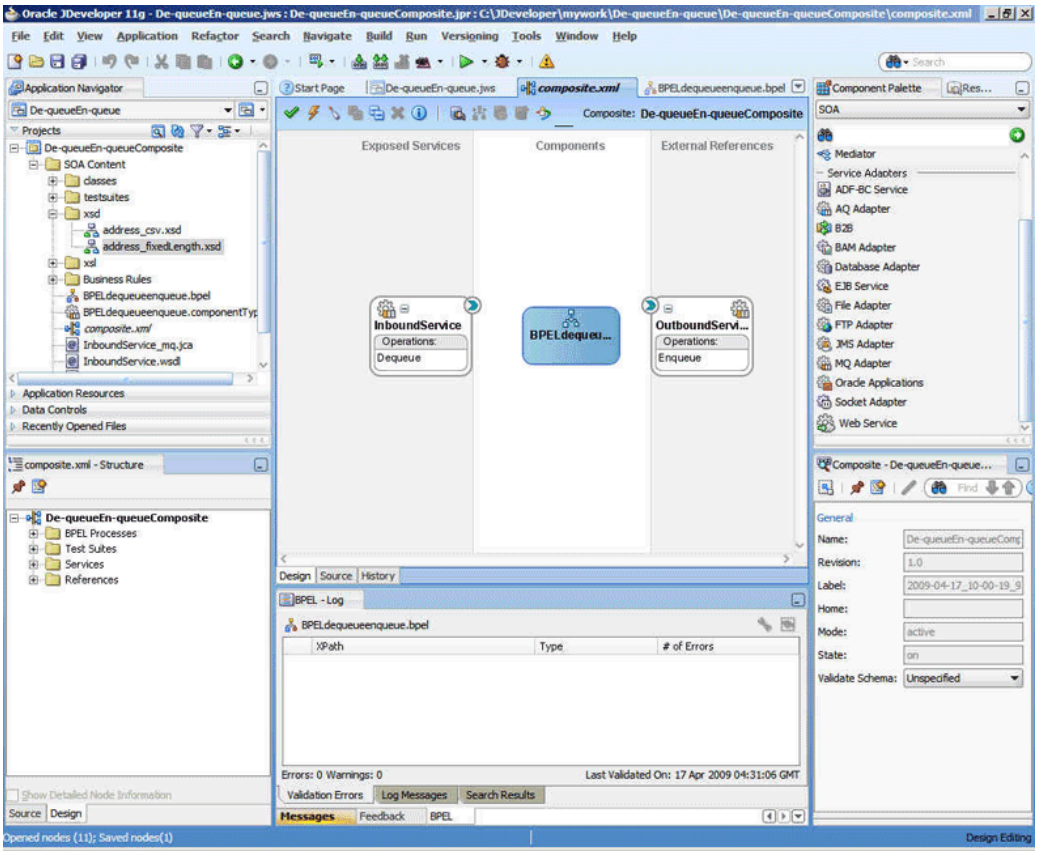
1. Drag and drop **MQ Adapter** from the Component Palette into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `OutboundService` in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, as shown in Figure 10–44, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, as shown in Figure 10–45, and click **Next**. The Operation Type page is displayed.
6. Select **Put Message into MQ**, and click **Next**. The Put Message into MQ page is displayed.
7. Enter `test_out` in the **Queue Name** field, and click **Next**. The Advanced Options page is displayed, as shown in Figure 10–51.

Figure 10–51 The Advanced Options Page



8. Accept the defaults and click **Next**. The Messages page is displayed.
9. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
10. Select **Project Schema Files**, **address-fixedLength.xsd**, and then **Root-Element**, and click **OK**. The address-fixedLength.xsd file appears in the URL field in the Messages page.
11. Click **Next**. The Finish page is displayed.
12. Click **Finish**. You have now configured the outbound adapter service, and the composite.xml page is displayed, as shown in Figure 10–52.

Figure 10–52 The JDeveloper Page - Composite.xml Page



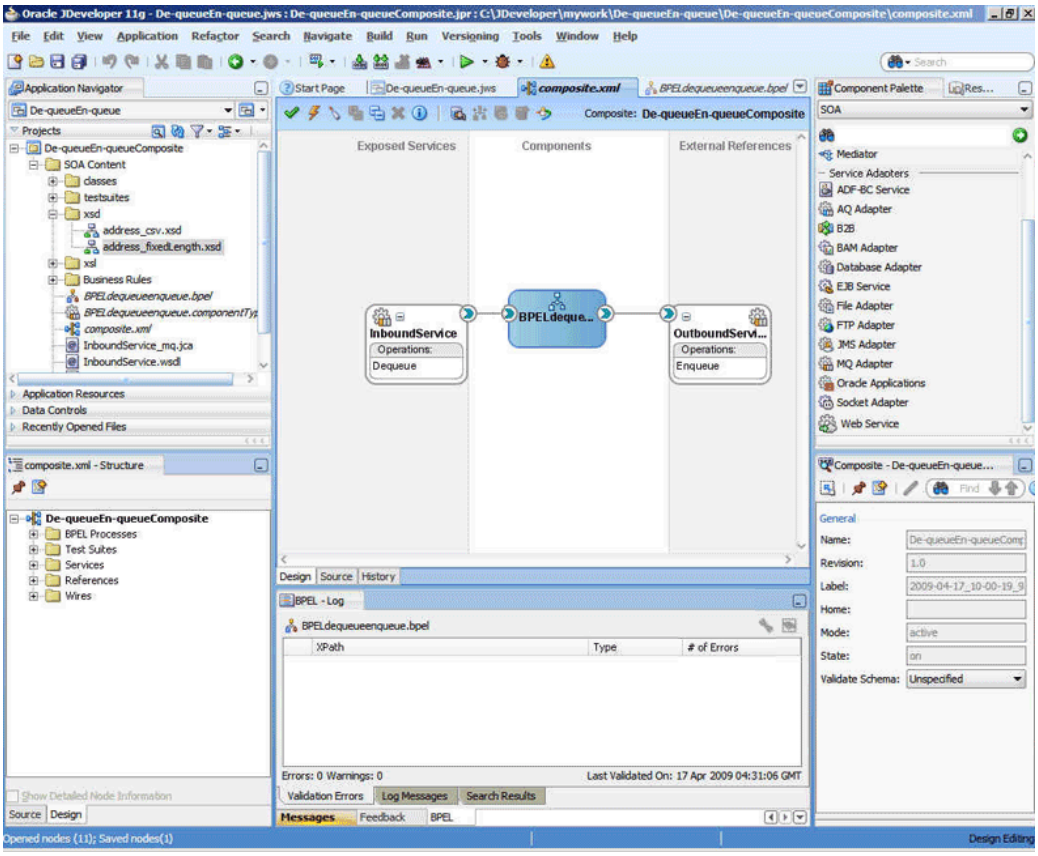
10.6.1.5 Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the InboundService in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in OutboundService in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 10–53.

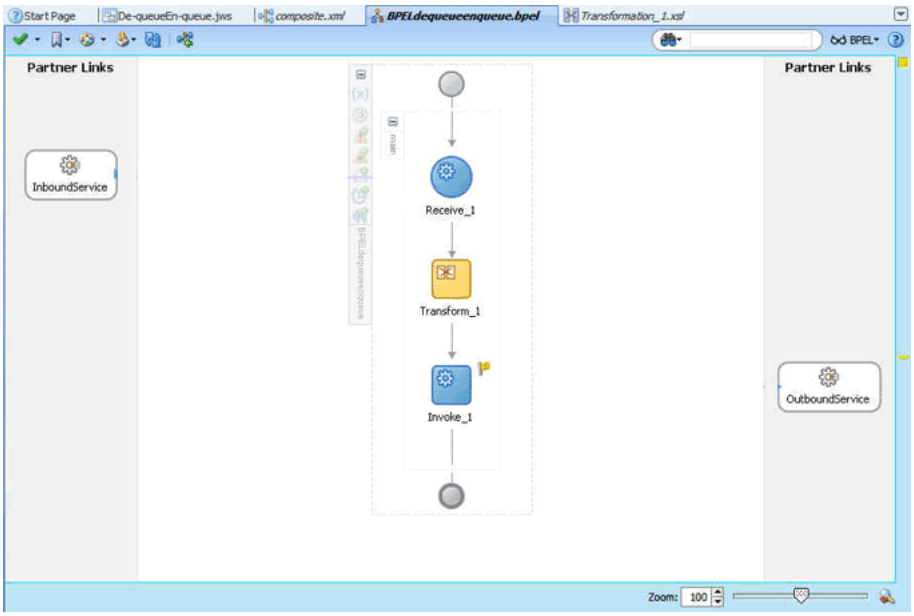
Figure 10–53 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.
4. Double-click **BPELdequeueenqueue**. The BPELdequeueenqueue.bpel page is displayed.
5. Drag and drop the **Receive**, **Transform**, and **Invoke** activities in the order mentioned from the Component Palette to the Components area.

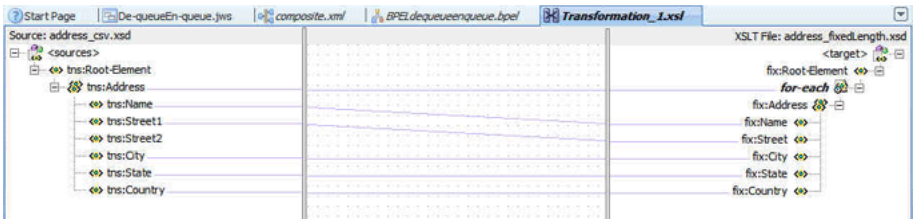
   The JDeveloper BPELdequeueenqueue.bpel page is displayed, as shown in Figure 10–54.

Figure 10–54 The BPELdequeueenqueue.bpel Page



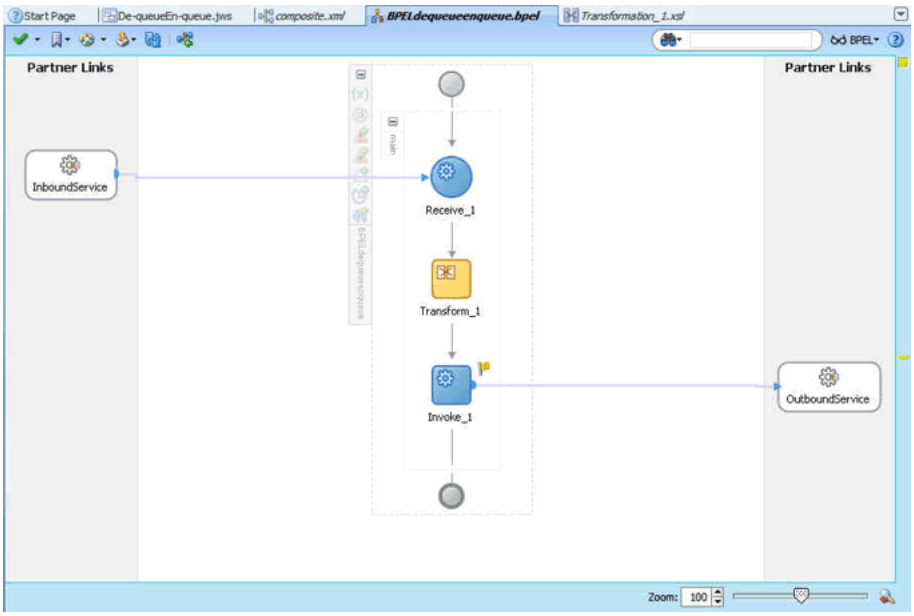
6. Drag and drop the **Receive** activity to InboundService. The Receive dialog is displayed.
7. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The Create Variable dialog is displayed.
8. Accept the defaults, and click **OK**.
9. Check the **Create Instance** box, and click **OK**.
10. Drag and drop the **Invoke** activity to OutboundService. The Invoke dialog is displayed.
11. Click the **Automatically Create Input Variable** icon that appears at the end of the Input Variable field.
12. Accept the defaults, and click **OK**. The Invoke dialog is displayed.
13. Click **OK**.
14. Double-click the **Transform** activity. The Transform dialog is displayed.
15. Click the **Create... (Alt+N)** icon. The Source Variable dialog is displayed.
16. Accept the defaults, and click **OK**.
17. Select the invoke variable as target, and click **OK**. The Transformation_xsl page is displayed.
18. Drag and drop **tns:Root-Element** in the Sources pane to the **fix:Root-Element** in the Target pane. The Auto Map Preferences dialog is displayed.
19. Click **OK**. The Transformation_xsl page is displayed, as shown in Figure 10–55.

Figure 10–55 The Trasformation_xsl Page



The BPELdequeueenqueue.bpel page appears, as shown in Figure 10–56.

Figure 10–56 The BPELdequeueenqueue.bpel Page



10.6.1.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper".

You must also create an application server connection. For more information about creating an application server connection, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters".

10.6.1.7 Monitoring Using the Oracle Enterprise Manager Fusion Middleware Control Console (Fusion Middleware Control Console)

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Log in to *http://servername:portnumber/em* using your username/password. The Oracle Enterprise Manager Fusion Middleware Control page is displayed.
2. In the left pane, navigate to **SOA**, **soa-infra (soa_server1)**. A list of all the composites that are deployed appears.

3. Click **De-queueEn-queueComposite[1.0]**. The De-queueEn-queueComposite[1.0] page is displayed.
4. Copy the **data.txt** file and put it in the test_in queue.
5. Wait for some time and then refresh the Fusion Middleware Control Console. An instance appears on the console. This is the instance that was triggered because of the processing that occurred.
6. Click the **Instances** tab.
7. Click the instance associated with this deployment. The Flow Trace page is displayed.
8. Click the **BPELdequeueenqueue** component instance. The Audit Trail page is displayed.
9. Click the **Flow** tab to debug the instance. The BPEL process instance flow is displayed.
10. Click an activity to view the relevant payload details.

10.6.2 Inbound Synchronous Request-Reply

In this use case, the inbound Oracle MQ Series Adapter dequeues the request message from MQ Series inbound queue `test_in` and publishes it to the BPEL process. The Oracle MQ Series Adapter waits for the response from the BPEL process. When the Oracle MQ Series Adapter receives the response, it enqueues the response message to the MQ Series queue specified in the `replyToQueueName` queue of the request message. This use case consists of the following sections:

- Section 10.6.2.1, "Prerequisites"
- Section 10.6.2.2, "Designing the SOA Composite"
- Section 10.6.2.3, "Creating an Inbound Adapter Service"
- Section 10.6.2.4, "Wiring Services and Activities"
- Section 10.6.2.5, "Deploying with JDeveloper"
- Section 10.6.2.6, "Monitoring Using the Fusion Middleware Control Console"

10.6.2.1 Prerequisites

This example assumes that you are familiar with basic BPEL constructs, such as activities and partner links, and JDeveloper environment for creating and deploying BPEL Process.

The Oracle MQ Series Adapter must be configured as specified in Section 10.5, "Configuring the Oracle MQ Series Adapter" and a queue `test_in` should be created.

To perform the inbound synchronous request-reply use case, you require the following files from the `artifacts.zip` file contained in the `Adapters-101MQAdapterDequeueEnqueue` sample:

- `artifacts/schemas/address-csv.xsd`
- `artifacts/schemas/address-fixedLength.xsd`
- `artifacts/input/data.txt`

You can obtain the `Adapters-101MQAdapterDequeueEnqueue` sample by accessing the Oracle SOA Sample Code site.

You must also create the following queues:

- `test_in`
- `test_reply`

10.6.2.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

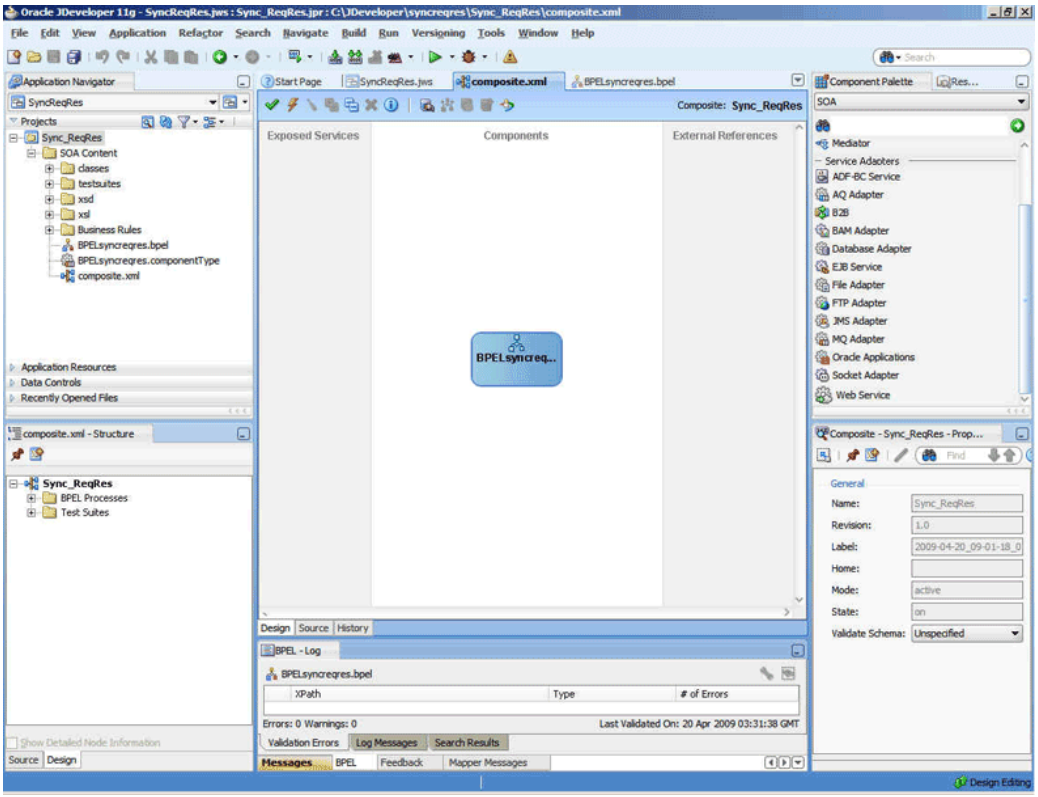
   The New Gallery dialog is displayed.

2. Expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.
4. In the **Name Your Application** screen, enter `SyncReqRes` in the **Application Name** field, and then click **Next**. The Name Your Project screen is displayed.
5. In the **Project Name** field, enter `Sync_ReqRes` and from the **Available** list, select SOA and click the right-arrow button.
6. Click **Next**. The Configure SOA Settings screen is displayed.
7. In the Composite Template list, select **Composite With BPEL** and then click **Finish**. The Create BPEL Process dialog is displayed.

   The Application Navigator of JDeveloper is updated with the new application and project and the Design tab contains, a blank palette.

8. Enter `BPELsyncreqres` in the **Name** field, select **Define Service Later** from the Template box.
9. Click **OK**. The SyncReqRes application and Sync_ReqRes project appears in the design area, as shown in Figure 10–57.

Figure 10–57 The JDeveloper - Composite.xml



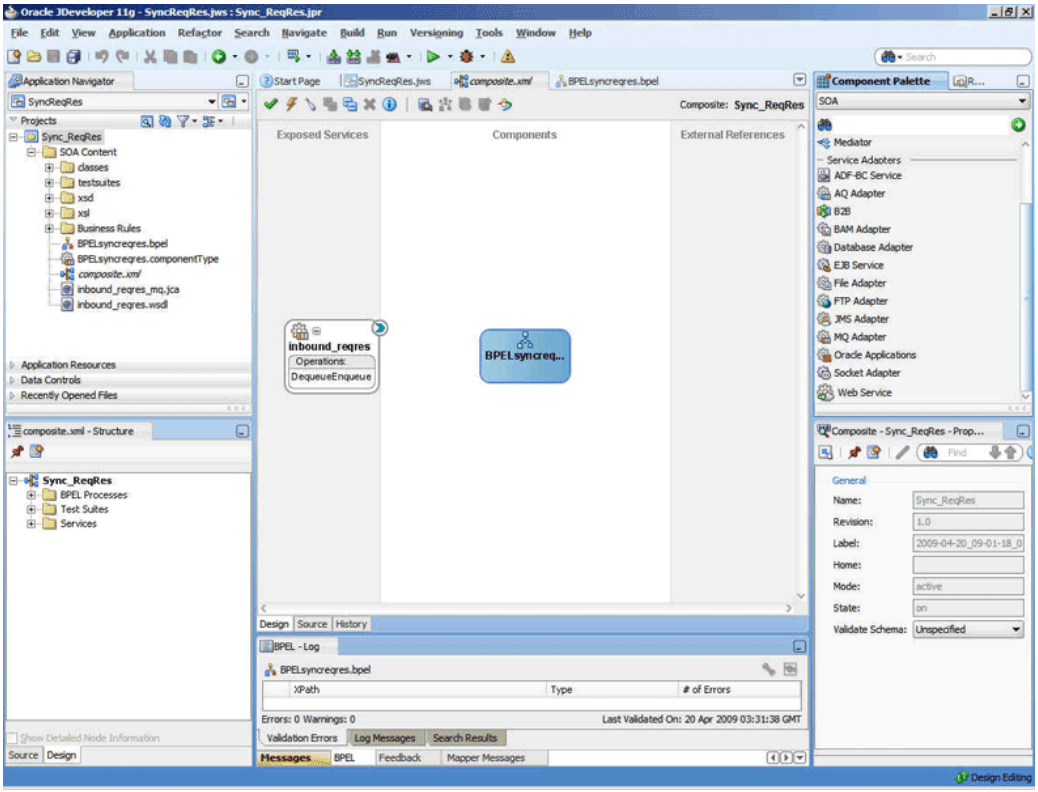
10.6.2.3 Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message from a queue:

1. Drag and drop **MQ Adapter** from the Component Palette into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `inbound_reqres` in the **Service Name** field, and click **Next**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection JNDI name, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Get Message from MQ and Send Reply/Reports**, and select **Synchronous**, as shown in Figure 10–18, and click **Next**. The Get Message from MQ and Send Reply/Reports page is displayed.
7. Select **Normal** in the Message Type box, and enter `test_in` in the **Queue Name** field.
8. Click **Next**. The Response page is displayed.
9. Accept the defaults, and click **Next**. The Messages page is displayed.
10. Select **Project Schema Files**, **address-csv.xsd**, and then **Root-Element**, and click **OK**. The address-csv.xsd file appears in the URL field in the Messages page.

11. In the Send Message Schema group, click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
12. Select **Project Schema Files**, **address-fixedLength.xsd**, and then **Root-Element**, and click **OK**. The address-fixedLength.xsd file appears in the URL field in the Messages page.
13. Click **Next**. The Finish page is displayed.
14. Click **Finish**. You have now configured the inbound adapter service, and the composite.xml page is displayed, as shown in Figure 10–58.

Figure 10–58 The JDeveloper Page - Composite.xml Page



15. Click **File**, **Save All**.

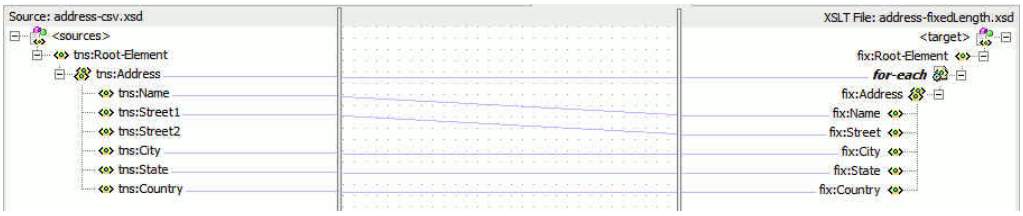
10.6.2.4 Wiring Services and Activities

Perform the following steps to wire components:

1. Drag and drop the `inbound_reqres` adapter service to the `BPELsyncreqres` BPEL process.
2. Double-click **BPELsyncreqres**. The BPELsyncreqres.bpel page is displayed.
3. Drag and drop the **Receive**, **Transform**, and **Reply** activities in the order mentioned from the Component Palette to the Components area.
4. Drag and drop the **Receive** activity to the `inbound_reqres` adapter service. The Receive dialog is displayed.
5. Enter `ReadMsg` in the **Name** field.
6. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The Create Variable dialog is displayed.

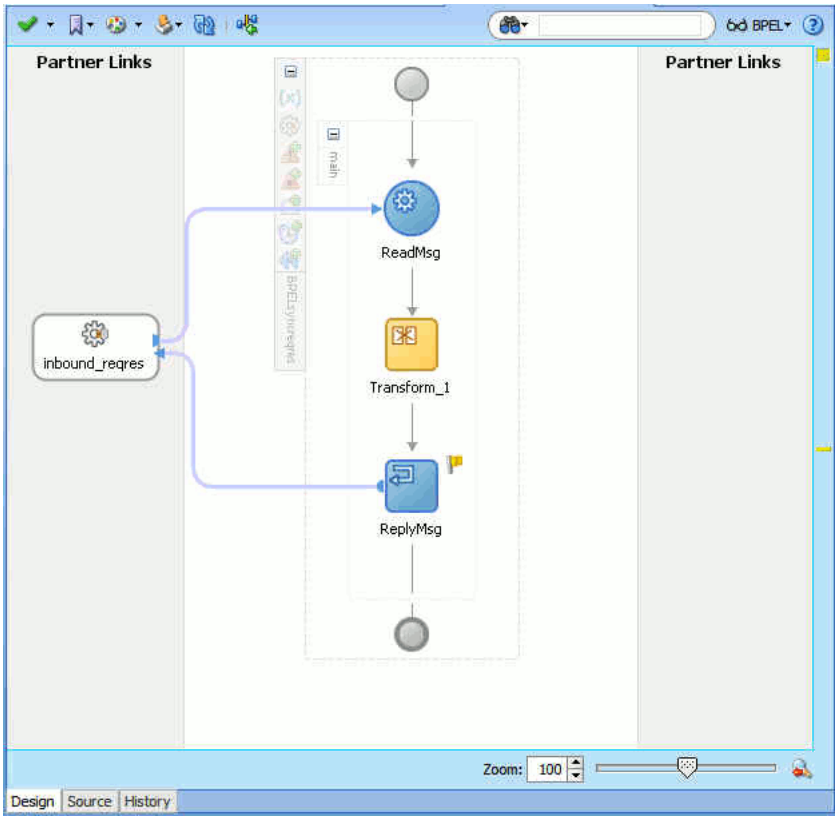
7. Accept the defaults, and click **OK**.
8. Check the **Create Instance** box in the Receive dialog, and click **OK**.
9. Drag and drop the **Reply** activity to the `inbound_reqres` adapter service. The Reply dialog is displayed.
10. Enter `ReplyMsg` in the **Name** field.
11. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The Create Variable dialog is displayed.
12. Accept the defaults, and click **OK**. The variable appears in the Reply dialog.
13. Click **OK**.
14. Double-click the **Transform** activity. The Transform dialog is displayed.
15. Click the plus icon. The Source Variable dialog is displayed.
16. From the Source Variable list, select **ReadMsg_DequeueEnqueue_InputVariable**, and click **OK**.
17. From the Target Variable list, select **ReplyMsg_DequeueEnqueue_ OutputVariable**.
18. Click the **Create Mappings** icon. The Transformation.xsl page is displayed, as shown in Figure 10–59.

Figure 10–59 The Transformation.xsl Page



19. Drag the t**ns:Root-Element** from <sources> panel to the **fix:Root-Element** of the <target> panel. The Auto Map Preferences dialog is displayed.
20. Click **OK**. The JDeveloper BPELsyncreqres.bpel page is displayed, as shown in Figure 10–60.

Figure 10–60 The BPELsyncreqres.bpel Page



10.6.2.5 Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

To deploy the application profile using JDeveloper, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper".

You must also create an application server connection. For more information about creating an application server connection, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters".

10.6.2.6 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Log in to *http://servername:portnumber/em* using your username/password. The Oracle Enterprise Manager Fusion Middleware Control page is displayed.
2. In the left pane, navigate to **SOA**, **soa-infra (soa_server1)**. A list of all the composites that are deployed appears.
3. Click **Sync_ReqRes[1.0]**. The Sync_ReqRes[1.0] page is displayed.
4. Create an MQ message with the contents of the `data.txt` file and set **replyToQueueName** to `test_reply`. Put this message in the `test_in` queue.
5. Wait for some time and then refresh the Fusion Middleware Control Console. An instance appears on the console. This is the instance that was triggered because of the processing that occurred.

6. Click the **Instances** tab.
7. Click the instance associated with this deployment. The Flow Trace page is displayed.
8. Click the **BPELsyncreqres** component instance. The Audit Trail page is displayed.
9. Click the **Flow** tab to debug the instance. The BPEL process instance flow is displayed.
10. Click an activity to view the relevant payload details.

10.6.3 Inbound-Outbound Synchronous Request-Reply

This use case is the end-to-end demonstration of the Synchronous Solicit Request-Reply scenario for MQ Adapter. In this use case, the composite dequeues the message from an inbound queue. Then, it enqueues a reply message to the replyToQueue queue as specified in the inbound message. This section contains the following topics:

- Section 10.6.3.1, "Prerequisites"
- Section 10.6.3.2, "Designing the SOA Composite"
- Section 10.6.3.3, "Creating an Inbound Adapter Service"
- Section 10.6.3.4, "Creating an Outbound Adapter Service"
- Section 10.6.3.5, "Wiring Services and Activities"
- Section 10.6.3.6, "Deploying with JDeveloper"
- Section 10.6.3.7, "Monitoring Using the Fusion Middleware Control Console"

10.6.3.1 Prerequisites

To perform the inbound synchronous request-reply use case, you require the following files from the `artifacts.zip` file contained in the `Adapters-101MQAdapterDequeueEnqueue` sample:

- `artifacts/schemas/address-csv.xsd`
- `artifacts/schemas/address-fixedLength.xsd`

You must also create queues named:

- `test_in`
- `test1`
- `ReplyQ`
- `test_reply`

You can obtain the `Adapters-101MQAdapterDequeueEnqueue` sample by accessing the Oracle SOA Sample Code site.

10.6.3.2 Designing the SOA Composite

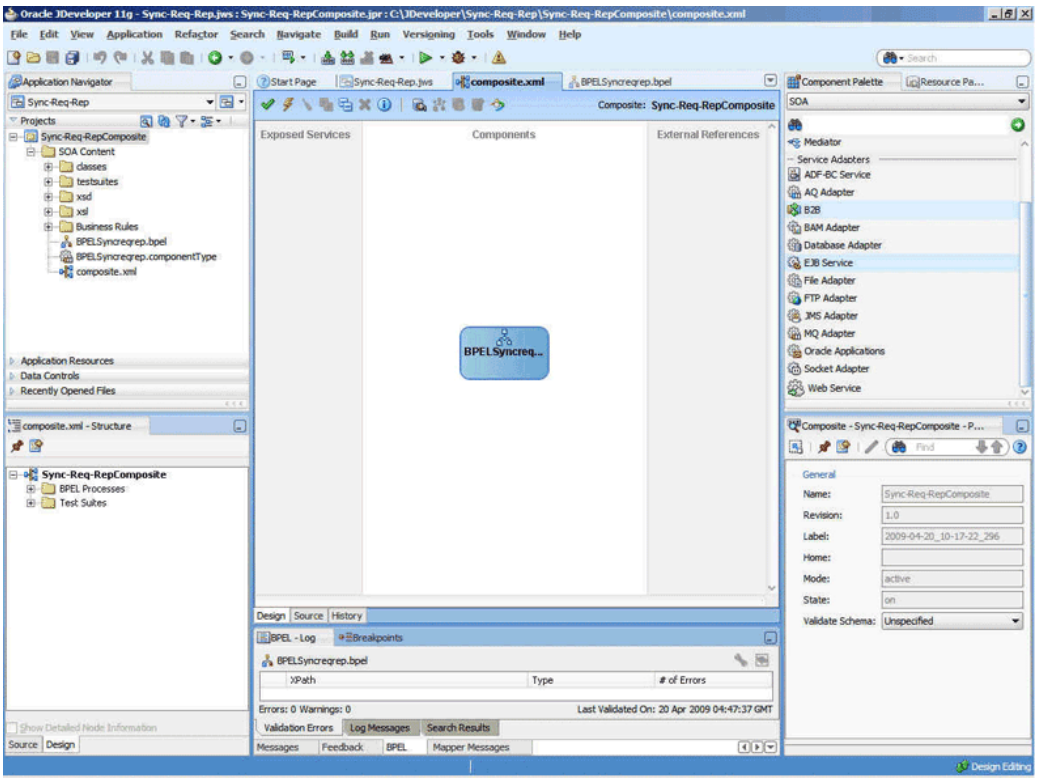
You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.
2. Expand the **General** node, and select the **Applications** category.

3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.
4. In the **Name Your Application** screen, enter `Sync-Req-Rep` in the **Application Name** field, and then click **Next**. The Name Your Project screen is displayed.
5. In the **Project Name** field, enter `Sync-Req-RepComposite` and from the **Available** list, select SOA and click the right-arrow button.
6. Click **Next**. The Configure SOA Settings screen is displayed.
7. In the Composite Template list, select **Composite With BPEL** and then click **Finish**. The Create BPEL Process dialog is displayed.
8. Enter `BPELSyncreqrep` in the **Name** field, select **Define Service Later** from the Template box.
9. Click **OK**. The Sync-Req-Rep application and Sync-Req-RepComposite project appears in the design area, as shown in Figure 10–61.

Figure 10–61 The JDeveloper - Composite.xml



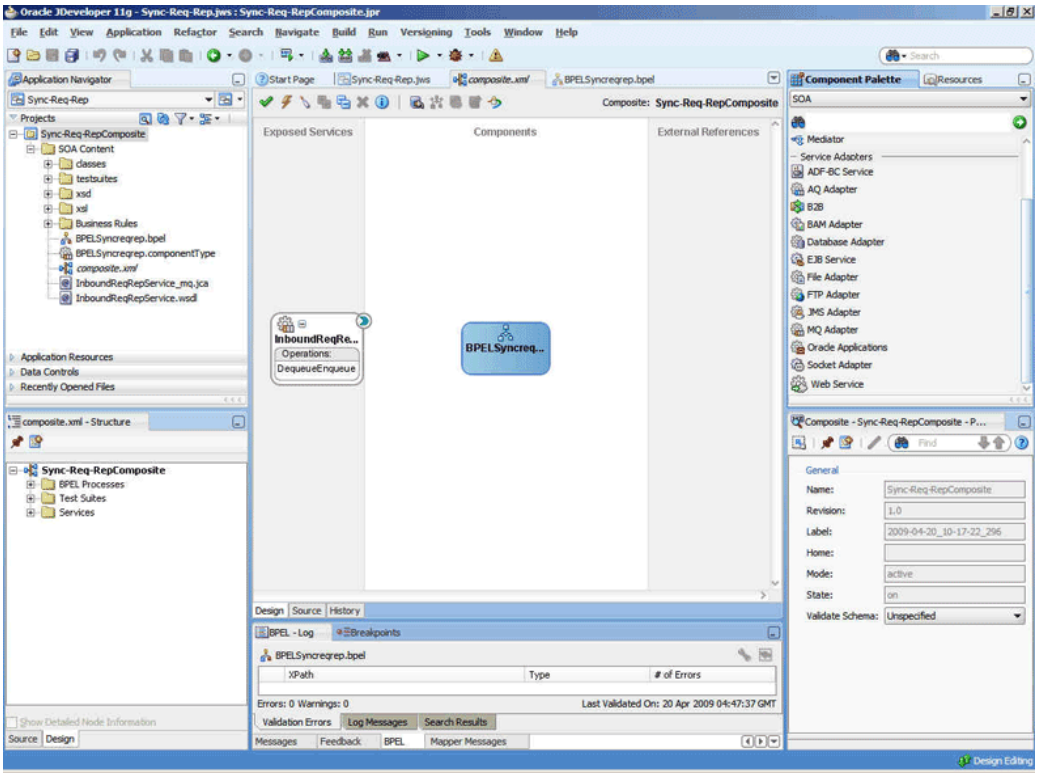
10.6.3.3 Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message from a queue:

1. Drag and drop **MQ Adapter** from the Component Palette into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `InboundReqRepService` in the **Service Name** field, and click **Next**. The MQ Series Connection page is displayed.

4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Get Message from MQ and Send Reply/Reports** and **Synchronous** in the **Operation Name** box, as shown in Figure 10–18, and click **Next**. The Get Message from MQ and Send Reply/Reports page is displayed.
7. Select **Normal** in the Message Type list and enter `test_in` in the **Queue Name** field and select **Choose Other Schema** in the **Schema Options** box, and click **Next**. The Response page is displayed.
8. Accept the defaults and click **Next**. The Message page is displayed.
9. Click **Browse** in the Get Message Schema box that appears at the end of the URL field. The Type Chooser dialog is displayed.
10. Select **Project Schema Files**, **address-csv.xsd**, and **Root-Element**, and then click **OK**. The Message page is populated with the address-csv.xsd file in the Get Message Schema box.
11. Click **Browse** in the Send Message Schema box that appears at the end of the URL field. The Type Chooser dialog is displayed.
12. Select **Project Schema Files**, **address-fixedLength.xsd**, and **Root-Element**, and then click **OK**. The Message page is populated with the address-fixedLength.xsd file in the Send Message Schema box.
13. Click **Next**. The Finish page is displayed.
14. Click **Finish**. You have configured the InboundReqRepService adapter service, and the composite.xml page is displayed, as shown in Figure 10–62.

Figure 10–62 The JDeveloper Page - Composite.xml Page



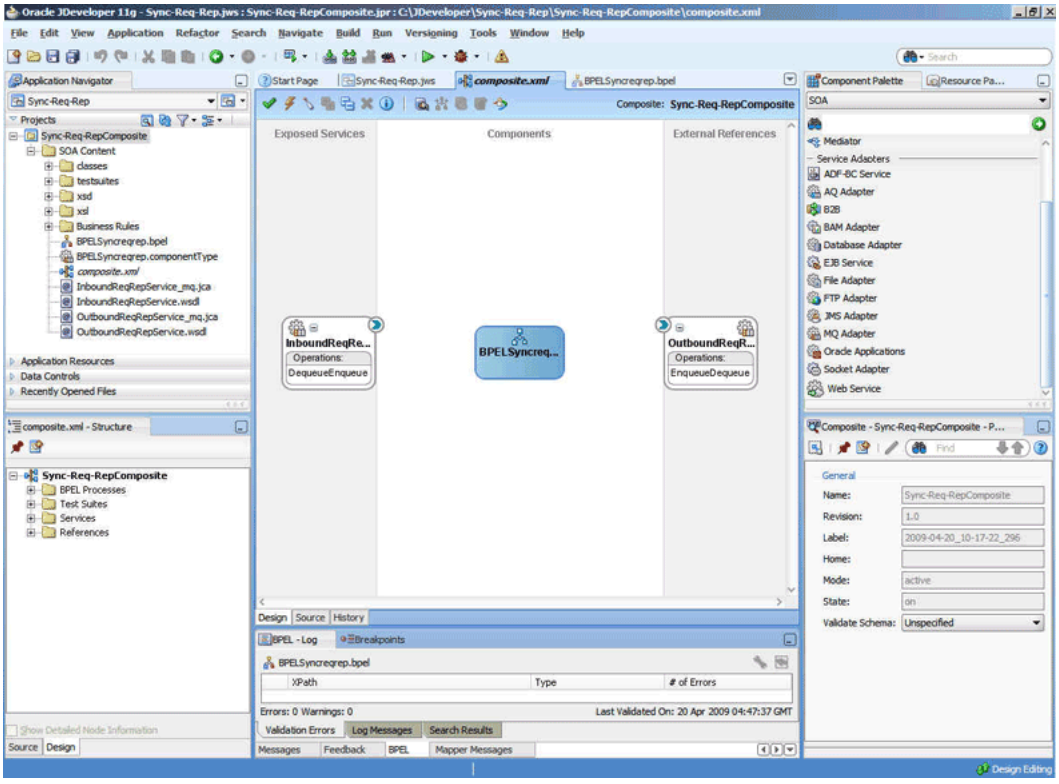
10.6.3.4 Creating an Outbound Adapter Service

Perform the following steps to create an adapter service that enqueues the request messages and dequeue the corresponding response messages (report) from a queue:

1. Drag and drop **MQ Adapter** from the Component Palette into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `OutboundReqRepService` in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.
4. Accept the defaults and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Send Message to MQ and Get Reply/Reports**, select **Synchronous** in the Operation Name box, and click **Next**. The Send Message to MQ and Get Reply/Reports page is displayed.
7. Enter `test1` in the **Queue Name** field and click **Next**. The Response page is displayed.
8. Enter the name of the queue in the **Reply To Queue Name** field such as `ReplyQ`, select the **Response Wait Interval** option and enter a value, and select the **Empty Response Message Allowed** option.
9. Click **Next**. The Advanced Options page is displayed.
10. Accept the default values and click **Next**. The Messages page is displayed.

11. Click **Browse** in the Get Message Schema box that appears at the end of the URL field. The Type Chooser dialog is displayed.
12. Select **Project Schema Files**, **address-csv.xsd**, and **Root-Element**, and then click **OK**. The Message page is populated with address-csv.xsd file in the Get Message Schema box.
13. Click **Browse** in the Send Message Schema box that appears at the end of the URL field. The Type Chooser dialog is displayed.
14. Select **Project Schema Files**, **address-fixedLength.xsd**, and **Root-Element**, and then click **OK**. The Message page is populated with address-fixedLength.xsd file in the Send Message Schema box.
15. Click **Next**. The Finish page is displayed.
16. Click **Finish**. You have configured the OutboundReqRepService service, and the composite.xml page is displayed, as shown in Figure 10–63.

Figure 10–63 The JDeveloper Page - Composite.xml Page



10.6.3.5 Wiring Services and Activities

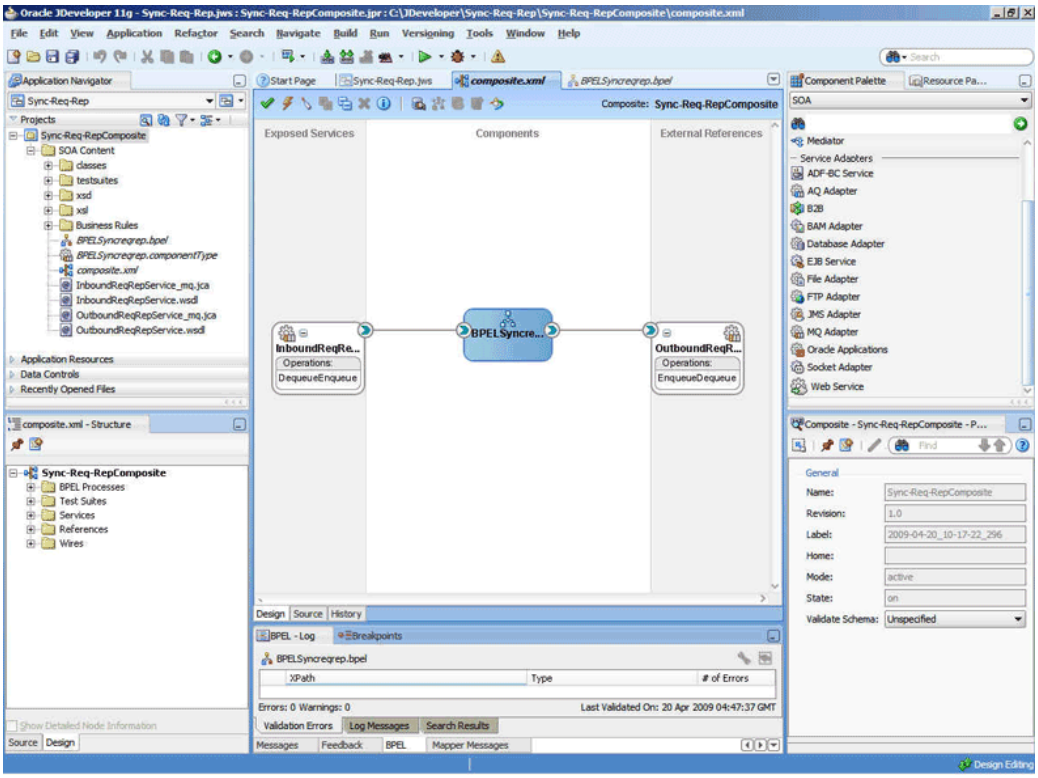
You have to assemble or wire the three components that you have created: InboundReqRepService, BPELSyncreqrep, and OutboundReqRepService. Perform the following steps to wire the components:

1. Drag the small triangle in the InboundReqRepService service in the Exposed Services area to the drop zone that appears as a green triangle in BPELSyncreqrep in the Components area.
2. Drag the small triangle in BPELSyncreqrep in the Components area to the drop zone that appears as a green triangle in OutboundReqRepService in the External References area.

3. Similarly, drag the small triangle in BPELSyncreqrep in the Components area to the drop zone in OutboundReqRepService in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 10–64.

Figure 10–64 The JDeveloper - Composite.xml

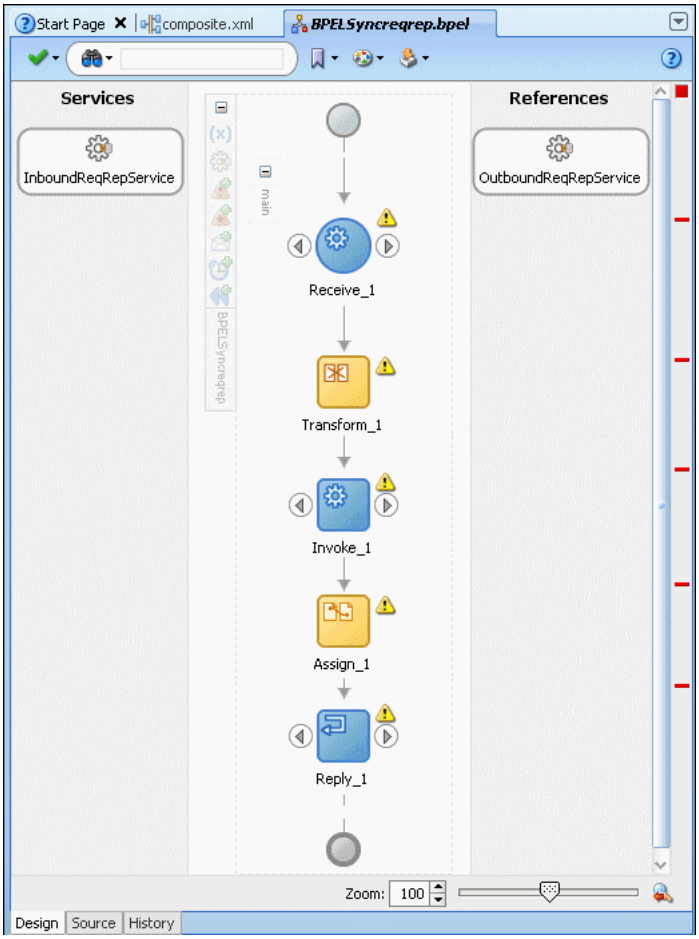


4. Click **File**, **Save All**.

5. Double-click **BPELSyncreqrep**. The BPELSyncreqrep.bpel page is displayed.

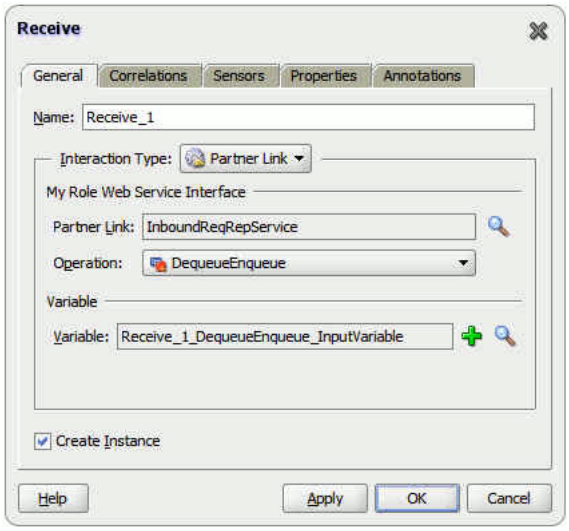
6. Drag and drop the **Receive**, **Transform**, **Invoke**, **Assign**, **Reply** activities in the order mentioned from the Component Palette to the Components area. The JDeveloper BPELSyncreqrep.bpel page is displayed, as shown in Figure 10–65.

Figure 10–65 The BPELSyncreqrep.bpel Page



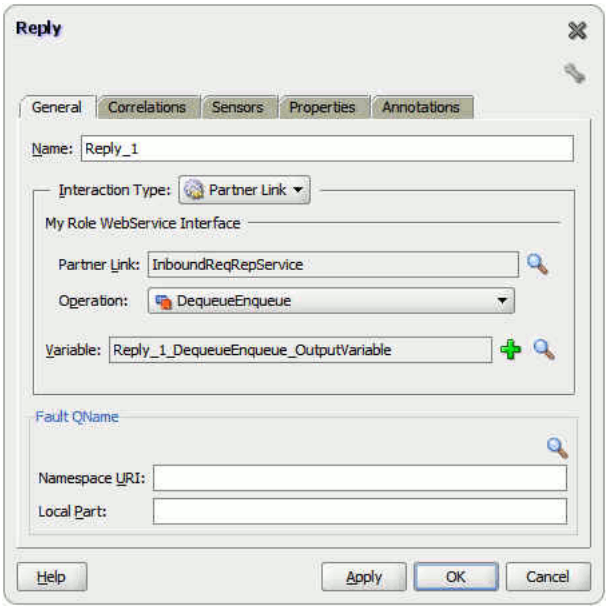
7. Drag and drop the **Receive** activity to `InboundReqRepService`. The Receive dialog is displayed.
8. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The Create Variable dialog is displayed.
9. Accept the defaults, and click **OK**.
10. Check the **Create Instance** box, as shown in Figure 10–66, and click **OK**.

Figure 10–66 The Receive Dialog



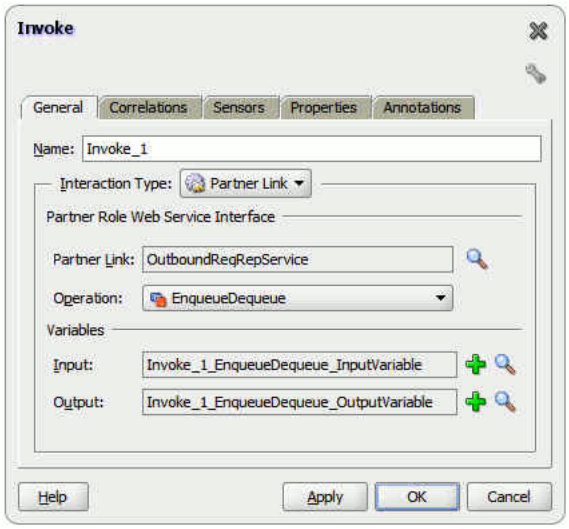
11. Drag and drop the Reply activity to InboundReqRepService. The Reply dialog is displayed.
12. Click the **Auto Create Variable** icon to create the variable, and then click **OK**. The Reply dialog is displayed, as shown in Figure 10–67.

Figure 10–67 The Reply Dialog



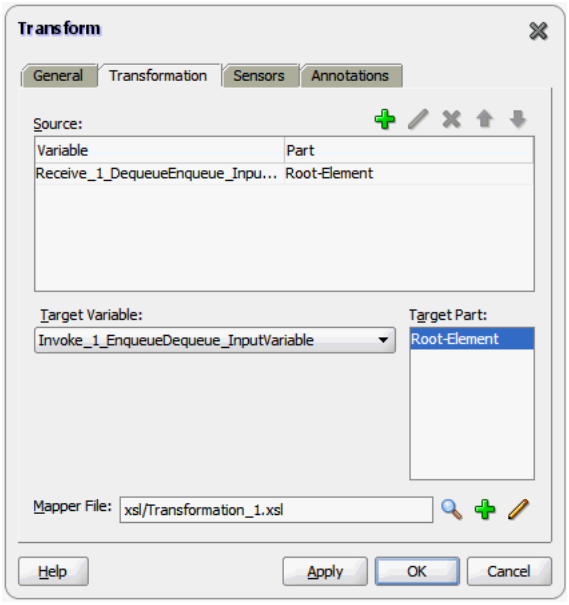
13. Drag and drop the **Invoke** activity to the OutboundReqRepService service. The Invoke dialog is displayed.
14. Click the **Automatically Create Input Variable** icon that appears at the end of the Input Variable field. The Create Variable dialog is displayed.
15. Click **OK**.
16. Similarly, create the output variable. Accept the defaults, and click **OK**. The Invoke dialog is displayed, as shown in Figure 10–68.

Figure 10–68 The Invoke Dialog



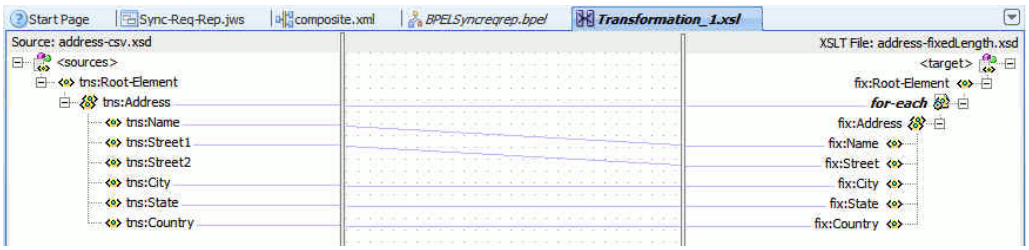
17. Click **OK**.
18. Double-click the **Transform** activity. The Transform dialog is displayed.
19. Click the plus icon, and select **Receive_1_DequeueEnqueue_InputVariable** as the source variable. Then, select **Invoke_1_EnqueueDequeue_InputVariable** for the target variable, as shown in Figure 10–69.

Figure 10–69 The Transform Dialog



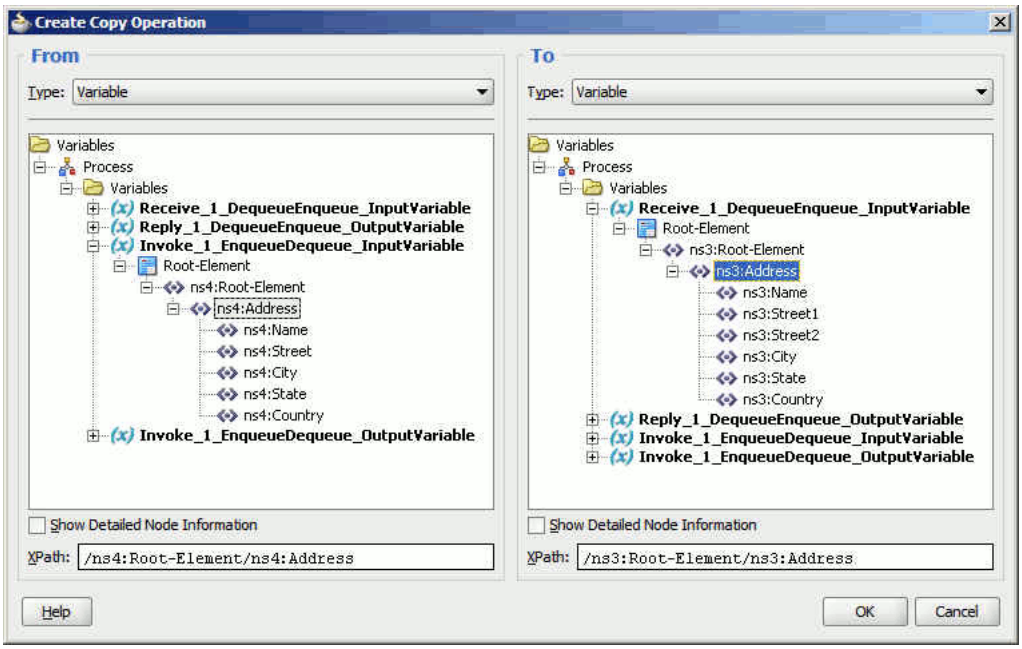
20. Click **Create Mapping**. The Transformation_1.xsl page is displayed.
21. Drag and drop the **tns:Root-Element** from the from <sources> panel to **fix:Root-Element** in the <target> panel. The Auto Map Preferences dialog is displayed.
22. Click **OK**. The mappings appear in the Transformation.xsl page, as shown in Figure 10–70.

Figure 10–70 The Transformation.xsl Page with Mappings



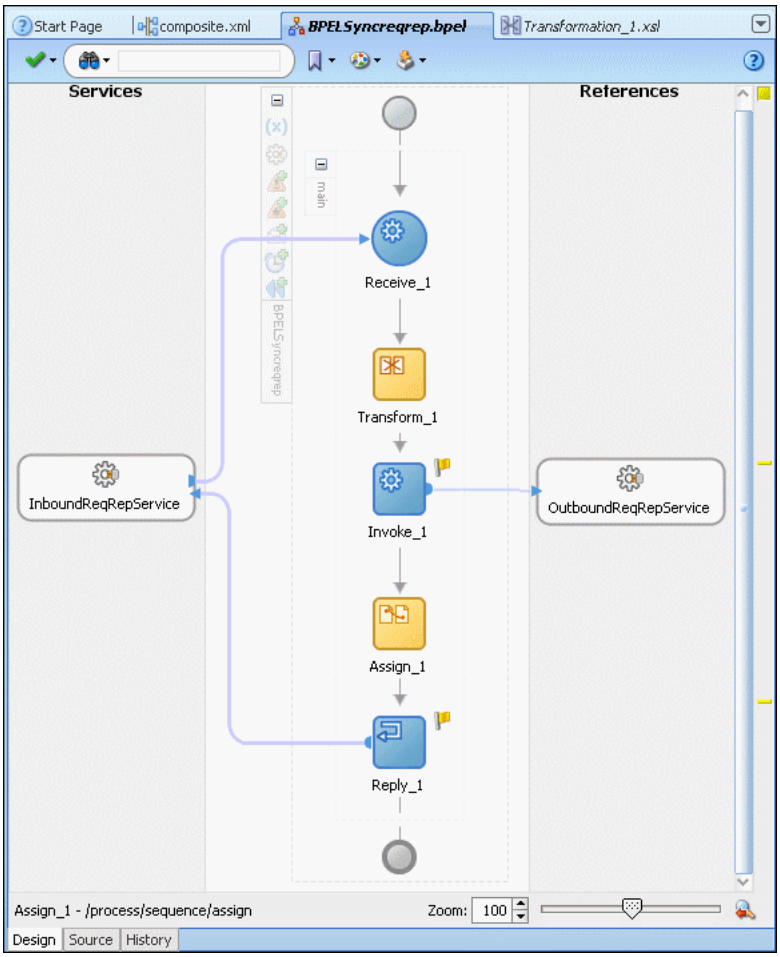
23. Click the **BPELSyncreqrep.bpel** tab.
24. Double-click the **Assign** activity. The Assign dialog is displayed.
25. Click the plus icon, and select **Copy Operation**. The Create Copy Operation dialog is displayed.
26. Select the variables, as shown in Figure 10–71, and click **OK**.

Figure 10–71 The Create Copy Operation Dialog



27. Click **OK** in the Assign dialog. The JDeveloper BPELSyncreqrep.bpel page is displayed, as shown in Figure 10–72.

Figure 10–72 The BPELSyncreqrep.bpel Page



10.6.3.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper".

You must also create an application server connection. For more information about creating an application server connection, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters".

10.6.3.7 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Log in to *http://servername:portnumber/em* using your username/password. The Oracle Enterprise Manager Fusion Middleware Control page is displayed.
2. In the left pane, navigate to **SOA**, **soa-infra (soa_server1)**. A list of all the composites that are deployed appears.
3. Click **Sync-Req-RepComposite[1.0]**. The Sync-Req-RepComposite[1.0] page is displayed.

4. Create an MQ message with the contents of the `data.txt` file and set **replyToQueueName** to `test_reply`. Put this message in the `test_in` queue.
5. Wait for some time and then refresh the Fusion Middleware Control Console. An instance appears on the console. This is the instance that was triggered because of the processing that occurred.
6. Click the **Instances** tab.
7. Click the instance associated with this deployment. The Flow Trace page is displayed.
8. Click the **BPELSyncreqrep** component instance. The Audit Trail page is displayed.
9. Click the **Flow** tab to debug the instance. The BPEL process instance flow is displayed.
10. Click an activity to view the relevant payload details.

10.6.4 Asynchronous-Request-Reply

This use case is the end-to-end demonstration of the Asynchronous-Request-Reply scenario. In this use case, first, the composite dequeues the message from an inbound queue. Then, it enqueues a request message and dequeues the reply message. Finally, the composite enqueues the reply message to the other queue. This section contains the following topics:

- Section 10.6.4.1, "Prerequisites"
- Section 10.6.4.2, "Designing the SOA Composite"
- Section 10.6.4.3, "Creating an Inbound Adapter Service"
- Section 10.6.4.4, "Creating an Asynchronous Outbound Request Reply Adapter Service Outbound"
- Section 10.6.4.6, "Wiring Services and Activities"
- Section 10.6.4.7, "Deploying with JDeveloper"
- Section 10.6.4.8, "Monitoring Using the Fusion Middleware Control Console"

10.6.4.1 Prerequisites

The Oracle MQ Series Adapter must be configured as specified in Section 10.5, "Configuring the Oracle MQ Series Adapter" and create the following queues: test_in, test_out, and test_demo queues.

10.6.4.2 Designing the SOA Composite

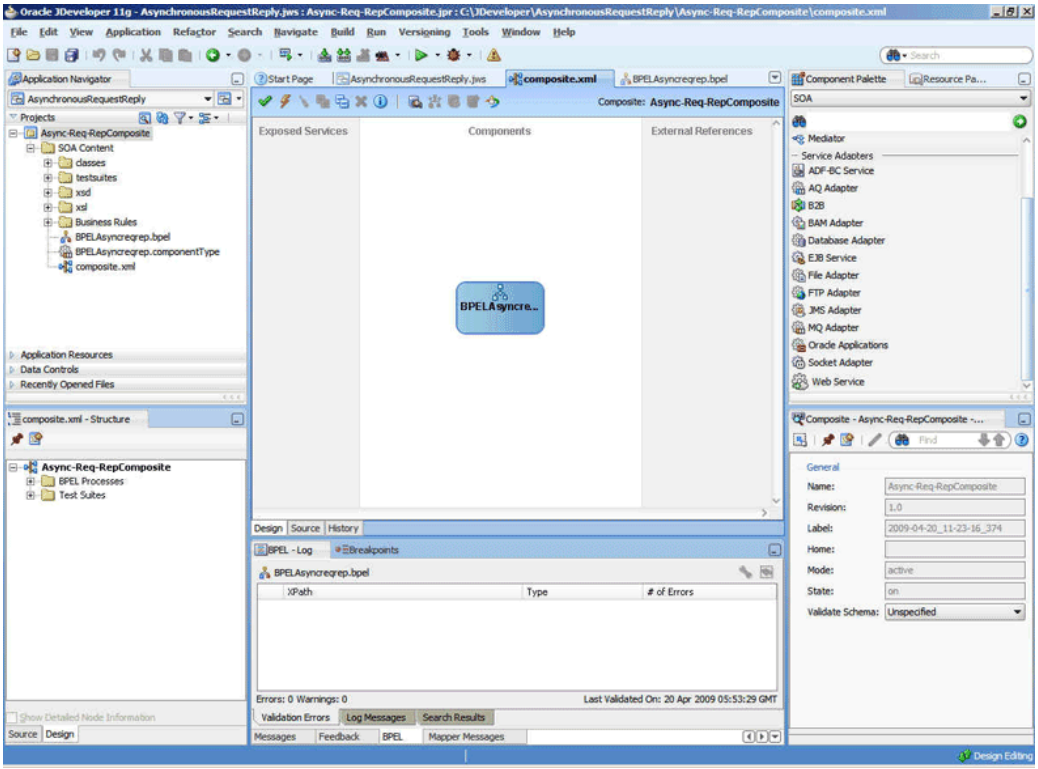
You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.
2. Expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.
4. In the **Name Your Application** screen, enter `AsynchronousRequestReply` in the **Application Name** field, and then click **Next**. The Name Your Project screen is displayed.

5. In the **Project Name** field, enter `Async-Req-RepComposite` and from the **Available** list, select SOA and click the right-arrow button.
6. Click **Next**. The Configure SOA Settings screen is displayed.
7. In the Composite Template list, select **Composite With BPEL** and then click **Finish**. The Create BPEL Process dialog is displayed.
8. Enter `BPELAsyncreqrep` in the **Name** field, select **Define Service Later** from the Template box.
9. Click **OK**. The AsynchronousRequestReply application and the Async-Req-RepComposite project appear in the design area, as shown in Figure 10–73.

Figure 10–73 The JDeveloper - Composite.xml



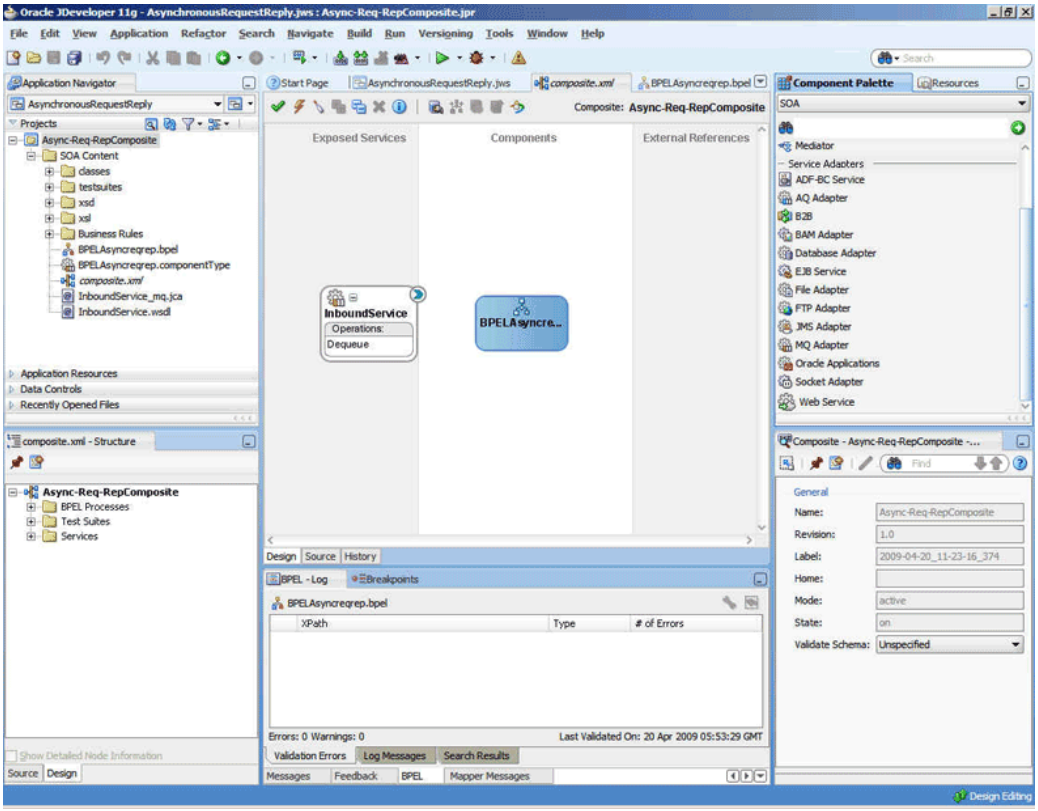
10.6.4.3 Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message from a queue:

1. Drag and drop **MQ Adapter** from the Component Palette into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `InboundService` in the **Service Name** field, and click **Next**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.

5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Get Message from MQ** and click **Next**. The Get Message from MQ page is displayed.
7. Enter `test_in` in the **Queue Name** field and click **Next**. The Messages page is displayed.
8. Select **Native Format Translation is not required (Schema is Opaque)** and click **Next**. The Finish page is displayed.
9. Click **Finish**. You have configured the inbound adapter service, and the composite.xml page is displayed, as shown in Figure 10–74.

Figure 10–74 The JDeveloper Page - Composite.xml Page



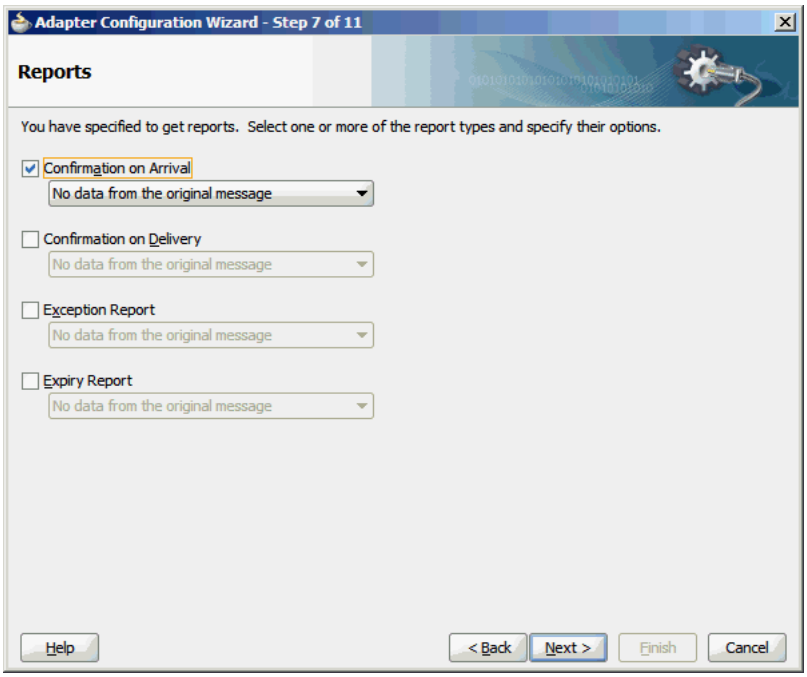
10.6.4.4 Creating an Asynchronous Outbound Request Reply Adapter Service Outbound

Perform the following steps to create an adapter service that enqueues the request messages and dequeue the corresponding response messages (report) from a queue:

1. Drag and drop **MQ Adapter** from the Component Palette into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `asyn-Req-Res` in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.

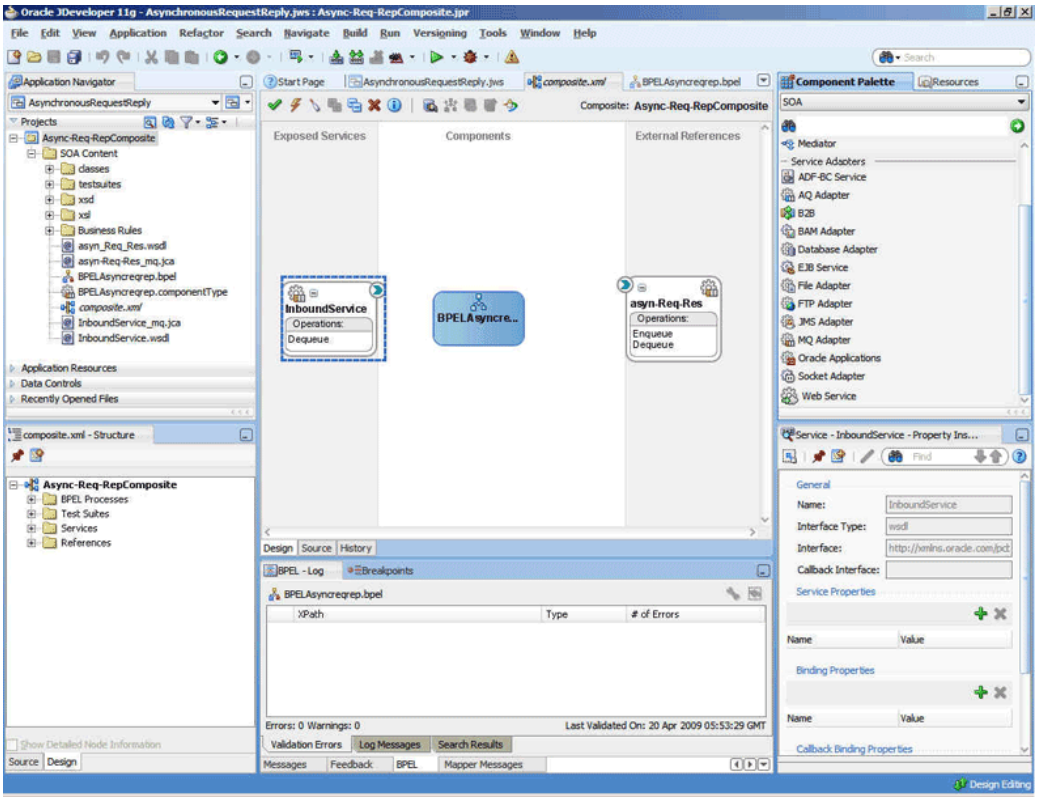
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Send Message to MQ and Get Reply/Reports** and select **Asynchronous** in the Operation Name box, and then click **Next**. The Send Message to MQ and Get Reply/Reports page is displayed.
7. Select **Normal** in the Message Type box and enter `test_out` in the **Queue Name** field, and then select the **Get Reports** check box, and click **Next**. The Reports page is displayed.
8. Select **Confirmation on Arrival**, as shown in Figure 10–75, and click **Next**. The Response page is displayed.

Figure 10–75 The Adapter Configuration Wizard Reports Page



9. Enter `test_out` in the **Reply To Queue Name** field, and click **Next**. The Advanced Options page is displayed.
10. Accept the default values, and click **Next**. The Messages page is displayed.
11. Select **Native Format Translation is not Required(Schema is Opaque)** in both the Get Message Schema and Send Message Schema boxes, and click **Next**. The Finish page is displayed.
12. Click **Finish**. You have configured the **async-Req-Res** service, and the composite.xml page is displayed, as shown in Figure 10–76.

Figure 10–76 The JDeveloper Page - Composite.xml Page

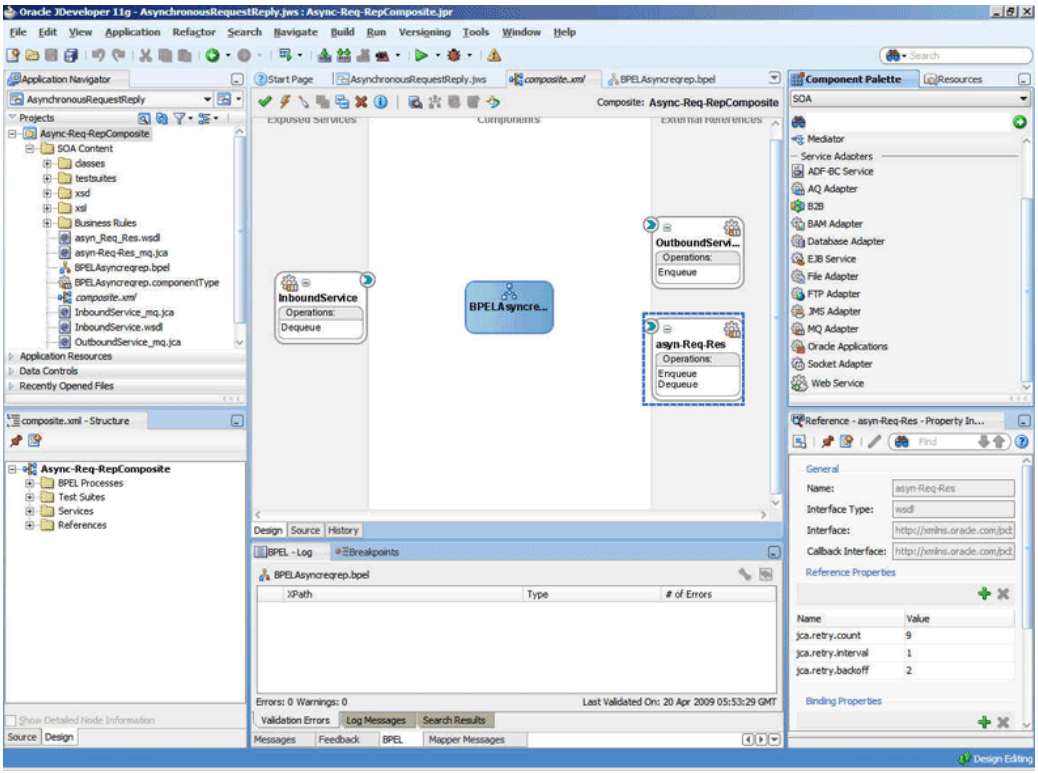


10.6.4.5 Creating Another Outbound Adapter Service

Perform the following steps to create an adapter service that enqueues the response (report) messages.

1. Drag and drop **MQ Adapter** from the Component Palette into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `OutboundService` in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Put Message into MQ**, and click **Next**. The Put Message into MQ page is displayed.
7. Enter `test_demo` in the **Queue Name** field, and click **Next**. The Advanced Options page is displayed.
8. Accept the default values, and click **Next**. The Messages page is displayed.
9. Select **Native Format Translation is not required(Schema is Opaque)**, and click **Next**. The Finish page is displayed.
10. Click **Finish**. You have configured the **OutboundService** service, and the composite.xml page is displayed, as shown in Figure 10–77.

Figure 10–77 The JDeveloper Page - Composite.xml Page



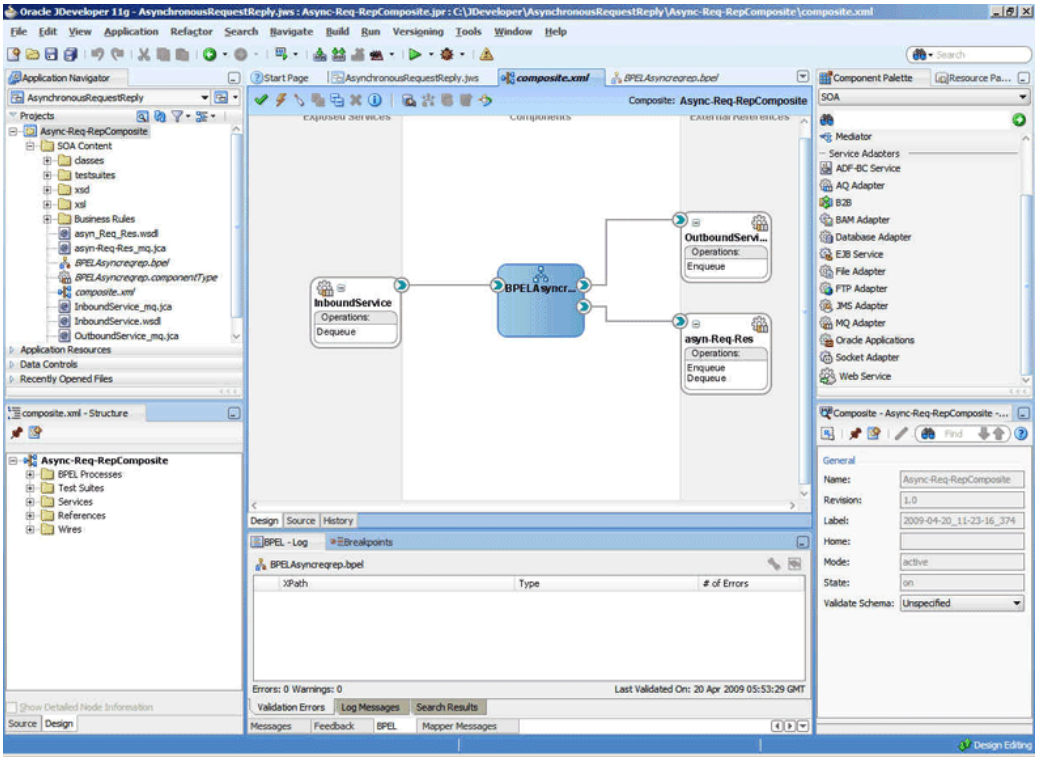
10.6.4.6 Wiring Services and Activities

You have to assemble or wire the four components that you have created: Inbound adapter service, BPEL process, async-Req-Res, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the InboundService service in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in async-Req-Res in the External References area.
3. Similarly, drag the small triangle in the BPEL process in the Components area to the drop zone in OutboundService in the External References area.

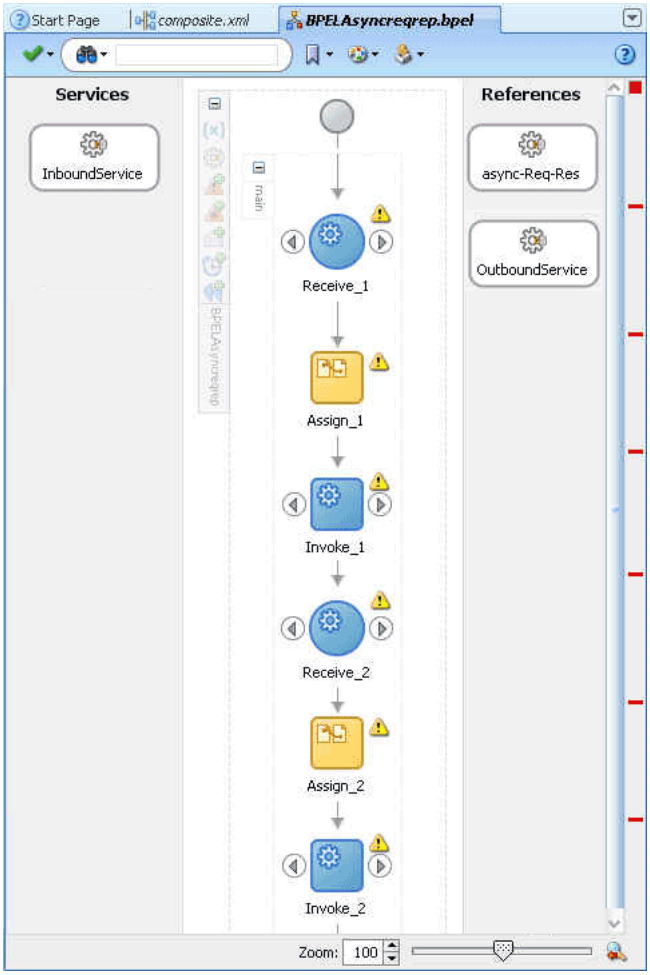
   The JDeveloper Composite.xml appears, as shown in Figure 10–78.

Figure 10–78 The JDeveloper - Composite.xml



4. Click **File**, **Save All**.
5. Double-click **BPELAsyncreqrep**. The BPELAsyncreqrep.bpel page is displayed.
6. Drag and drop the **Receive**, **Assign**, **Invoke**, **Receive**, **Assign**, **Invoke** activities in the order mentioned from the Component Palette to the Components area. The JDeveloper BPELAsyncreqrep.bpel page is displayed, as shown in Figure 10–79.

Figure 10–79 The BPELAsyncreqrep.bpel Page



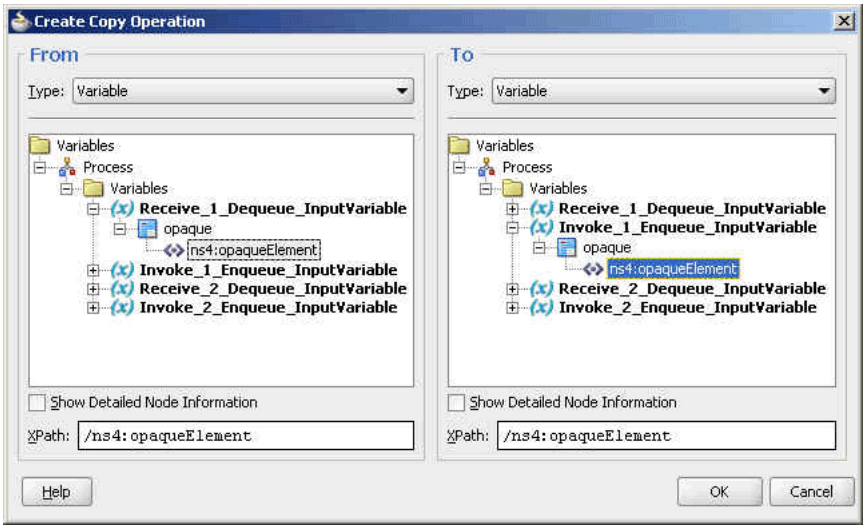
7. Drag and drop the first **Receive** activity to the InboundService adapter service. The Receive dialog is displayed.
8. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The Create Variable dialog is displayed.
9. Accept the defaults, and click **OK**.
10. Check the **Create Instance** box, and click **OK**.
11. Drag and drop the first **Invoke** activity to the async-Req-Res service. The Invoke dialog is displayed.
12. Click the **Automatically Create Input Variable** icon that appears at the end of the Input Variable field.
13. Accept the defaults, and click **OK**. The Invoke dialog is displayed.
14. Click **OK**.
15. Drag and drop the second **Receive** activity to the async-Req-Rep service. The Receive dialog is displayed.
16. Click the **Auto Create Variable** icon to create variable.

Note: Do not check the Create Instance box.

---

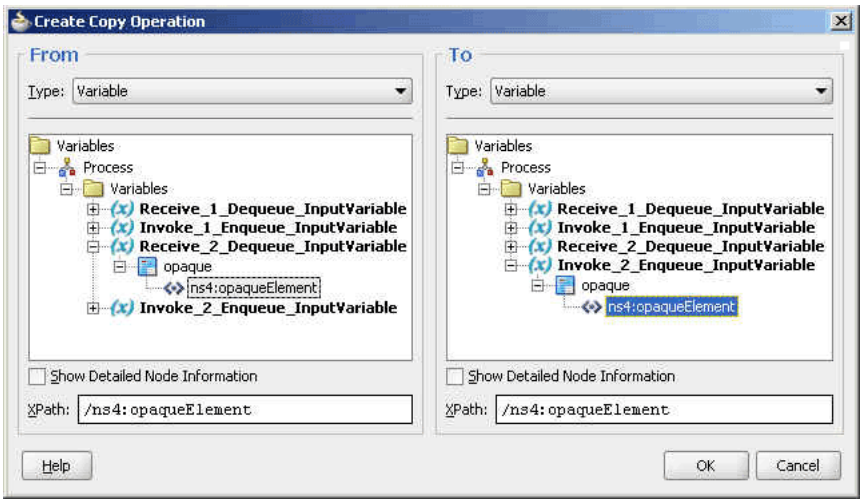
17. Click **OK** in the Receive dialog.
18. Drag and drop the second **Invoke** activity to OutboundService. The Invoke dialog is displayed.
19. Click the **Automatically Create Input Variable** icon to create a variable.
20. Click **OK** in the Invoke dialog.
21. Double-click the first **Assign** activity. The Assign dialog is displayed.
22. Click the plus icon, and select **Copy Operation**. The Create Copy Operation dialog is displayed.
23. Select the variables, as shown in Figure 10–80, and click **OK**.

Figure 10–80 The Create Copy Operation Dialog



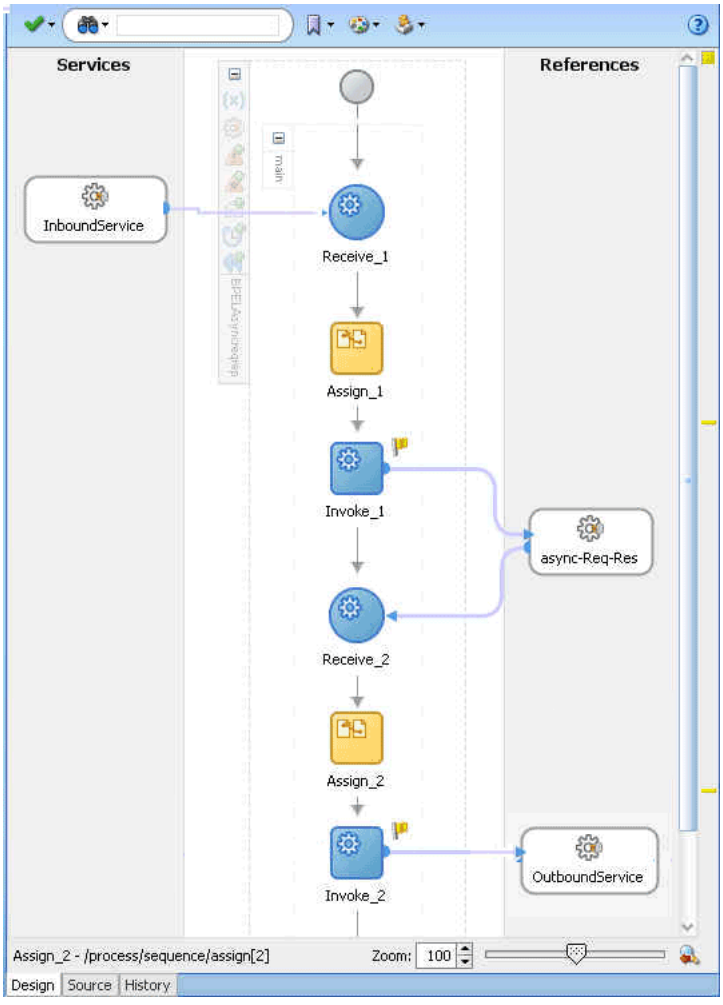
24. Click **OK** in the Assign dialog.
25. Double-click the second **Assign** activity. The Assign dialog is displayed.
26. Click the plus icon, and select **Copy Operation**. The Create Copy Operation dialog is displayed.
27. Select the variables, as shown in Figure 10–81, and click **OK**.

Figure 10–81 The Create Copy Operation Dialog



28. Click **OK** in the Assign dialog. The JDeveloper BPELAsyncreqrep.bpel page is displayed, as shown in Figure 10–82.

Figure 10–82 The BPELAsyncreqrep.bpel Page



10.6.4.7 Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper".

You must also create an application server connection. For more information about creating an application server connection, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters".

10.6.4.8 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Log in to *http://servername:portnumber/em* using your username/password. The Oracle Enterprise Manager Fusion Middleware Control page is displayed.
2. In the left pane, navigate to **SOA**, **soa-infra (soa_server1)**. A list of all the composites that are deployed appears.
3. Click **Async-Req-RepComposite[1.0]**. The Async-Req-RepComposite[1.0] page is displayed.
4. Put a message that has the content that conforms to the address-csv.xsd and also contains the Reply Queue as the header in the test_in queue.
5. Wait for some time and then refresh the Fusion Middleware Control Console. An instance appears on the console. This is the instance that was triggered because of the processing that occurred.
6. Click the **Instances** tab.
7. Click the instance associated with this deployment. The Flow Trace page is displayed.
8. Click the **BPELAsyncreqrep** component instance. The Audit Trail page is displayed.
9. Click the **Flow** tab to debug the instance. The BPEL process instance flow is displayed.
10. Click an activity to view the relevant payload details.

10.6.5 Outbound Dequeue

This use case is the end-to-end demonstration of how MQ Adapter dequeues a single message at a time. This section contains the following topics:

- Section 10.6.5.1, "Prerequisites"
- Section 10.6.5.2, "Designing the SOA Composite"
- Section 10.6.5.3, "Creating an Outbound Dequeue Adapter Service"
- Section 10.6.5.4, "Wiring Services and Activities"
- Section 10.6.5.5, "Deploying with JDeveloper"
- Section 10.6.5.6, "Monitoring Using the Fusion Middleware Control Console"

10.6.5.1 Prerequisites

To perform the outbound dequeue use case, you require the following files from the `Adapters-101MQAdapterDequeueEnqueue` sample:

- `De-queueEn-queue/De-queueEn-queueComposite/xsd/singleString.xsd`

You also require the following files from the `artifacts.zip` file contained in the `Adapters-101MQAdapterDequeueEnqueue` sample:

- `artifacts/input/data.txt`

You can onbtain the `Adapters-101MQAdapterDequeueEnqueue` sample by accessing the Oracle SOA Sample Code site.

You must also create a queue named `test_out`.

10.6.5.2 Designing the SOA Composite

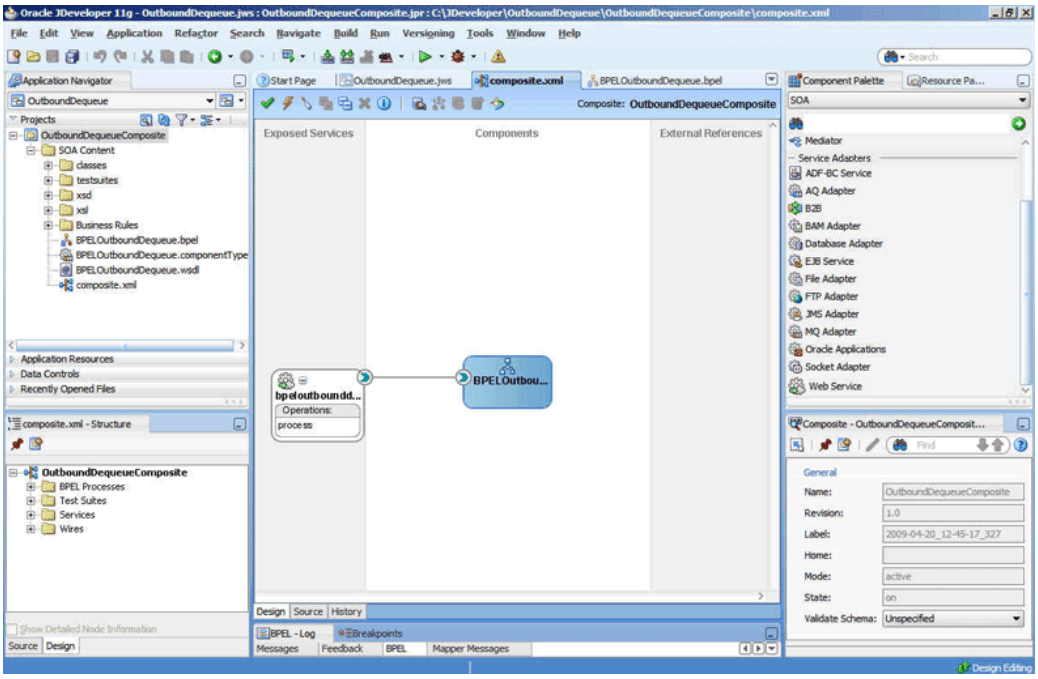
You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.

2. Expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.
4. In the Name Your Application screen, enter `OutboundDequeue` in the **Application Name** field, and then click **Next**. The Name Your Project screen is displayed.
5. In the **Project Name** field, enter `OutboundDequeueComposite` and from the **Available** list, select SOA and click the right-arrow button.
6. Click **Next**. The Configure SOA Settings screen is displayed.
7. In the Composite Template list, select **Composite With BPEL** and then click **Finish**. The Create BPEL Process dialog is displayed.
8. Enter `BPELOutboundDequeue` in the **Name** field, select **Synchronous BPEL Process** in the Template box.
9. Click **Browse** at the end of the Input field. The Type Chooser dialog is displayed.
10. Select **Project Schema Files**, **singleString.xsd**, **singleString**, and then click **OK**.
11. Click **Browse** at the end of the Output field. The Type Chooser dialog is displayed.
12. Select **Project Schema Files**, **singleString.xsd**, **singleString**, and then click **OK**.
13. Click **OK**. The OutboundDequeue application and OutboundDequeueComposite project appears in the design area, as shown in Figure 10–83.

Figure 10–83 The JDeveloper - Composite.xml

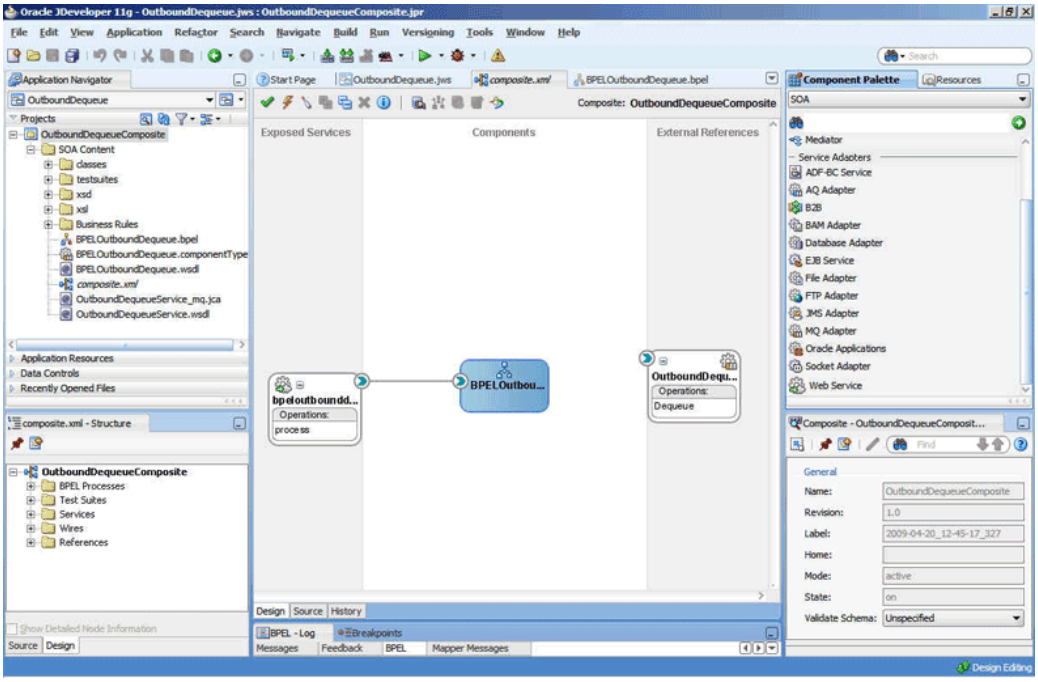


10.6.5.3 Creating an Outbound Dequeue Adapter Service

Perform the following steps to create an adapter service that dequeues the message to a queue:

1. Drag and drop **MQ Adapter** from the Component Palette into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `OutboundDequeueService` in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Get Message from MQ** and **Synchronous**, and click **Next**. The Get Message from MQ page is displayed.
7. Enter `test_out` in the **Queue Name** field and enter `10` in the **Wait Interval** field, and then click **Next**. The Messages page is displayed.
8. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
9. Select **Project Schema Files**, **singleString.xsd**, and then **singleString**, and click **OK**. The singleString.xsd file appears in the URL field in the Messages page.
10. Click **Next**. The Finish page is displayed.
11. Click **Finish**. You have now configured the inbound adapter service, and the composite.xml page is displayed, as shown in Figure 10–84.

Figure 10–84 The JDeveloper Page - Composite.xml Page



12. Click **File**, **Save All**.

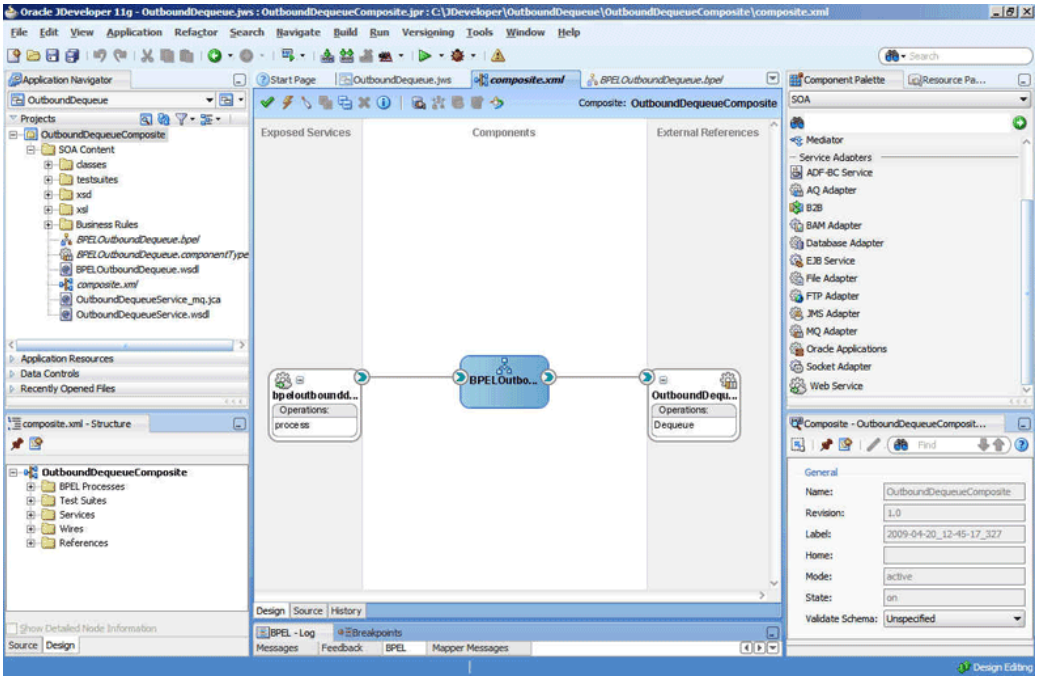
10.6.5.4 Wiring Services and Activities

You have to assemble or wire the three components that you have created: Client, BPEL process, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in OutboundDequeueService in the External References area.
2. Double-click **BPELOutboundDequeue**. The BPELOutboundDequeue.bpel page is displayed.
3. Drag and drop the **Invoke** and **Assign** activities in the order mentioned from the Component Palette to the Components area in between the receiveInput and replyOutput activities.

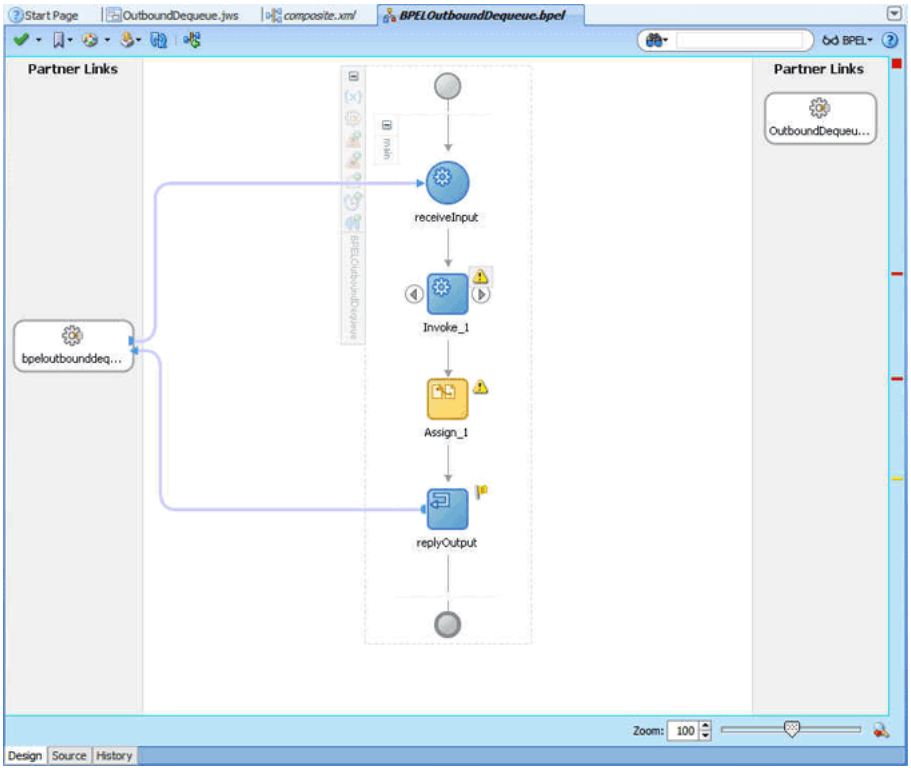
   The composite.xml page is displayed, as shown in Figure 10–85.

Figure 10–85 The JDeveloper - Composite.xml Page



The JDeveloper BPELOutboundDequeue.bpel page is displayed, as shown in Figure 10–86.

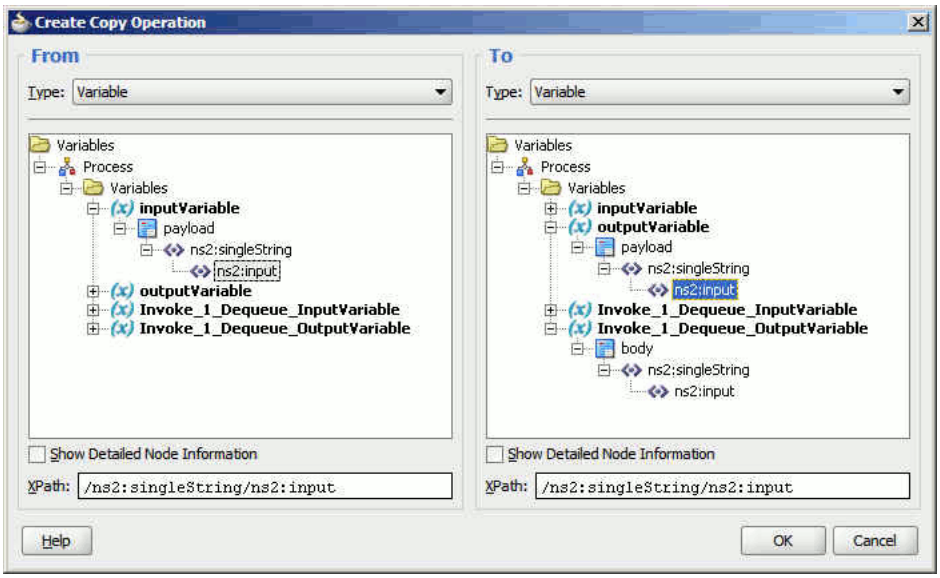
Figure 10–86 The BPELOutboundDequeue.bpel Page



4. Drag and drop the **Invoke** activity to the OutboundDequeueService adapter reference. The Invoke dialog is displayed.

5. Click the **Auto Create Variable** icon that appears at the end of the Input Variable field. The Create Variable dialog is displayed.
6. Accept the defaults, and click **OK**.
7. Repeat the same for the output variable and click **OK**.
8. Double-click the **Assign** activity. The Assign dialog is displayed.
9. Click the plus icon and select Copy Operation. The Create Copy Operation dialog is displayed.
10. Select the variables, as shown in Figure 10–87, and then click **OK**.

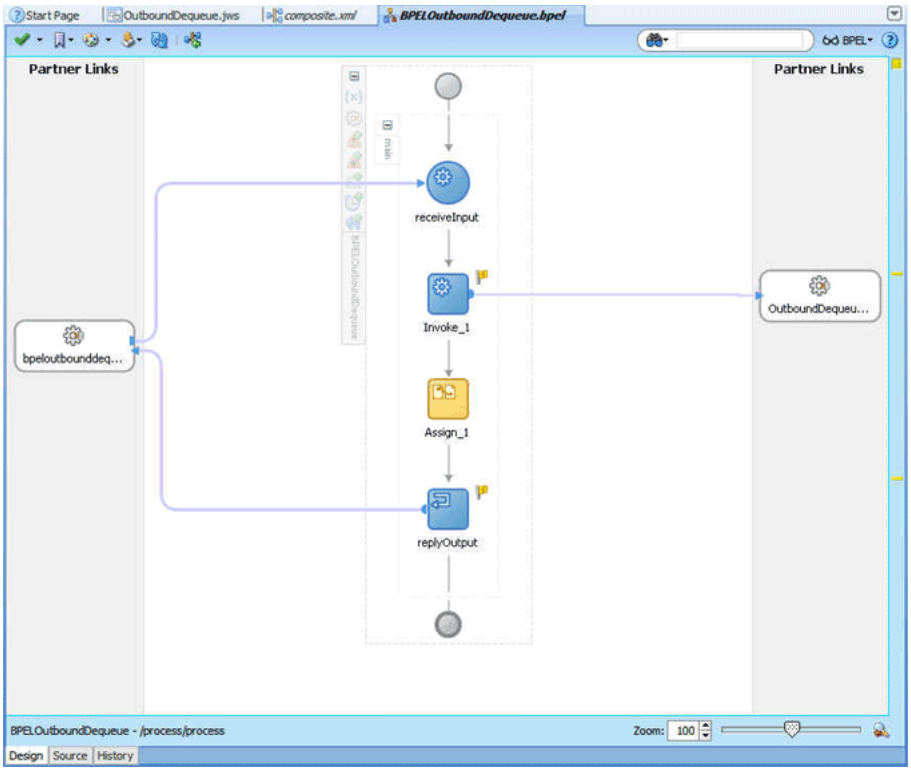
Figure 10–87 Create Copy Operation Dialog



11. Click **OK** in the Assign dialog.

    The BPELOutboundDequeue.bpel page appears, as shown in Figure 10–88.

Figure 10–88 The BPELOutboundDequeue.bpel Page



10.6.5.5 Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper".

You must also create an application server connection. For more information about creating an application server connection, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters".

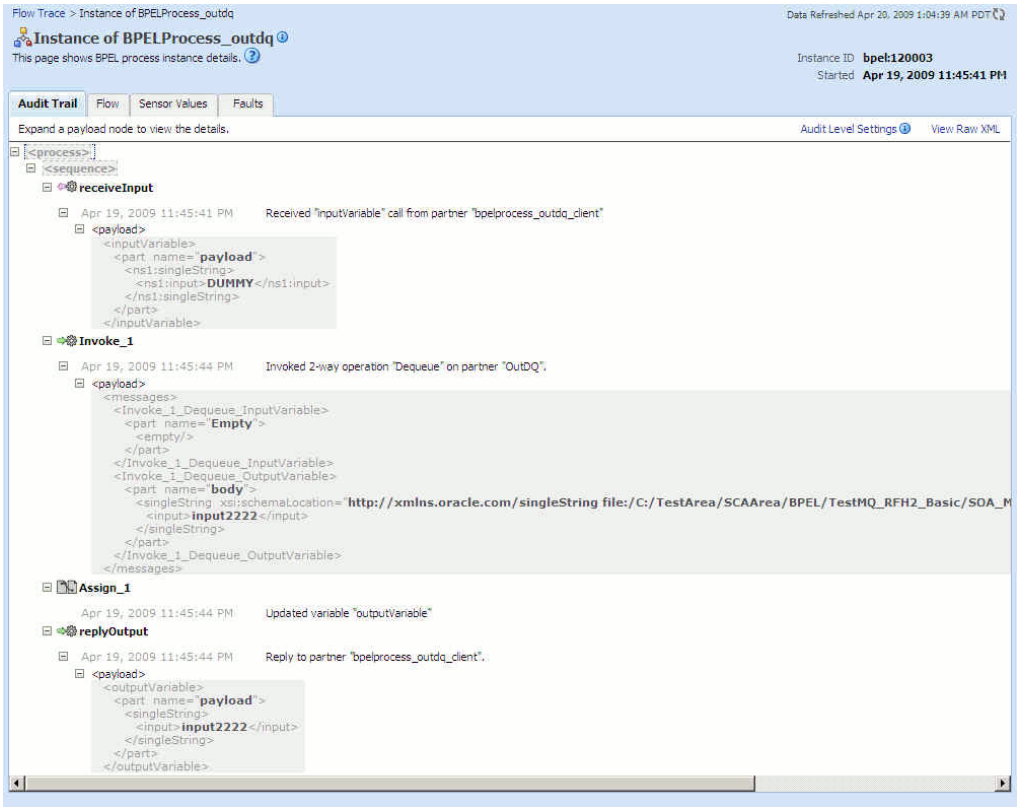
10.6.5.6 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Log in to *http://servername:portnumber/em* using your username/password.
2. In the left pane, navigate to **SOA**, **soa-infra (soa_server1)**. A list of all the composites that are deployed appears.
3. Click **OutboundDequeueComposite[1.0]**. The OutboundDequeueComposite[1.0] page is displayed.
4. Click the **Test** button. The Test Web Service page is displayed.
5. Click the **Request** tab, and scroll to the Input Arguments pane.
6. Enter `Test Outbound Dequeue` in the **Input** field, and then click the **Test Web Service** button.
7. Wait for some time and then click the **Response** tab. The message in the singleString xsd that you provided appears in the Response tab.

8. Click the **Instances** tab.
9. Click the instance associated with this deployment. The Flow Trace page is displayed.
10. Click the **BPELOutboundDequeue** component instance. The Audit Trail page is displayed, as shown in Figure 10–89.

Figure 10–89 Audit Trail Page



11. Click the **Flow** tab to debug the instance. The BPEL process instance flow is displayed.
12. Click an activity to view the relevant payload details.

10.6.6 Using RFH2 Headers

This use case demonstrates how to set the RFH2 header properties for a message to be added to a MQ Series queue. This is applicable only for outbound references. You must ensure that the format of the message is RFH2 and also set the fixed portions of the header properties. This section contains the following topics:

- Section 10.6.6.1, "Designing the SOA Composite"
- Section 10.6.6.2, "Creating an Inbound Adapter Service"
- Section 10.6.6.3, "Creating an Outbound Adapter Service"
- Section 10.6.6.4, "Wiring Services and Activities"
- Section 10.6.6.5, "Deploying with JDeveloper"

10.6.6.1 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.

2. Expand the **General** node, and select the **Applications** category.

3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.

4. In the Name your application screen, enter the name of your application in the **Application Name** field, and then click **Next**. The Name your project page is displayed.

5. In the **Project Name** field, enter the project name and from the **Available** list, select **SOA** and click the right-arrow button.

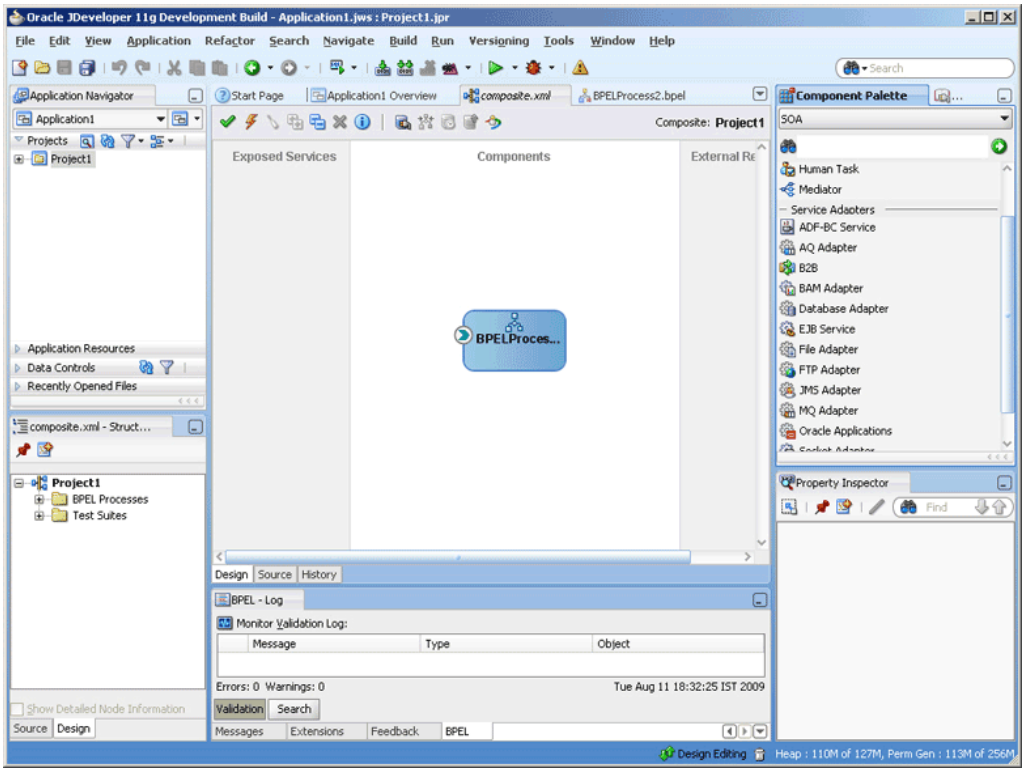
6. Click **Next**. The Configure SOA Settings page is displayed.

7. In the Composite Template list, select **Composite With BPEL**, and then click **Finish**. The Create BPEL Process page is displayed.

8. Enter a name for your BPEL process in the **Name** field, and select **One Way BPEL Process** from the Template box.

9. Click **OK**. Your application and project appears in the design area, as shown in Figure 10–42.

Figure 10–90 The JDeveloper - Composite.xml

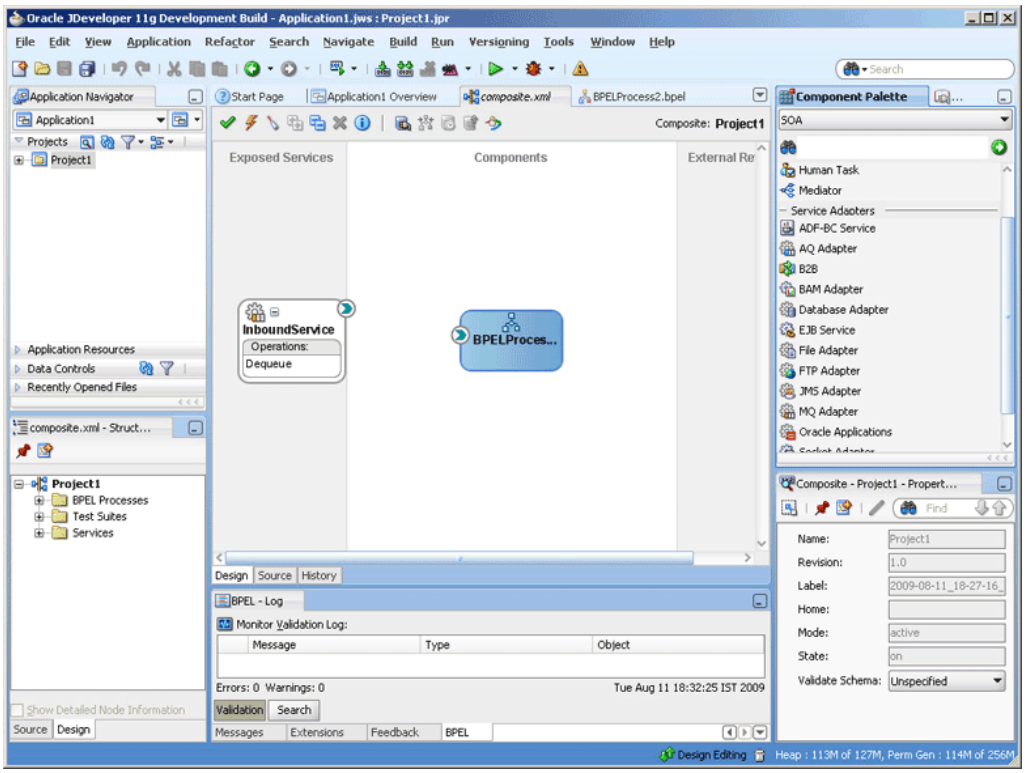


10.6.6.2 Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message from a queue:

1. Drag and drop **MQ Adapter** from the Component Palette into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter a service name in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Get Message from MQ**, and click **Next**. The Get Message from MQ page is displayed.
7. Enter a queue name in the **Queue Name** field, and click **Next**. The Messages page is displayed.
8. Select **Native format translation is not required (Schema is Opaque)**, and click **Next**. The Finish page is displayed.
9. Click **Finish**. You have now configured the inbound adapter service, and the composite.xml page is displayed, as shown in Figure 10–91.

Figure 10–91 The JDeveloper Page - Composite.xml Page



10.6.6.3 Creating an Outbound Adapter Service

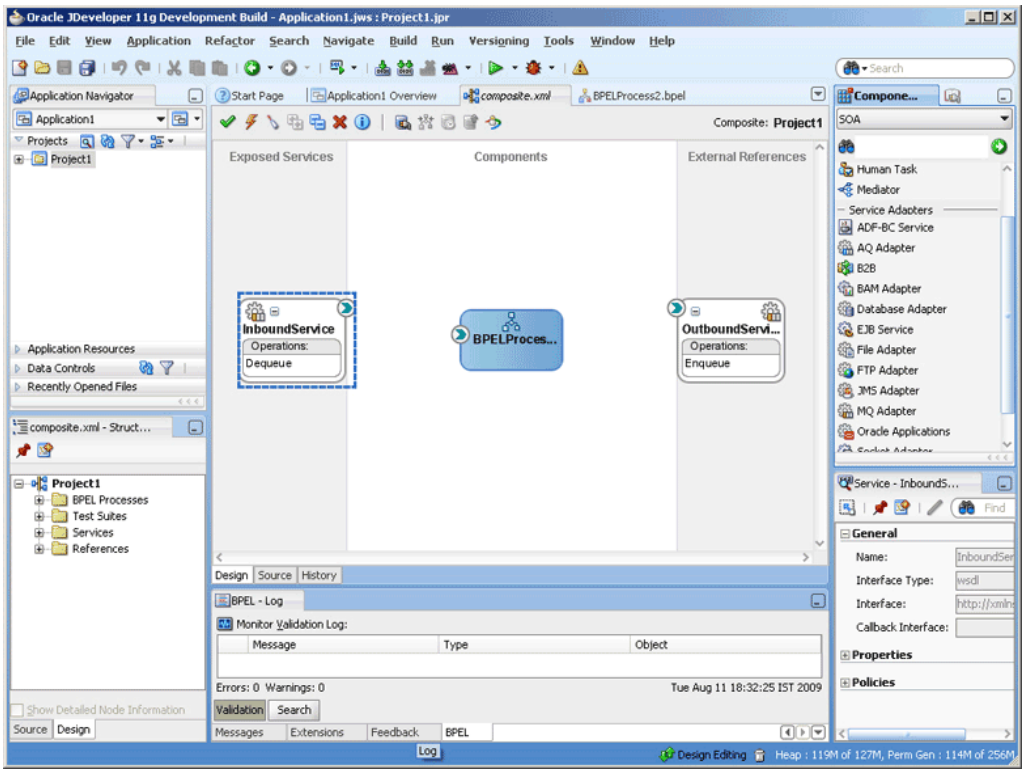
Perform the following steps to create an adapter service that enqueues the messages.

1. Drag and drop **MQ Adapter** from the Component Palette into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter the service name in the **Service Name** field, and click **OK**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Put Message into MQ**, and click **Next**. The Put Message into MQ page is displayed.
7. Enter a queue name in the **Queue Name** field, and click **Next**. The Advanced Options page is displayed.
8. Accept the defaults and click **Next**. The Messages page is displayed.
9. Select **Native format translation is not required (Schema is Opaque)**, and click **Next**. The Finish page is displayed.

> **Note:** You can also specify a sample schema that you must use if native format translation is required.

10. Click **Finish**. You have now configured the outbound adapter service, and the composite.xml page is displayed, as shown in Figure 10–92.

Figure 10–92 The JDeveloper Page - Composite.xml Page



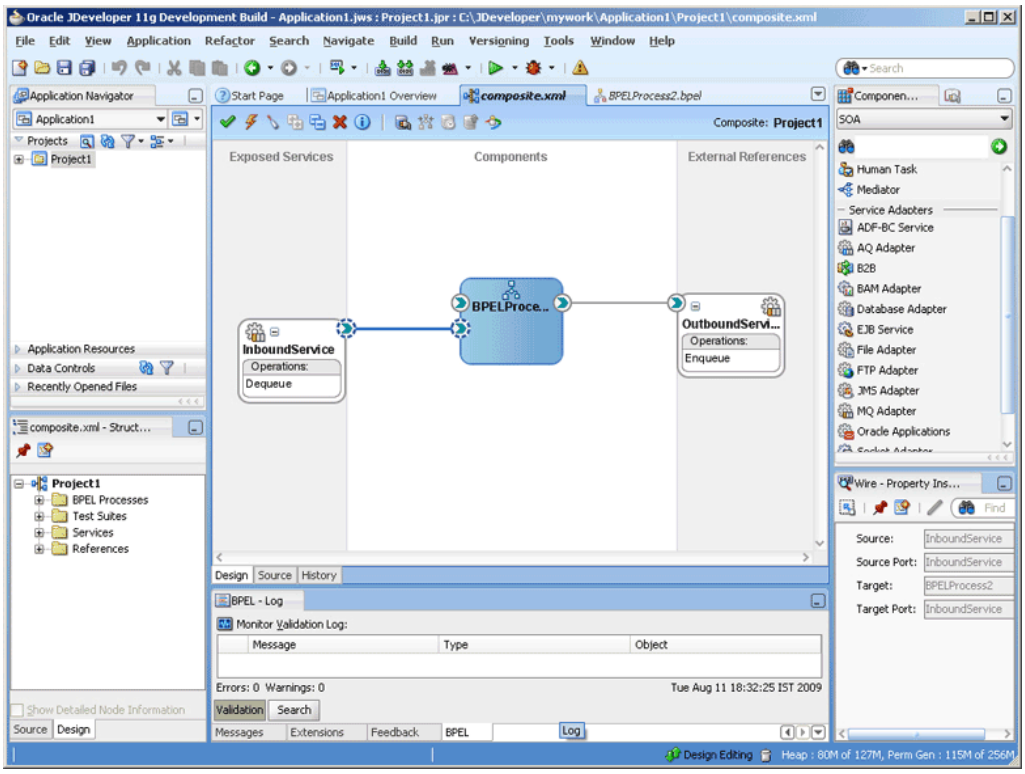
10.6.6.4 Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the InboundService in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in OutboundService in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 10–93.

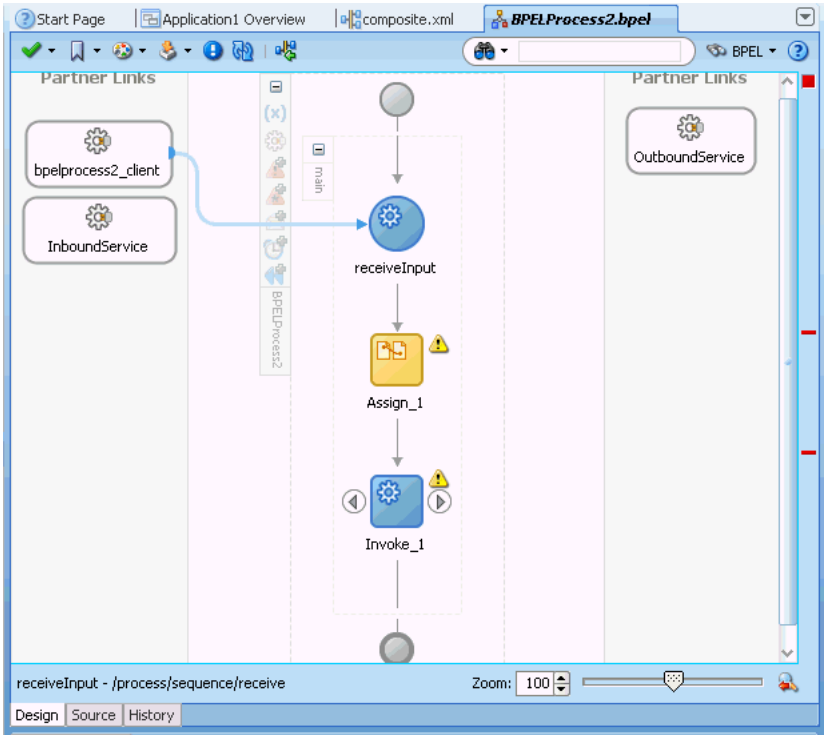
Figure 10–93 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.
4. Double-click the BPEL process. The *BPELprocess*.bpel page is displayed.
5. Drag and drop the **Assign**, and **Invoke** activities in the order mentioned from the Component Palette to the Components area.

   The JDeveloper *BPELprocess*.bpel page is displayed, as shown in Figure 10–54.

Figure 10–94 The BPELprocess.bpel Page



6. Create temporary variables to store the RFH2 header portions. Additionally, you must create a `messageFormat` variable, a `JMSFolder` variable, and an `MCDFolder` variable.

   a. Click the **Variables...** icon represented by (x). The Variables dialog is displayed.

   b. Click the **Create... (Alt+N)** icon. The Create Variable dialog is displayed.

   c. Enter a variable name in the **Name** field.

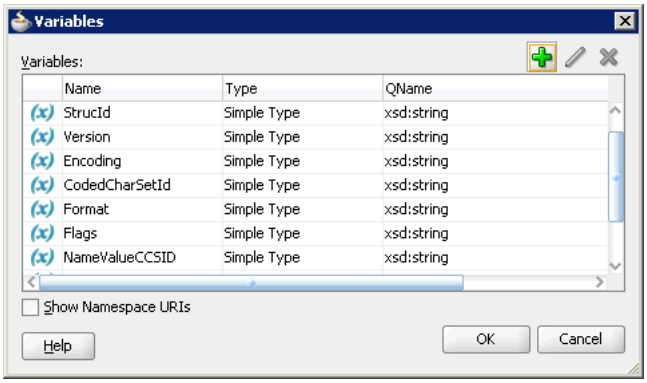
   d. Select **Simple Type** as the variable type.

   e. Click **Browse XML Schema Types** icon at the end of Simple Type field. The Type Chooser dialog is displayed.

   f. Choose **string** type.

   g. Click **OK**. A variable of type Simple Type is added to the Variables list in the variable dialog, as shown in Figure 10–95.

Figure 10–95 The Variable Dialog



h. Repeat Step a to g until all the required variables have been created.

7. Assign values to variables created in Step 6 by performing the following steps:

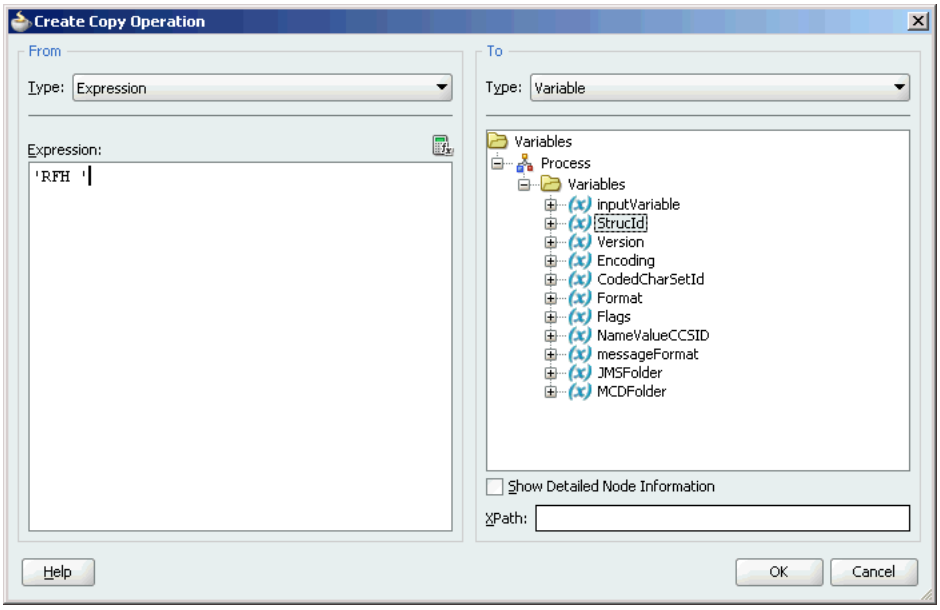
a. Double-click the assign activity. The Assign dialog is displayed.

b. Click the **Copy Operation** tab. The Assign dialog is displayed.

c. Select **Copy Operation**. The Create Copy Operation dialog is displayed.

d. In the Create Copy Operation dialog, select **Expression** from Type and specify the value and select the variable in the To pane to which the copy operation is being created, as shown in Figure 10–96.

Figure 10–96 The Create Copy Operation Dialog



The following is a code snippet from the BPELProcess_JMSFolder.bpel file, with the copy operation defined:

```
<sequence name="main">
  <receive name="Receive_1" createInstance="yes"
           variable="Receive_1_Dequeue_InputVariable" partnerLink="DQ"
           portType="ns1:Dequeue_ptt" operation="Dequeue"/>
  <assign name="Assign_1">
    <copy>
```

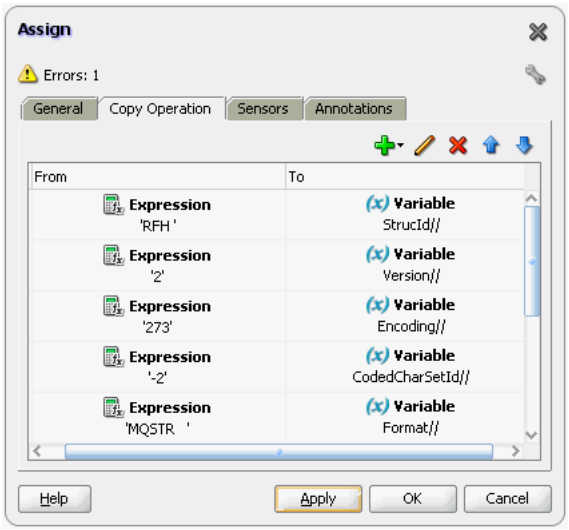
```
        <from variable="Receive_1_Dequeue_InputVariable" part="opaque"
              query="/ns3:opaqueElement"/>
        <to variable="Invoke_1_Enqueue_InputVariable" part="opaque"
            query="/ns3:opaqueElement"/>
      </copy>
      <copy>
        <from expression="'RFH '"/>
        <to variable="StructId"/>
      </copy>
      <copy>
        <from expression="'2'"/>
        <to variable="Version"/>
      </copy>
      <copy>
        <from expression="'273'"/>
        <to variable="Encoding"/>
      </copy>
      <copy>
        <from expression="'-2'"/>
        <to variable="CodedCharSetId"/>
      </copy>
      <copy>
        <from expression="'MQSTR   '"/>
        <to variable="Format"/>
      </copy>
      <copy>
        <from expression="'0'"/>
        <to variable="Flags"/>
      </copy>
      <copy>
        <from expression="'819'"/>
        <to variable="NameValueCCSID"/>
      </copy>
      <copy>
        <from expression="'RF_HDR_2'"/>
        <to variable="messageFormat"/>
      </copy>
      <copy>
        <from
expression="'&lt;jms>&lt;Dst>MYTOPIC&lt;/Dst>&lt;Exp>2000&lt;/Exp>&lt;Pri>4
&lt;/Pri>&lt;Cid>22344&lt;/Cid>&lt;Rto>REPLY.QUEUE&lt;/Rto>&lt;Gid>3334&lt;
/Gid>&lt;Seq>2&lt;/Seq>&lt;Dlv>1&lt;/Dlv>&lt;xxx>User
Space&lt;/xxx>&lt;/jms>'"/>
        <to variable="JMSFolder"/>
      </copy>
      <copy>
        <from expression="'&lt;mcd>&lt;Msd>jms_object&lt;/Msd>&lt;/mcd>'"/>
        <to variable="MCDFolder"/>
      </copy>
    </assign>
```

Note: The values for `StructId` and `Version` are fixed, as specified in the preceding code snippet. You can set the values for the other variables according to your requirements.

e. Repeat step b to d, until the copy operation is created for all the variables.

f. Click **OK** in the Create Copy Operation dialog. The Assign dialog is displayed as shown in Figure 10–97.

Figure 10–97 The Assign Dialog



g. Click **OK**. The BPELProcess.bpel page is displayed.

8. Set the header properties by using the following steps:

a. Double-click the invoke activity. The Invoke dialog is displayed.

b. Click the **Browse Partner Links** icon to the right of the Partner Link field. The Partner Link Chooser page is displayed.

c. Select the outbound service and click **OK**.

d. Click the **Auto-Create Variable** icon to the right of the Variable field under the General tab. The Create Variable dialog is displayed.

e. Select the default variable name and click **OK**. The Variable field is populated with the default variable name.

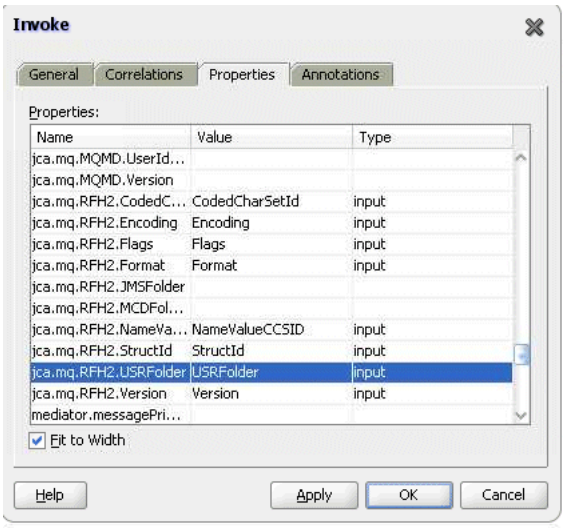
f. Click the **Properties** tab.

g. Select a property from the **Properties** column and set the **Value** accordingly.

For example, select the `jca.mq.MQMD.Format` property from the **Properties** column and set the **Value** as `MessageFormat`. Ensure that the **Type** column is set to `input`, as shown in Figure 10–98.

Note: if you want to create a user folder, then you must first assign the value to a `String` variable, then go to Properties in the Invoke activity and then map this `String` value to the property `jca.mq.RFH2.USRFolder`.

Figure 10–98 The Invoke Dialog



h. Click **OK**. The BPELProcess.bpel page is displayed.

i. Click the **Properties** tab. The BPELProcess.bpel page is displayed.

10.6.6.5 Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper".

You must also create an application server connection. For more information about creating an application server connection, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters".

10.6.7 Configuring a Backout Queue

This use case demonstrates how a backout queue must be configured for Oracle MQ Series Adapter. Oracle MQ Series Adapter dequeues a message and enqueues the same message after transformation from the MQ Series queue. During this process, a failure can occur either during an invoke activity or when a response is being sent. You must configure a Backout Queue to send the rejected messages to a Backout Queue instead of the default rejected messages folder. This section contains the following topics:

- Section 10.6.7.1, "Prerequisites"
- Section 10.6.7.2, "Designing the SOA Composite"
- Section 10.6.7.3, "Creating an Inbound Adapter Service"
- Section 10.6.7.4, "Creating an Outbound Adapter Service"
- Section 10.6.7.5, "Wiring Services and Activities"
- Section 10.6.7.6, "Deploying with JDeveloper"
- Section 10.6.7.7, "Monitoring Using the Fusion Middleware Control Console"

10.6.7.1 Prerequisites

To perform the use case for configuring a backout queue, you must ensure that the adapter JNDI is configured for XA. Also, you require the `singleString.xsd` file, which you can create using the following code:

```
<schema attributeFormDefault="unqualified"
      elementFormDefault="qualified"
      targetNamespace="http://xmlns.oracle.com/singleString"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="singleString">
            <complexType>
                  <sequence>
                        <element name="input" type="string"/>
                  </sequence>
            </complexType>
      </element>
</schema>
```

10.6.7.2 Designing the SOA Composite

You must create a JDeveloper application to contain the SOA composite. To create an application and a project for the use case, perform the following:

1. In JDeveloper, click **File** and select **New**.

   The New Gallery dialog is displayed.

2. Expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **Generic Application** and click **OK**. The Create Generic Application Wizard is displayed.
4. In the Name Your Application screen, enter `MQ_BackoutQ_Retry` in the **Application Name** field, and then click **Next**. The Name Your Project screen is displayed.
5. In the **Project Name** field, enter `SOA_BackoutQ_Retry` and from the **Available** list, select **SOA** and click the right-arrow button.
6. Click **Next**. The Configure SOA Settings screen is displayed.
7. In the Composite Template list, select **Composite With BPEL**, and then click **Finish**. The Create BPEL Process dialog is displayed.
8. Enter `BPELProcess_BackoutQ_Retry` in the **Name** field, and select **Define Service Later** from the Template box.
9. Click **OK**. The MQ_BackoutQ_Retry application and the SOA_BackoutQ_Retry project appears in the design area.

10.6.7.3 Creating an Inbound Adapter Service

Perform the following steps to create an adapter service that dequeues the message and put the message to a queue:

1. Drag and drop **MQ Adapter** from the Component Palette into the Exposed Services swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `InboundService` in the **Service Name** field, and click **Next**. The MQ Series Connection page is displayed.

4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Get Message from MQ and Send Reply/Reports** (Synchronous), and click **Next**. The Get Message from MQ and Send Rep page is displayed.
7. Enter `INBOUND_QUEUE` in the **Queue Name** field, and click **Next**. The Response page is displayed.
8. Accept the defaults, and click **Next**. The Messages page is displayed.
9. Click **Browse** at the end of the URL fields. The Type Chooser dialog is displayed.
10. Select **Project Schema Files**, **singleString.xsd**, and then **singleString**.
11. Click **OK**. The singleString.xsd file appears in the URL fields in the Messages page.
12. Click **Next**. The Finish page is displayed.
13. Click **Finish**. You have now configured the inbound adapter service, and the composite.xml page is displayed with an inbound adapter added.
14. Add the Backout Queue properties to the corresponding `JCA` file (`ReqReply_mq.jca`), as shown in the following sample:

```
<property name="BackoutQueueName" value="BACKOUT.QUEUE"/>
<property name="MaximumBackoutCount" value="5"/>
<property name="BackoutRetries" value="3"/>
```

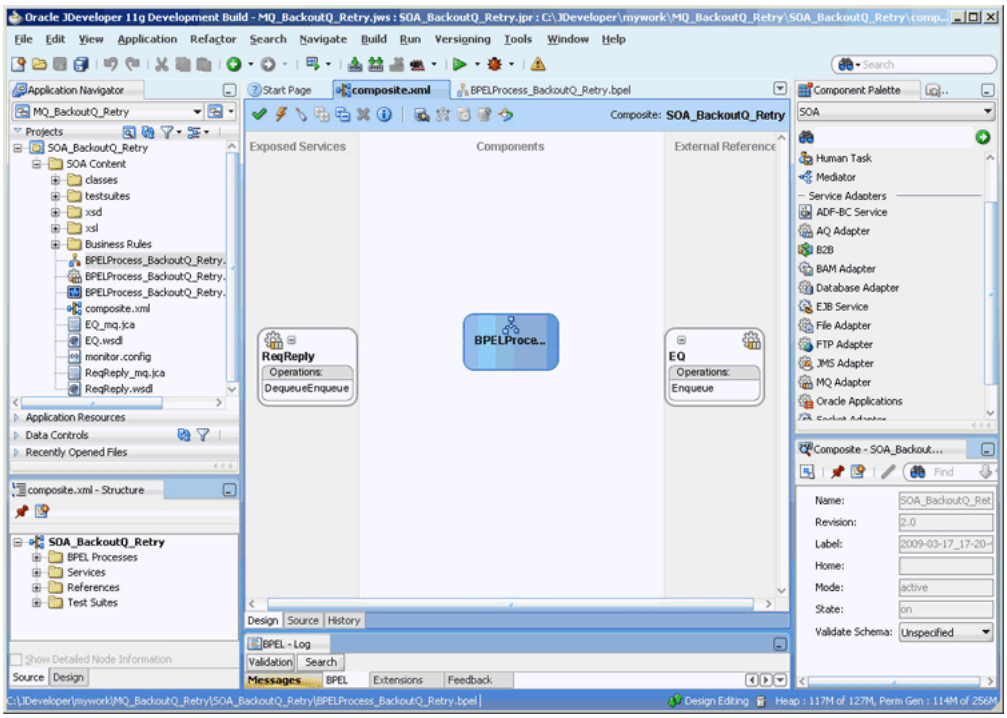
10.6.7.4 Creating an Outbound Adapter Service

Perform the following steps to create an adapter service that enqueues the messages.

1. Drag and drop **MQ Adapter** from the Component Palette into the External References swim lane. The Adapter Configuration Wizard Welcome page is displayed.
2. Click **Next**. The Service Name page is displayed.
3. Enter `EQ` in the **Service Name** field, and click **Next**. The MQ Series Connection page is displayed.
4. Accept the default JNDI name for the MQ Series connection, and click **Next**. The Adapter Interface page is displayed.
5. Select **Define from operation and schema (specified later)**, and click **Next**. The Operation Type page is displayed.
6. Select **Put Message into MQ**, and click **Next**. The Put Message into MQ page is displayed.
7. Enter a `test_out` in the **Queue Name** field, and click **Next**. The Advanced Options page is displayed.
8. Accept the defaults and click **Next**. The Messages page is displayed.
9. Click **Browse** at the end of the URL field. The Type Chooser dialog is displayed.
10. Select **Project Schema Files**, **singleString.xsd**, and then **singleString**, and click **OK**. The singleString.xsd file appears in the URL field in the Messages page.
11. Click **Next**. The Finish page is displayed.

12. Click **Finish**. You have now configured the outbound adapter service, and the composite.xml page is displayed, as shown in Figure 10–99.

Figure 10–99 The JDeveloper Page - Composite.xml Page



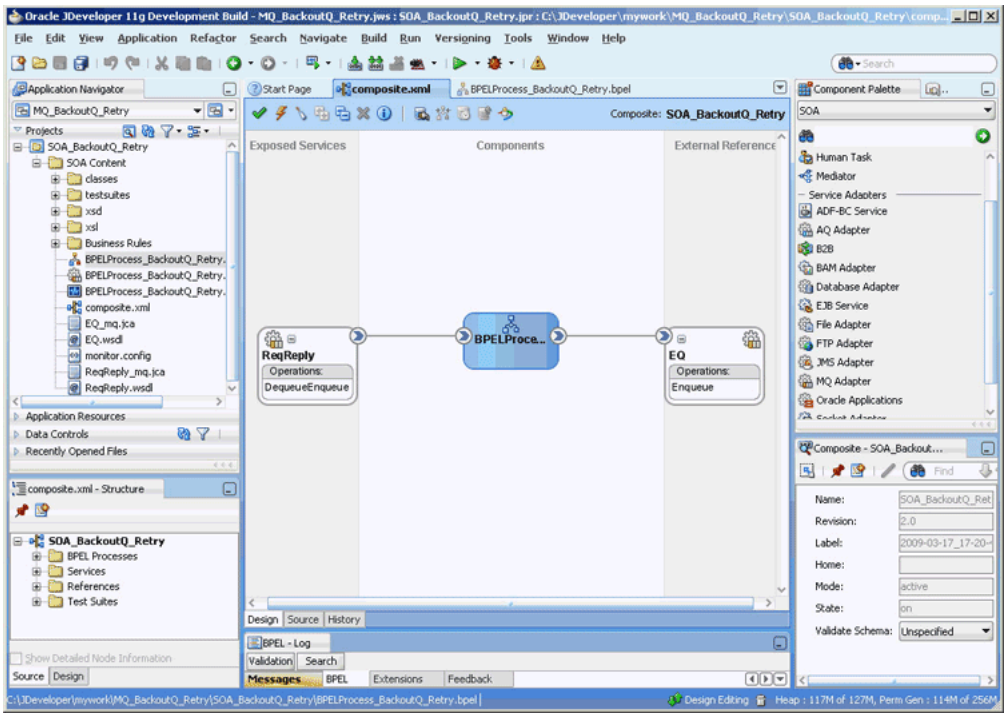
10.6.7.5 Wiring Services and Activities

You have to assemble or wire the three components that you have created: Inbound adapter service, BPEL process, and Outbound adapter reference. Perform the following steps to wire the components:

1. Drag the small triangle in the InboundService in the Exposed Services area to the drop zone that appears as a green triangle in the BPEL process in the Components area.
2. Drag the small triangle in the BPEL process in the Components area to the drop zone that appears as a green triangle in OutboundService in the External References area.

   The JDeveloper Composite.xml appears, as shown in Figure 10–100.

Figure 10–100 The JDeveloper - Composite.xml



3. Click **File**, **Save All**.
4. Double-click **BPELProcess_BackoutQ_Retry**. The BPELProcess_backoutQ_Retry.bpel page is displayed.
5. Drag and drop the **Receive**, **Assign**, **Invoke**, and **Reply** activities in the order mentioned from the Component Palette to the Components area.
6. Drag and drop the **Receive** activity to ReqReply. The Receive dialog is displayed.
7. Click the **Auto Create Variable** icon that appears at the end of the Variable field. The Create Variable dialog is displayed.
8. Accept the defaults, and click **OK**.
9. Check the **Create Instance** box, and click **OK**.
10. Drag and drop the **Reply** activity to ReqReply. The Reply dialog is displayed.
11. Enter `ReplyOutput` in the **Name** field.
12. Click the **Browse Variables** icon that appears at the end of the Variable field. The Variable Chooser dialog is displayed.
13. Select **replyOutput_DequeueEnqueue_OutputVariable**, and click **OK**. The variable appears in the Reply dialog.
14. Click **OK**.
15. Drag and drop the **Invoke** activity to EQ. The Invoke dialog is displayed.
16. Click the **Automatically Create Input Variable** icon that appears at the end of the Input Variable field.
17. Accept the defaults, and click **OK**. The Invoke dialog is displayed.
18. Click **OK**.

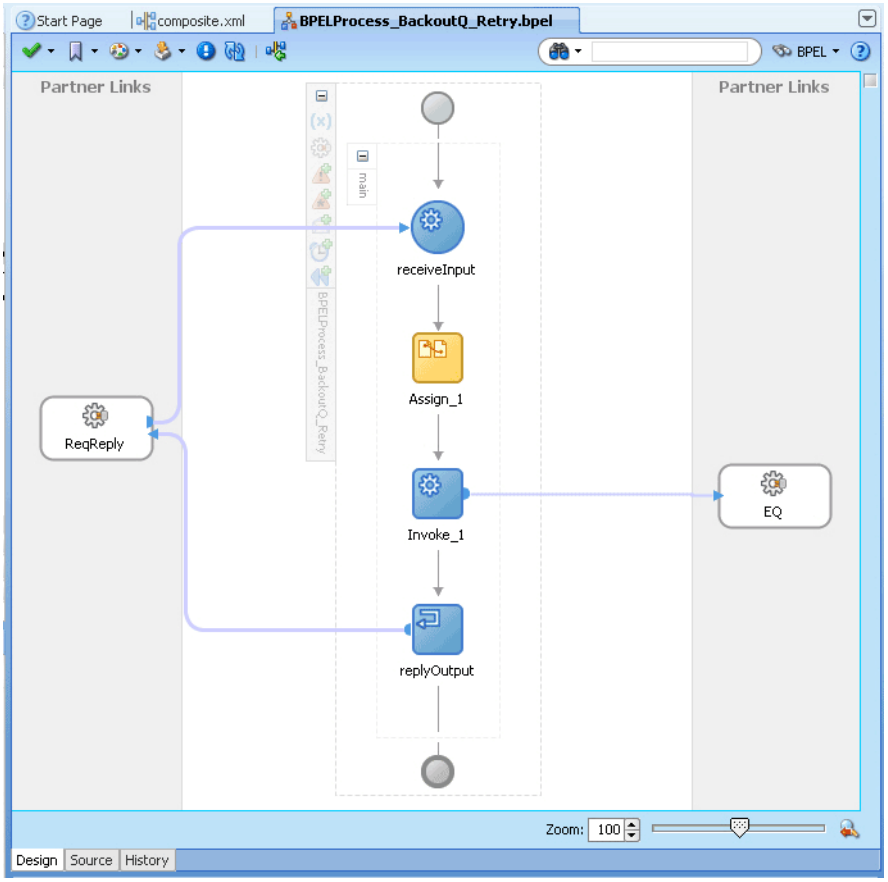
19. Double-click the **Assign** activity. The Assign dialog is displayed.
20. Click the plus icon, and select **Copy Operation**. The Create Copy Operation dialog is displayed.
21. In the Create Copy Operation dialog, select `receiveInput_DequeueEnqueue_InputVariable` as the From Type and select the variable in the To pane to which the copy operation is being created.

    The following is a code snippet from the BPELProcess_BackoutQ_Retry.bpel file, with the copy operation defined:

```
<assign name="Assign_1">
      <copy>
        <from variable="receiveInput_DequeueEnqueue_InputVariable"
              part="singleString" query="/ns3:singleString/ns3:input"/>
        <to variable="Invoke_1_Enqueue_InputVariable" part="body"
            query="/ns3:singleString/ns3:input"/>
      </copy>
      <copy>
        <from variable="receiveInput_DequeueEnqueue_InputVariable"
              part="singleString" query="/ns3:singleString/ns3:input"/>
        <to variable="replyOutput_DequeueEnqueue_OutputVariable"
            part="singleString" query="/ns3:singleString/ns3:input"/>
      </copy>
    </assign>
```

22. Click **OK**. The BPELdequeueenqueue.bpel page appears, as shown in Figure 10–101.

Figure 10–101 The BPELProcess_BackoutQ_Retry.bpel Page



10.6.7.6 Deploying with JDeveloper

You must deploy the application profile for the SOA project and application you created in the earlier steps.

For more information about deploying the application profile using JDeveloper, see Section 2.7, "Deploying Oracle JCA Adapter Applications from JDeveloper".

You must also create an application server connection. For more information about creating an application server connection, see Section 2.6, "Creating an Application Server Connection for Oracle JCA Adapters".

10.6.7.7 Monitoring Using the Fusion Middleware Control Console

You can monitor the deployed SOA composite using the Fusion Middleware Control Console. Perform the following steps:

1. Navigate to `http://servername:portnumber/em`. The composite you deployed appears in the application navigator.
2. Disable (Put Inhibit) the test_out queue or the reply queue that is provided with the Inbound message and then put a message to the `INBOUND_QUEUE`.
3. Wait for some time and then refresh the Fusion Middleware Control Console. Instances that are triggered because of the processing appear on the console.

Note: The number of instances that are triggered must be equal to `BackoutRetries + 1`.

4. Click the **Instances** tab.
5. Click an instance associated with this deployment. The Flow Trace page is displayed.
6. Click the **BPELProcess_BackoutQ_Retry** component instance. The Audit Trail page is displayed.
7. Click the **Flow** tab to debug the instance. The BPEL process instance flow is displayed.
8. Click an activity to view the relevant payload details.

A

Oracle JCA Adapter Properties

This appendix lists and describes the JCA and binding properties applicable Oracle JCA Adapters, and is meant to be used with the chapters in this book on the specific JCA Adapters, to assist in the configuration of the Adapters.

This appendix includes the following sections:

- Section A.1, "Oracle File and FTP Adapters Properties"
- Section A.2, "Oracle Socket Adapter Properties"
- Section A.3, "Oracle AQ Adapter Properties"
- Section A.4, "Oracle JMS Adapter Properties"
- Section A.5, "Oracle Database Adapter Properties"
- Section A.6, "Oracle MQ Series Adapter Properties"
- Section A.7, "Generic Oracle JCA Adapter Properties"

For more information, see Chapter 33, "Configuring Service and Reference Binding Components", in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

A.1 Oracle File and FTP Adapters Properties

This section describes the properties applicable to the Oracle File and FTP Adapters, including:

- Table A–1, " JCA Properties for Oracle File and FTP Adapters"
- Table A–2, " JCA Properties Specific to Oracle FTP Adapter"
- Table A–3, " Binding Properties for Oracle File and FTP Adapters"
- Table A–4, " Binding Properties Specific to Oracle FTP Adapter"
- Table A–5, " JCA Properties for Oracle File Adapter: Normalized Properties"
- Table A–6, " JCA Properties for Oracle FTP Adapter: Normalized Properties"

For properties applicable to all Oracle JCA Adapters, see Section A.7, "Generic Oracle JCA Adapter Properties".

For more information, see:

- Section 33.1.2.3, "Oracle File Adapter", in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*

- Section 33.1.2.4, "Oracle FTP Adapter", in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*

Table A–1 JCA Properties for Oracle File and FTP Adapters

Property	Description
`Append`	If this property is set to `true`, it causes Oracle File and FTP Adapters to append to a file on outbound. If the file does not exist, then a new file is created. The file name can either be specified in the `JCA` file for the outbound operation or in the `jca.file.FileName` header.
`AsAttachment`	If set to `true`, it causes the inbound file to be published as an attachment.
`BatchSize`	Set it to the batch size for the batching transformation.
`CharacterSet`	Set it to the character set for the attachment. This parameter is not used internally by the Oracle File and FTP Adapters, and it is meant for third party applications that process the attachments published by the Oracle File and FTP Adapters.
`ChunkSize`	Set it to the chunk size for the chunked interaction operation.
`ConcurrentThreshold`	The maximum number of translation activities that can be allowed to execute in parallel for a particular outbound scenario. The translation step during the outbound operation is CPU-intensive and must be guarded because it might cause other applications or threads to starve. The maximum value is `100`.
`ContentType`	Set it to the mime-type of the attachment. This parameter is not used internally by the Oracle File and FTP Adapters, and it is meant for third party applications that process the attachments published by the Oracle File or FTP Adapter.
`DeleteFile`	If set to `true`, the Oracle File or FTP Adapter deletes the file after processing.
`DirectorySeparator`	When you choose multiple directories, the generated JCA files use semicolon (`;`) as the separator for these directories. However, you can change the separator to something else. If you do so, manually add `<property name="DirectorySeparator" value="`*chosen separator*`"/>` in the generated JCA file. For example, to use comma (`,`) as the separator, you must first change the separator to comma (`,`) in the Physical directory and then add `<property name="DirectorySeparator" value=",/>` in the JCA file.
`ElapsedTime`	This property is used for outbound batching. When the time specified elapses, the outgoing file is created. The parameter is of type `int` and is not mandatory. The default value is `1`.
`Encoding`	Set it to the encoding used for the attachment. This parameter is not used internally by the Oracle File and FTP Adapters, and it is meant for third party applications that process the attachments published by the Oracle File and FTP Adapters.
`ExcludeFiles`	This property specifies the pattern for types of files to be excluded during polling. The property is of type `String` and is not mandatory.
`FileName`	Use this parameter to specify a static single file name during the write operation.
`FileNamingConvention`	This property is used for the naming convention for the outbound write operation file.
`FileSize`	This property is used for outbound batching. The outgoing file is created when the file size condition is met. The parameter is of type `int` and is not mandatory. The default value is `1000` KB.

Table A–1 (Cont.) JCA Properties for Oracle File and FTP Adapters

Property	Description
`IncludeFiles`	This property specifies the pattern for types of files to pick up during polling. The parameter is of type `String` and is mandatory.
`Lenient`	If set to `true`, then the Orcle File Adapter does not complain if it does not have enough permission to read or write to the inbound directory. The default value of this property is `false`.
`ListSorter`	This property specifies the sorter that the Oracle File and FTP Adapters use to sort files in inbound. You can set this parameter to: `oracle.tip.adapter.file.inbound.listing.TimestampSorterAscending` to sort the file names by their modified time stamps in ascending manner or `oracle.tip.adapter.file.inbound.listing.TimestampSorterDescending` to sort the file names by their modified time stamps in descending manner
`LogicalArchiveDirectory`	This property specifies the logical directory in which to archive successfully processed files. The property is of type `String` and is not mandatory.
`LogicalDirectory`	This parameter specifies the logical input directory to be polled. The parameter is of type `String`.
`MaxRaiseSize`	This property specifies the maximum number of files that the Oracle File or FTP Adapter submits for processing in each polling cycle. For example, if the inbound directory has 1000 files and `MaxRaiseSize` is set to `100` and the polling frequency is one minute, then the Oracle File or FTP Adapter submits 100 files every minute.
`MinimumAge`	This parameter specifies the minimum age of files to be retrieved. This specification enables a large file to be completely copied into the input directory before it is retrieved for processing. The age is determined by the last modified time stamp. For example, if you know that it takes three to four minutes for a file to be written, then set the minimum age of pollable files to five minutes. If a file is detected in the input directory and its modification time is within five minutes of the current time, then the file is not retrieved because it is still potentially being written to.
`NumberMessages`	This property is used for outbound batching. The outgoing file is created when the number of messages condition is met. The parameter is of type `int` and is not mandatory. The default value is `1`.
`PhysicalArchiveDirectory`	This property specifies where to archive successfully processed files. The property is of type `String` and is not mandatory.
`PhysicalDirectory`	This property specifies the physical input directory or directories to be polled. The parameter is of type `String`. The inbound directory where the files appear is mandatory. You must specify the physical directory or logical directory.
`PollingFrequency`	This parameter specifies how often to poll a given input directory for new files. The parameter is of type `int` and is mandatory. The default value is 1 minute.
`PublishSize`	This property indicates whether the file contains multiple messages and how many messages to publish to the BPEL process at a time. The parameter is of type `int` and is not mandatory. The default value is `1`. For example, if a certain file has 11 records and this parameter is set to 2, then the file processes 2 records at a time and the final record is processed in the sixth iteration.
`Recursive`	If this property is set to `true`, then the adapter can process all the sub-directories under the main input directory recursively.

Table A–1 (Cont.) JCA Properties for Oracle File and FTP Adapters

Property	Description
SequenceName	Specifies the Oracle database sequence name to be used if you have configured the outbound Oracle File or FTP Adapter for High Availability.
SingleThreadModel	If the value is true, the Oracle File or FTP Adapter poller processes files in the same thread. In other words, it does not use the global in-memory queue for processing.
SourceFileName	The source file for the File I/O operation.
SourcePhysicalDirectory	The source directory for the File I/O operation.
SourceSchema	Set to the schema for the source file.
SourceSchemaRoot	Set to the root element name for the source file.
SourceType	Set this to native if the source file is native and xml if the source file is xml.
TargetFileName	The target file for the File I/O operation.
TargetPhysicalDirectory	The target directory for the File I/O operation.
TargetSchema	Set it to the schema for the target file.
TargetSchemaRoot	Set it to the root element name for the target file.
TargetType	Set this to native if the target file is native and xml if the source file is xml.
ThreadCount	If this property is available, then the adapter creates its own processor threads rather than depend on the global thread pool processor threads (by deafult, 4 of them). In other words, this parameter partitions the in-memory queue and each composite application gets its own in-memory queue. ■ If the ThreadCount property is set to 0, then the threading behavior equals that of the single-threaded model. ■ If the ThreadCount property is set to -1, then the global thread pool is used, which equals the default threading model. ■ The maximum value for the ThreadCount property is 40.
TriggerFile	The name of the trigger file that activates the inbound Oracle File or FTP Adapter.
TriggerFilePhysicalDirectory	The directory path where the Oracle File or FTP Adapter looks for the trigger files.
TriggerFileStrategy	This property defines the strategy that the Oracle File or FTP Adapter uses to look for the specified trigger file in the trigger file directory. The acceptable values are EndpointActivation, EveryTime, or OnceOnly.
Type	Set it to COPY, MOVE, or DELETE for the File IO interaction.
UseRemoteErrorArchive	This property defines where an error is archived during an Inbound Read. During an Inbound Read operation, if a malformed XML file is read, the malformed file results in an error. The errored file is by default sent to the remote file system for archival. The errored file can be archived at a local file system by specifying the useRemoteErrorArchive property in the jca file and setting that property to false. The default value for this property is true.

Table A–1 (Cont.) JCA Properties for Oracle File and FTP Adapters

Property	Description
`UseHeaders`	This parameter can be set to `true` or `false`. If you must read file headers and skip reading the payload while using inbound Oracle File or FTP Adapter, then set the `UseHeader` property to `true`. This is typically used in large payload scenarios where the inbound adapter is used as a notifier.
`UseStaging`	If set to `true`, then the outbound Oracle File or FTP Adapter writes translated data to a staging file, and later it streams the staging file to the target file. If set to `false`, then the outbound Oracle File or FTP Adapter does not use an intermediate staging file.
`Xsl`	Set it to the `xsl` transformer between the source and the target.

Table A–2 JCA Properties Specific to Oracle FTP Adapter

Property	Description
`FileType`	Set this property to either `ascii` or `binary` depending on the requirement.
`SourceIsRemote`	Set this property to `false` if you want to notify the Oracle FTP Adapter that the source for the IO operation is a local file system as opposed to a remote FTP server.
`TargetIsRemote`	Set this property to `false` if you want to notify the Oracle FTP Adapter that the target for the IO operation is a local file system as opposed to a remote FTP server.
`UseNlst`	Set this property to `true` if you want the Oracle FTP Adapter to use the `NLST` FTP command instead of the `LIST` command that the adapter uses by default.
`UseRemoteArchive`	Set this property to `true` to notify the Oracle FTP Adapter that the archival directory is on the same FTP server. If set to `false`, the Oracle FTP Adapter uses a local file system folder for archival.

Table A–3 Binding Properties for Oracle File and FTP Adapters

Property	Description
`ignoreListingErrors`	Lets you control the behavior of the inbound Oracle File Adapter during the polling operation. If set to `true`, the Oracle File Adapter does not complain if it cannot read from a nested folder.
`IgnoreZeroByteFile`	Set it to `true` if you do not want Oracle File and FTP Adapters to throw an exception during the outbound read operation, if the file could not be found. This property is ignored if the schema for the inbound file is anything other than `Opaque`.
`inMemoryTranslation`	This property is applicable only if `UseStaging` is set to `false` .If `UseStaging` is set to `true`, then the translation step occurs in-memory (that is, an in-memory byte array is created). If set to `false`, then the adapter creates an output stream to the target file (FTP, FTPS, and SFTP included) and allows the translator to translate and write directly to the stream.
`jca.message.encoding`	This property is used to override the encoding specified in the NXSD schema for inbound Oracle File and FTP Adapters.
`notifyEachBatchFailure`	Setting to `true` causes the Oracle File or FTP Adapter to call the Notification Agent's `onBatchFailure` every time an error occurs in a debatching scenario. If set to `false`, Oracle File or FTP Adapter call `onBatchFailure` only once after all messages are debatched.

Table A–3 (Cont.) Binding Properties for Oracle File and FTP Adapters

Property	Description
`oracle.tip.adapter.file.debatching.rejection.quantum`	This property lets you control the size of rejected messages for inbound Oracle File or FTP Adapter partner link. For example, if you set it to `100`, it causes the Oracle File or FTP Adapter to reject 100 lines from the file because the actual file is too large.
`oracle.tip.adapter.file.highavailability.maxRetry`	Number of times that inbound Oracle File and FTP Adapters retry to establish a database connection in distributed polling scenarios.
`oracle.tip.adapter.file.highavailability.maxRetryInterval`	Number of milliseconds after which inbound Oracle File and FTP Adapters retry to establish a database connection in distributed polling scenarios.
`oracle.tip.adapter.file.mutex`	Set it to the class name that specifies the mutex for the outbound write operation. This class must extend the `oracle.tip.adapter.file.Mutex` abstraction.
`oracle.tip.adapter.file.rejectOriginalContent`	Setting to `true` causes Oracle File or FTP Adapter to reject the original content. If set to `false`, the adapter rejects the XML data created because of the translation step.
`oracle.tip.adapter.file.timeout.recoverpicked.minutes`	This property is used by the inbound highly available adapter when using `FILEADAPTER_IN` as the coordinator. Remember that when a file is first claimed (enqueued) by a node for processing, the `FILE_PROCESSED` column in `FILEADAPTER_IN` is set to `0`. At a later point in time, when a decoupled processor threads picks up the file for processing, the value of the `FILE_PROCESSED` column is updated from `0` to `1`. And when the file is processed completely, the `FILE_PROCESSED` column is updated from `1` to `2`. However, if the processor thread picks up a file but the node crashes before processing the file, then the file is never processed. This property is used to *undo* the pick operation. The adapter does this by deleting the entries in the `FILEADAPTER_IN` table that have been picked up but not processed within the value specified here.
`oracle.tip.adapter.file.timeout.recoverunpicked.minutes`	This property is used by the inbound highly available adapter when using `FILEADAPTER_IN` as the coordinator. Remember that when a file is first claimed by a node for processing, `FILE_PROCESSED` column in `FILEADAPTER_IN` is set to `0`. Later on, when the decoupled-processor thread picks up the file for processing, the value of the `FILE_PROCESSED` column is updated from `0` to `1`. And when the file is processed completely, the `FILE_PROCESSED` column is updated from `1` to `2`. If the node crashes when the `FILE_PROCESSED` is still `0`, it would mean that the file is enqueued by a node (no other nodes can pick this one up). However, it also means that the decoupled processor threads have still not picked this one for processing. This property is used to *undo* the claim(`enqueue_ operation`.) The adapter does this by deleting the entries in the `FILEADAPTER_IN` table that have been claimed (for example, `FILE_PROCESSED =="0"`), but not picked up till now.
`recoveryInterval`	This property is used by the inbound adapter to configure the recovery interval in case of errors. For example, if the physical directory is nonexistent, then the adapter uses this value to perform periodic sleep or wakeup checks to check whether the physical directory has been created and is accessible.
`serializeTranslation`	If set to `true`, then the translation step is serialized using a semaphore. The number of permits for semaphore (guarding the translation step) comes from the `ConcurrentThreshold` property. If `false`, then the translation step occurs outside the semaphore.
`useFileSystem`	This property is used by inbound Oracle File or FTP Adapter during read-only polling in a clustered environment. Setting it to `true` causes the adapter to use the file system to store the metadata of the processed files. Setting it to `false` causes the adapter to use a database table.

Table A–4 Binding Properties Specific to Oracle FTP Adapter

Property	Description
`timestampOffset`	This property is used by the Oracle FTP Adapter to handle time zone issues, typically to convert the time difference between the FTP server and the system on which the Oracle FTP Adapter is running to millisecond.

Table A–5 JCA Properties for Oracle File Adapter: Normalized Properties

Property	Description
`jca.file.FileName`	This property specifies the name of the file read from the inbound directory or written to the outbound directory.
`jca.file.Directory`	This property specifies the name of the directory from which file is read from or written to.
`jca.file.Size`	This property specifies the size of the file published from the inbound Oracle File Adapter.
`jca.file.Batch`	This property is used to specify a unique identifier for the file being published from the inbound adapter.
`jca.file.BatchIndex`	If a file has multiple messages and de-batching is used, then this normalized property specifies the message (record) number from the same batch. In this case, the `jca.file.Batch` remains the same but `jca.file.BatchIndex` increments by one for each publish from the same batch.

Table A–6 JCA Properties for Oracle FTP Adapter: Normalized Properties

Property	Description
`jca.ftp.FileName`	This property specifies the name of the file read from the inbound directory or written to the outbound directory.
`jca.ftp.Directory`	This property specifies the name of the directory from which file is read from or written to.
`jca.ftp.Size`	This property specifies the size of the file published from the inbound Oracle File Adapter. The size can be zero.
`jca.ftp.Batch`	This property is used to specify a unique identifier for the file being published from the inbound Oracle FTP Adapter.
`jca.ftp.BatchIndex`	If a file has multiple messages and de-batching is used, then this normalized property specifies the message (record) number from the match batch. In this case, the `jca.ftp.Batch` remains the same but `jca.ftp.BatchIndex` increments by one for each publish from the same batch.

A.2 Oracle Socket Adapter Properties

This section describes the properties applicable to the Oracle Socket Adapter, including:

- Table A–7, " JCA Properties for Oracle Socket Adapter"

For properties applicable to all Oracle JCA Adapters, see Section A.7, "Generic Oracle JCA Adapter Properties".

For more information, see Section 33.1.2.7, "Oracle Socket Adapter", in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

Table A–7 JCA Properties for Oracle Socket Adapter

Property	Description
`ByteOrder`	Byte order of the remote computer being communicated with.
`CustomImpl`	If `CustomImpl` is chosen as the `TransMode` property, then it is the name of the Java class defining the handshake. This property is a concrete implementation of the ICustomParser interface.
`Encoding`	Character encoding used by the remote computer.
`Host`	In case of outbound, the computer name on which the socket server is running, to which you want to connect. In case of inbound, it is always `localhost`.
`Port`	In case of outbound, it is the port number on which a socket server is running, to which the adapter is connecting. In case of inbound, it is the port number on which the socket adapter listens for incoming connections.
`ReplyXslt`	If `XSLT` is chosen as the `TransMode` property, then it specifies the path to the style sheet defining the handshake for inbound reply.
`TransMode`	Mechanism for defining the protocol. Set to `XSLT` to use style sheets, set to `CustomImpl` to use custom Java code, and set to `NXSD` for plain schema translation.
`Xslt`	If `XSLT` is chosen as the `TransMode` property, then it specifies the path to the style sheet defining the handshake for inbound request, in case of inbound and outbound request or reply.

A.3 Oracle AQ Adapter Properties

This section describes the properties applicable to the Oracle AQ Adapter, including:

- Table A–8, " JCA Properties for Oracle AQ Adapter"
- Table A–9, " JCA Properties for Oracle AQ Adapter: Normalized Properties"
- Table A–10, " Binding Properties for Oracle AQ Adapter"

For properties applicable to all Oracle JCA Adapters, see Section A.7, "Generic Oracle JCA Adapter Properties".

For more information, see Section 33.1.2.1, "Oracle AQ Adapter", in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

Table A–8 JCA Properties for Oracle AQ Adapter

Property	Description
`QueueName`	The name of the AQ Queue being read from or written to.
`DatabaseSchema`	The schema where the queue resides. If not specified, the schema of the current connection is used.
`SchemaValidation`	When this property is set to `"true"`, the payload is validated against the schema specified. If the validation fails, then the message is rejected.
`EnableStreaming`	When this property is set to `"true"`, the payload from the queue is streamed instead of being processed as an in-memory DOM. You must use this feature while handling large payloads. This property is applicable when processing RAW messages, XMLType messages, and ADT type messages for which a payload is specified through an ADT attribute.
`RecipientList`	Specify the consumer name or names that are the intended recipients for the messages enqueued by the adapter. The message remains in the queue until all recipients have dequeued the message. If the field is left empty, then all the currently active consumers are recipients.

Table A–8 (Cont.) JCA Properties for Oracle AQ Adapter

Property	Description
`Consumer`	Applicable only for multiconsumer queues. If specified, only the messages targeted for the particular consumer are made available for processing.
`ObjectFieldName`	This property is used to identify the field containing the business payload if the queue is an ADT queue. You can specify an attribute of ADT to constitute a payload or an entire ADT to represent payload. In former case the 'ObjectFieldName' should be same as the attribute name of the ADT. In latter case this property is not specified.
`PayloadHeaderRequired`	Only applicable if the `ObjectFieldName` property specifies a value. If set to `"true"`, it ensures that all non payload attributes of ADT are available for processing as Normalized Message property `'jca.aq.HeaderDocument'`.
`MessageSelectorRule`	When a dequeue is performed from a multiconsumer queue, it is sometimes necessary to screen the messages and accept only those that meet certain conditions. These conditions can be based on payload or queue header values and is specified using `MessageSelectorRule` property of the adapter. These conditions may concern selecting messages of a certain priority, or some aspect of the message payload, such as in selecting only loan applications above $100,000.
`DequeueCondition`	This property is valid for dequeue operations only. Enter a Boolean expression similar to the `WHERE` clause of a SQL query. This expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions. If multiple messages satisfy the dequeue condition, then the order of dequeuing is indeterminate, and the sort order of the queue is not honored.
`Correlation`	You can assign an identifier to each message, thus providing a means to retrieve specific messages at a later time. The value to enter is agreed upon between the enqueuing sender and the dequeuing receiver for asynchronous conversations. This can be overridden on a per message basis through the normalized message property, `'jca.aq.Correlation'`. When specified for the dequeue operation, it only dequeues messages that match the value specified. If none is specified, all messages in the queue are available to the dequeue operation.
`PayloadSizeThreshold`	This property exposes a configurable control mechanism through which you can specify the payload size threshold in the adapter layer. The messages that have sizes beyond the configured threshold limit are rejected. If this property is not configured, it does not impose any restriction on the size of messages.

Table A–9 JCA Properties for Oracle AQ Adapter: Normalized Properties

Property	Description
`jca.aq.Attempts`	The number of failed attempts at dequeuing the message.
`jca.aq.Correlation`	User-assigned correlation ID.
`jca.aq.Delay`	The number of seconds after which the message is available for dequeuing.
`jca.aq.EnqueueTime`	The time at which the message was enqueued.
`jca.aq.ExceptionQueue`	The exception queue name.
`jca.aq.Expiration`	The number of seconds before the message expires. This parameter is an offset from the `Delay` parameter. Default value of `-1(never expires)` is used if not specified.
`jca.aq.MessageId`	The hexadecimal representation of the message ID for the dequeued message.
`jca.aq.OrigMessageId`	The hexadecimal representation of the original message ID.

Table A–9 (Cont.) JCA Properties for Oracle AQ Adapter: Normalized Properties

Property	Description
`jca.aq.Priority`	Priority of the message. A smaller number indicates a higher priority. The priority can be any number. The default value is zero.
`jca.aq.RecipientList`	The list of recipients for this message, separated by commas. This overrides `RecipientList` in `InteractionSpec`.
`jca.aq.HeaderDocument`	Contains string or DOM of current headers (XML DOM representation of payload headers.)

Table A–10 Binding Properties for Oracle AQ Adapter

Property	Description
`ConnectionRetryDelay`	The time for which the Oracle AQ Adapter waits before trying to re-create a connection after a connection is lost. The default value is 15s.
`DequeueTimeOut`	It is the interval after which the `dequeue()` API times out if no message is received on the inbound queue. The default value is 1s.
`adapter.aq.dequeue.threads`	Specifies the number of poller threads that are created when an endpoint is activated. The default value is 1.

A.4 Oracle JMS Adapter Properties

This section describes the properties applicable to the Oracle JMS Adapter, including:

- Table A–11, " JCA Properties for Oracle JMS Adapter"
- Table A–12, " JCA Properties for Oracle JMS Adapter: Normalized Properties"
- Table A–13, " Binding Properties for Oracle JMS Adapter"

For properties applicable to all Oracle JCA Adapters, see Section A.7, "Generic Oracle JCA Adapter Properties".

For more information, see Section 33.1.2.5, "Oracle JMS Adapter", in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*

Table A–11 JCA Properties for Oracle JMS Adapter

Property	Description
DestinationName	The name of the queue or topic being read from or written to.
`UseMessageListener`	The only value of this property supported in the current version is false. A value of `false` ensures that the JMS adapter uses synchronous mechanism to poll queues or topics for messages
`PayloadType`	This property specifies the type of JMS message that is being dequeued or enqueued by the adapter. For Map messages the value is `'MapMessage'` and for Text messages the value is `'TextMessage'`.
DurableSubscriber	Name used to identify a durable subscription. When working with durable subscriptions ensure that ClientID is also specified in addition to `DurableSubscriber` property. `ClientID` is specified as part of the `factoryProperties` property when defining a Managed Connection Factory instance. This property is only applicable when working with JMS Topic scenarios
MessageSelector	A string whose syntax is based on a subset of the SQL92 conditional expression syntax and lets you specify the messages adapter is interested in, by using header field references and property references. Only messages whose header and property values match the selector are delivered.

Table A–11 (Cont.) JCA Properties for Oracle JMS Adapter

Property	Description
PayloadEntry	Only applicable when dealing with messages of type `MapMessage`. This property is used to identify the field containing the business payload when dealing with Map messages. All other Map message entries are made available as normalized message properties accessed using `jca.jms.Map.`*`MapMessage entry name`*
AttachmentList	Only applicable when dealing with message of type `MapMessage`. This property is used to identify the field containing the business payload when dealing with Map messages. Also, the payload in this case is published as an attachment. You can use either `PayloadEntry` or `AttachmentList`. All other Map message entries are made available as normalized message properties accessed using `jca.jms.Map.`*`MapMessage entry name`*
`RequestDestinationName`	This property is applicable for a synchronous request-reply scenario and specify the name of destination for sending a message.
ReplyDestinationName	This property is applicable for a synchronous request-reply scenario and specify the name of destination for receiving a reply.
AllowTemporaryReplyDestination	This property is applicable for a synchronous request-reply scenario. When set to `true` `ReplyDestinationName` is not required and JMS adapter in turn uses a temporary destination to recieve a reply from.
EnableStreaming	When this property is set to `"true"`, the payload from the queue or topic is streamed instead of being processed as an in-memory DOM. You must use this feature while handling large payloads.
DeliveryMode	Represents the delivery mode to use. The message producer's default delivery mode is `PERSISTENT`. This can be overridden on a per message basis using normalized message property `jca.aq.JMSDeliveryMode`
TimeToLive	Represents the message's lifetime (in milliseconds). The message producer's default time to live is unlimited; the message never expires. A value of `0` signifies that the message never expires.
PayloadSizeThreshold	This property exposes a configurable control mechanism through which you can specify the payload size threshold in the adapter layer. The messages that have sizes beyond the configured threshold limit are rejected. If this property is not configured, it does not impose any restriction on the size of messages
Priority	Represents priority for this message. The message producer's default priority is 4. This can be overridden on a per message basis using normalized message property `jca.aq.JMSPriority`

Table A–12 JCA Properties for Oracle JMS Adapter: Normalized Properties

Property	Description
`jca.jms.JMSDestinationName`	This property specifies the destination to which the message is sent, and is set by the JMS producer.
`jca.jms.JMSDestinationProperties`	This property represents the properties that define the context used to look up the destination object to which the message must be sent
`jca.jms.JMSCorrelationID`	This property is set by both producers and consumers for linking the response message with the request message. This is an optional attribute.
`jca.jms.JMSType`	This property specifies the JMS message type.
`jca.jms.JMSReplyTo`	This is an optional attribute that indicates the destination to which a message reply must be sent.

Table A–12 (Cont.) JCA Properties for Oracle JMS Adapter: Normalized Properties

Property	Description
`jca.jms.JMSPriority`	This property is used by the consumer to set a priority number between 0 and 9. Larger numbers represent a higher priority.
`jca.jms.JMSExpiration`	This property specifies the duration of the message before the expiration. When a message's expiration time is reached, the JMS provider should discard it.
`jca.jms.JMSDeliveryMode`	This property is set to `persistent` or `nonpersistent` mode by the JMS client.
`jca.jms.JMSMessageID`	This property is used to specify a unique message identifier. The exact scope of uniqueness is provider-defined.
`jca.jms.JMSRedelivered`	This property is used as an indication of whether a message is being re-delivered. If a client receives a message with the JMSRedelivered field set, it is likely, but not guaranteed, that this message was delivered earlier but that its receipt was not acknowledged at that time.
`jca.jms.JMSTimestamp`	This property is used to specify the time when the message was handed off to the JMS provider to be sent.
`jca.jms.JMSProperty.`*`name`*	This property represents any custom (application-specific) properties of the message. The supported properties conforms to the one allowed according to JMS specification. If an invalid property value is specified, the adapter warns the user (captured in the log files) and ignore the invalid property.
`jca.jms.Map.`*`name`*	This property represents any MapMessage element that is not transferred as payload.

Table A–13 Binding Properties for Oracle JMS Adapter

Property	Description
`adapter.jms.encoding`	Used to encode inbound text messages. This property is superseded by the newly supported property called `jca.message.encoding` that is applicable for both inbound and outbound messages.
`adapter.jms.receive.threads`	Specifies the number of poller threads that are created when an endpoint is activated. The default is 1.
`adapter.jms.receive.timeout`	Timeout value used for the synchronous receive call. It is the time after which receive() API times out if no message is received on the inbound queue. The default value is 1s.
`adapter.jms.registration.interval`	This property is not supported anymore.
`adapter.jms.retry.interval`	Used by the inbound connection retry layer. The time for which the Oracle JMS Adapter waits before trying to re-create a connection after a connection is lost. The default value is 30s.
`JMSReplyToDestinationProperties`	Declaratively impose custom property settings on Destination objects received during inbound request/reply scenarios.
`JMSReplyUseCorrelationIdForCorrelation`	Used to specify whether you want to use a correlation Id for correlation. Valid values are `true` and `false`. The default value is `false`.

Table A–13 (Cont.) Binding Properties for Oracle JMS Adapter

Property	Description
`JMSReplyUseMessageIdForCorrelation`	Used to specify whether message Id for correlation. Valid values are `true` and `false`. The default value if none is specified is false.
`JMSReplyPropagateJMSExpiration`	The boolean property specifies if the reply message TTL is set to 0 (message never expires) or some specified value related to message expiration. The default value is false.
`requestReply.cacheReceivers`	If the same small number of JMS receivers are used for the same request destination repeatedly, then set this property to `true` to improve performance. The default value if none is specified is false.
`requestReply.useCorrelation`	Applicable for a synchronous request-reply scenario. If set to `true`, then it applies a JMS Message selector for obtaining the reply message. If the request Normalized Message property, `jca.jms.JMSCorrelationID` is specified, then it is used for correlation, otherwise the JMS Message ID property is used. The JMS adapter uses the following message selector: `"JMSCorrelationID = '<corrId>' [ AND (<wsdlSelector>) ]"` where the AND branch is only included if the user specifies a `MessageSelector` property. The default value is `true`.
`suppressHeaders`	Used to bypass headers. For scenarios in which a composite does not use or produce the headers, the value of `true` can be used. It may improve performance for such scenarios. The default value is `false`.

A.5 Oracle Database Adapter Properties

This section describes the properties applicable to the Oracle Database Adapter, including:

- Table A–14, " JCA Properties for Oracle Database Adapter: Instance Properties"
- Table A–15, " JCA Properties for Oracle Database Adapter: Normalized Message Properties"

For properties applicable to all Oracle JCA Adapters, see Section A.7, "Generic Oracle JCA Adapter Properties".

For more information, see:

- Section 33.1.2.2, "Oracle Database Adapter", in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.
- Appendix H, "Normalized Message Properties", in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

Table A–14 JCA Properties for Oracle Database Adapter: Instance Properties

Property	Description
`dataSourceName`	Either this property or xADataSource name is a mandatory property, or both. Refers to the JNDI name (`jdbc/...`) of the `tx-level="local"` data source connecting to. All operations using this pool are locally transacted, independent on the global transaction. If both `xADataSourceName` and `dataSourceName` are specified, then the latter is used for `READ` operations.
`logTopLinkAll`	The default value is `FALSE`. You must increase DB Adapter logging to include all underlying `TopLink` log messages at maximum granularity. This property provides maximum visibility, but adapter logging is tuned to show the most relevant `TopLink` SQL logging.
`platformClassName`	This is a mandatory property. This points to the type of database being connected to. The suggested values for this property are: ■ Oracle11Platform ■ Oracle10Platform ■ Oracle9Platform ■ Oracle8Platform ■ DB2Platform ■ InformixPlatform ■ SybasePlatform ■ SQLServerPlatform ■ MySQLPlatform ■ DatabasePlatform You also can give the full package and class name of a subclass of `oracle.toplink.platform.database.DatabasePlatform`. For DB2 on AS/400, Oracle recommends that you give the value of `oracle.tip.adapter.db.toplinkext.DB2AS400Platform`.
`usesBatchWriting`	The default value is `TRUE`. Multiple identical statements are executed as a single batch statement. You must only disable this property for certain JDBC drivers that have known issues.
`usesSkipLocking`	The default value is `TRUE`. Oracle Database polling statements using `SELECT FOR UPDATE` are automatically be upgraded to `SELECT FOR UPDATE SKIP LOCKED`, which provides better concurrent performance. Disable this property only for certain cases where skip locking is not compatible with another feature or your use case.
`usesNativeSequencing`	The default value is `TRUE`. If any SOA services are configured to automatically assign sequence numbers on `INSERT` operation, then a `TRUE` value indicates that the sequence values are coming from a database native sequence.
`xADataSourceName`	This is a mandatory property. It specifies the JNDI name (`jdbc/...`) of the `tx-level="global"` data source connecting to the database. All operations using this pool bind to the global transaction and commit or roll back as a unit.

Table A–15 JCA Properties for Oracle Database Adapter: Normalized Message Properties

Property	Description
`jca.db.CursorName`	Inbound/Outbound.
`jca.db.DataSourceName`	Outbound.
`jca.db.Password`	Outbound. You cannot assign values to the `jca.db.password` property on Oracle Containers for Java EE because its data source does not support setting password dynamically to the `getConnection` method. Consider using `jca.db.ProxyPassword` instead.

Table A–15 (Cont.) JCA Properties for Oracle Database Adapter: Normalized Message Properties

Property	Description
`jca.db.ProxyCertificate`[1]	Outbound. When set, specifies `OracleConnection.PROXYTYPE_CERTIFICATE` as the proxy type. The value is a `base64Binary` encoded `byte[]` array that contains the certificate. If set, do not set `jca.db.ProxyDistinguishedName`, `jca.db.ProxyUser`, and `jca.db.ProxyPassword`.
`jca.db.ProxyDistinguishedName`[1]	Outbound. When set, specifies `OracleConnection.PROXYTYPE_DISTINGUISHED_NAME` as the proxy type. The value should be the proxy distinguished name as a `java.lang.String`. If set, then set none of `jca.db.ProxyCertificate`, `jca.db.ProxyUserName` and `jca.db.ProxyPassword`.
`jca.db.ProxyIsThickDriver`[1]	Outbound. Valid values are `true` and `false`. Set to `true` if using the OCI driver, as there is some discrepancy in the JDBC-level API between the thick and thin drivers.
`jca.db.ProxyPassword`[1]	Outbound. When set, specifies `OracleConnection.PROXYTYPE_USER_PASSWORD` as the proxy type. The value should be the password for the proxy user as a `java.lang.String`. If set, you must also set `jca.db.ProxyUserName`. If set, then set neither `jca.db.ProxyCertificate` nor `jca.db.ProxyDistinguished` name.
`jca.db.ProxyRoles`[1]	Outbound. Set to define the roles associated with the proxy user. The value should be a `String[]` array where each `java.lang.String` corresponds to a role name. This property is applicable when using any of `OracleConnection.PROXYTYPE_USER_PASSWORD`, `OracleConnection.PROXYTYPE_CERTIFICATE`, and `OracleConnection.PROXYTYPE_DISTINGUISHED_NAME`.
`jca.db.ProxyUserName`[1]	Outbound. When set, specifies `OracleConnection.PROXYTYPE_USER_PASSWORD` as the proxy type. The value should be the user name of the proxy user as a `java.lang.String`. If set, you must also set `jca.db.ProxyPassword`. If set, then set neither `jca.db.ProxyCertificate` nor `jca.db.ProxyDistinguished` name.
`jca.db.UserName`	Outbound. You cannot assign values to the `jca.db.userName` property on Oracle Containers for Java EE because its data source does not support setting user name dynamically to the `getConnection` method. Consider using `jca.db.ProxyUserName` instead.
`jca.db.XADataSourceName`	Outbound.

[1] For more information, see Section 9.3.4, "Proxy Authentication Support".

A.6 Oracle MQ Series Adapter Properties

This section describes the properties applicable to the Oracle MQ Series Adapter, including:

- Table A–16, " JCA Properties for Oracle MQ Series Adapter"
- Table A–17, " JCA Properties for Oracle MQ Series Adapter: Normalized Properties"
- Table A–18, " Connection Properties for Oracle MQ Series Adapter"
- Table A–19, " Binding Properties for Oracle MQ Series Adapter"

For properties applicable to all Oracle JCA Adapters, see Section A.7, "Generic Oracle JCA Adapter Properties".

For more information, see Section 33.1.2.6, "Oracle MQ Adapter", in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

Table A–16 JCA Properties for Oracle MQ Series Adapter

Property	Description
QueueName	This property specifies the name of the MQ Queue for sending or retrieving messages.
MessageType	This property specifies the type of message: Normal, Request, Reply, or Report
MessageFormat	This property specifies the type of MQ message format, such as Default, and Request/Reply.
Priority	This property specifies the message priority. Its value ranges from 0 to 9. The default value is AS_Q_DEF, which uses the value defined in the destination queue.
Persistence	This property is used to set the message persistence. The message persists when this property is set to true. The defaults value is AS_Q_DEF, which uses the value defined in the destination queue.
OnDeliveryFailure	This property is used when message delivery fails. The default value is DeadLetterQueue, which sends messages to a dead letter queue. If the value is set to DISCARD, then messages are discarded.
PartialDeliveryForDL	This property is used for partial delivery to a distribution list. The default value is false.
SegmentIfRequired	This property is used when the size of the message is larger than the maximum limit set on the queue.
Expiry	This property specifies the time after which the message would be removed by the Queue Manager. The default value is NEVER.
ReplyToQueueName	This property specifies the name of the queue to which the reply or report must be sent.
ReportCOA	If this property is set, a confirmation on arrival report is sent to the replyto queue on arrival of a message in the destination queue. The default value is WITH_NO_DATA, and no data is sent in this case.
ReportCO	If this property is set, a confirmation on delivery report is sent to the replyto queue on arrival of a message in the destination queue. The default value is WITH_NO_DATA, and no data is sent in this case.
ReportException	If this property is set, an exception report is sent to the replyto queue when message delivery to the destination queue fails. The default value is WITH_NO_DATA, and no data is sent in this case.
ReportExpiry	If this property is set, an expiry report is sent to the replyto queue when a message sent to the destination queue expires. The default value is WITH_NO_DATA, and no data is sent in this case.
WaitInterval	This property specifies the waiting interval for dequeuing the message in outbound MQ queue.
MessageId	This property is used to generate a Message Id for a reply or a report message. By default a new Message Id is generated.
CorrelationId	This property is used to generate a correlation Id for a reply or a report message. By default the message Id of the request message is used as the correlation Id.
QueueOpenOptions	This property specifies the queue open options to use while accessing the queue.
SecondaryQueueManagerName	This property specifies the queue manager for the enqueue queue. Use this property only when the outbound enqueue queue resides outside the inbound queue manager.

Table A–16 (Cont.) JCA Properties for Oracle MQ Series Adapter

Property	Description
BackoutQueue	This property is used to specify a backout queue to which rejected messages from an inbound queue are to be sent.
BackoutQueueManagerName	This property is used to specify the queue manager for the backout queue. Use this property only when the Backout Queue resides outside the inbound queue manager.
MaximumBackoutCount	This property is used to specify the maximum backout retry count after which rejected message is sent to the backout queue.
BackoutInterval	This property is used to specify intervel between the backout retries. The default value is 5 seconds.
BackoutRetries	This property is used to specify the number of backout retries. The default value is 3.
FallbackReplyToQueueName	This property is used for sending the report to the Normal Message Queue.
FallbackReplyToQueueManagerName	This property is used when the Primary Queue Manager specified in JNDI connection cannot access the queue.
DistributionList	This property is used to specify the elements of the distribution list for enqueuing the message.

Table A–17 JCA Properties for Oracle MQ Series Adapter: Normalized Properties

Property	Description
jca.mqAccountingToken	Inbound/Outbound. Accounting token information of the message. A hexadecimal-encoded string.
jca.mq.ApplIdentityData	Inbound/Outbound. Provides additional information about the Identity of the message or its originator. Accepts any string.
jca.mq.MQMD.ApplOriginData	Inbound/Outbound. Provides additional information about the origin of this message. Accepts any string.
jca.mq.MQMD.BackoutCount	Inbound/Outbound.Count of the number of times the message has previously been returned by an MQQueue.get() call as part of a unit of work, and subsequently backed out. Accepts zero and positive integer values.
jca.mq.MQMD.CorrelId	Inbound/Outbound. Correlation identifier of the message to be retrieved/ to be put. Accepts a hexadecimal-encoded string.
jca.mq.MQMD.Encoding.Decimal	Inbound/Outbound. Representation used for numeric values in the application message data. Accepts NORMAL and REVERSED.
jca.mq.MQMD.Encoding.Float	Inbound/Outbound. representation used for numeric values in the application message data. Accepts NORMAL, REVERSED and S390

Table A–17 (Cont.) JCA Properties for Oracle MQ Series Adapter: Normalized Properties

Property	Description
jca.mq.MQMD.Expiry	Inbound/Outbound. A message's expiry time has elapsed, and it is eligible to be discarded by the queue manager. Accepts NEVER or a non-Inbound/Outbound.negative integer value
jca.mq.MQMD.Feedback	Inbound/Outbound. Used with a message of type MQC.MQMT_REPORT to indicate the nature of the report. Accepts any string.
jca.mq.MQMD.Feedback.ApplicationDefined	Inbound/Outbound. application defined feedback.Accepts any string.
jca.mq.MQMD.Format	Inbound/Outbound. Format name used by the sender of the message to indicate the nature of the data in the message to the receiver.Accepts following formats NONE, ADMIN, CHANNEL_COMPLETED, CICS, CMD1, CMD2, DEAD_LETTER_HDR, DIST_HDR, EVENT, IMS, IMS_VAR_STRING, MD_EXTN, PCF, REF_MSG_HDR, RF_HDR_1, RF_HDR_2, STRING, TRIGGER, WORK_INFO_HDR, XMIT_Q_HDR
jca.mq.MQMD.GroupId	Inbound/Outbound. Byte string that identifies the message group to which the physical message belongs. Accepts hexadecimal-encoded string.
jca.mq.MQMD.MsgFlags.IsMsgInGroup	Inbound/Outbound. Specifies if the message belongs to a group. Accepts true, false.
jca.mq.MQMD.MsgFlags.IsLastMsgInGroup	Inbound/Outbound. Specifies if the message is the last message of the group. Accepts true, false
jca.mq.MQMD.MsgFlags.IsSegment	Inbound/Outbound. Specifies if the message is a segment. Accepts true, false
jca.mq.MQMD.MsgFlags.IsLastSegment	Inbound/Outbound. Specifies if message is the last segment. Accepts true, false
jca.mq.MQMD.MsgId	Inbound/Outbound. Message identifier of the message to be retrieved/ to be put. Accepts hexadecimal encoded string
jca.mq.MQMD.MsgSeqNumber	Inbound/Outbound. Sequence number of a logical message within a group. Accepts non- Inbound/Outbound.negative integer value
jca.mq.MQMD.MsgType	Inbound/Outbound. Indicates the type of the message. Accepts any string.
jca.mq.MQMD.MsgType.ApplicationDefined	Inbound/Outbound. Application -defined message type. Accepts any string.
jca.mq.MQMD.Offset	Inbound/Outbound. The offset of data in a physical message from the start of a logical message. Accepts non-Inbound/Outbound.Negative integer value.

Table A–17 (Cont.) JCA Properties for Oracle MQ Series Adapter: Normalized Properties

Property	Description
jca.mq.MQMD.OriginalLength	Inbound/Outbound. Original length of a segmented message. Accepts non-Negative integer value Inbound/Outbound
jca.mq.MQMD.Persistence	Inbound/Outbound. message persistence. Accepts true, false, AS_Q_DEF
jca.mq.MQMD.Priority	Inbound/Outbound. Message priority. Accepts 0- Inbound/Outbound.9, AS_Q_ DEF
jca.mq.MQMD.PutApplName	Inbound/Outbound. Name of the application that Put the message. Accepts any string
jca.mq.MQMD.PutApplType	Inbound/Outbound. Type of application that Put the message. Accepts any string.
jca.mq.MQMD.PutApplType.UserDefined	Inbound/Outbound. User-defined Put application type. Accepts any string.
jca.mq.MQMD.PutDateTime	Inbound/Outbound. Time and date that the message was Put. Accepts year:month:date, year:month:date:hour:minute, year:month:date:hour:minute:second
jca.mq.MQMD.ReplyToQMgr	*I*nbound/Outbound. Name of the queue manager to which reply or report messages should be sent. Accepts any sting.
jca.mq.MQMD.ReplyToQ	Inbound/Outbound. Name of the queue to which reply or report messages should be sent. Accepts any string
jca.mq.MQMD.Report.Generate.CorrelId	Inbound/Outbound. scheme to generate the CorrelationId of reply or report message. Accepts PASS_CORREL_ID, COPY_MSG_ID
jca.mq.MQMD.Report.Generate.MsgId	Inbound/Outbound. Scheme to generate the MessageId of reply or report message. Accepts NEW_MSG_ID, PASS_MSG_ID
jca.mq.MQMD.Report.Generate.COA	Inbound/Outbound. Specifies the content of COA report. Accepts WITH_NO_DATA, WITH_PARTIAL_DATA, WITH_FULL_ DATA
jca.mq.MQMD.Report.Generate.COD	Inbound/Outbound. Specifies the content of COD report. Accepts WITH_NO_DATA, WITH_PARTIAL_DATA, WITH_FULL_ DATA
jca.mq.MQMD.Report.Generate.Exception	Inbound/Outbound. Specifies the content of the Exception report. Accepts WITH_ NO_DATA, WITH_PARTIAL_DATA, WITH_FULL_DATA
jca.mq.MQMD.Report.Generate.Expiry	Inbound/Outbound. Specifies the content of the Expiry report. Accepts WITH_NO_ DATA, WITH_PARTIAL_DATA, WITH_ FULL_DATA
jca.mq.MQMD.Report.Generate.NAN	Inbound/Outbound. Specifies if the incoming/outgoing message is NAN or not. Accepts true, false

Table A–17 (Cont.) JCA Properties for Oracle MQ Series Adapter: Normalized Properties

Property	Description
jca.mq.MQMD.Report.Generate.PAN	Inbound/Outbound. specify if incoming/outgoing message is PAN or not. Accepts true, false
jca.mq.MQMD.Report.TakeAction.OnMsgDeliveryFailure	Inbound/Outbound.
jca.mq.MQMD.Report.TakeAction.OnMsgDeliveryFailure	Accepts DISCARD, DEADLETTERQUEUE
jca.mq.MQMD.StrucId	Inbound/Outbound. Struct id of MQMD. Accepts any string
jca.mq.MQMD.UserIdentifier	Inbound/Outbound. User who originated this message. Accepts any string.
jca.mq.MQMD.Version	Inbound/Outbound. Version of MQMD. Accepts VERSION_1, VERSION_2
jca.mq.Inbound.MQMD.CorrelId	Outbound. Correlation identifier of the message retrieved in Async req-reply scenario. Accepts hexadecimal encoded string
jca.mq.Inbound.MQMD.MsgId	Outbound. Message identifier of the messageretrieved in Async req-reply scenario. Accepts hexadecimal-encoded string.
jca.mq.Inbound.MQMD.MsgType	Outbound. Message Type of the message retrieved in Async req-reply scenario. Accepts any string.
jca.mq.Inbound.MQMD.Nan	Outbound. NAN report option of the message retrieved in the Async req-reply scenario. Aceepts true, false
jca.mq.Inbound.MQMD.Pan	Outbound. PAN report option of the message retrieved in the Async req-reply scenario. Accepts true, false
jca.mq.Inbound.MQMD.ReplyToQMgr	Outbound. ReplyToQueueManager of the message retrieved in the Async req-reply scenario. Accepts any string.
jca.mq.Inbound.MQMD.ReplyToQ	Outbound. ReplyToQueue of the message retrieved in the Async req-reply scenario. Accepts any string.
jca.mq.Inbound.MQMD.Report.Generate.CorrelId	Outbound. Correlation scheme, for generation of CorrelationId, of the message retrieved in Async req-reply scenario. Accepts PASS_CORREL_ID, COPY_MSG_ ID
jca.mq.Inbound.MQMD.Report.Generate.MsgId	Outbound. Correlation scheme, for generation of MessageId, of the message retrieved in Async req-reply scenario. Accepts NEW_MSG_ID, PASS_MSG_ID
jca.mq.ISpec.EnqueueMsgToQMgr	Outbound. Queue Manager for outbound queue. Accepts any string
jca.mq.ISpec.EnqueueMsgToQ	Outbound. Queue name of outbound queue. Accepts any string.

Table A–18 Connection Properties for Oracle MQ Series Adapter

Property	Description
hostName	Name of the host computer.
portNumber	Port number to be used.
channelName	Set it to the server connection channel to be used.
queueManagerName	A valid queue manager name.
CipherSuite	Set CipherSuite to the name matching the CipherSpec set on the SVRCONN channel. If set to null (default), then no SSL encryption is performed.
clientEncoding	Character encoding used by the client.
connectionFactoryLocation	Location of the connection factory.
hostOSType	Operating system used by the host computer.
KeyStoreAlgorithm	Algorithm used by the key store.
KeyStoreLocation	This value is the keystore where Oracle MQ Series Adapter has its private keys. This is required when an adapter must authenticate itself to the MQ Series server.
KeyStorePassword	This value is the password that is required to access keystore.
KeyStoreProviderName	The name of the keystore provider.
TrustStoreLocation	This is the location where the adapter keeps its trusted certificates information. This information is required when an adapter must authenticate to the MQ Series server.
TrustStorePassword	This property specifies the password of the Trust Store location.
KeyStoreType	This property specifies the type of the key store.
Protocol	Key Management Algorithm.
SSLPeerName	A distinguished name pattern. If CipherSuite is set, you can use this variable to ensure that the correct queue manager is used. If set to null (default), then the DN of the queue manager is not checked. This variable is ignored if sslCipherSuite is null.
SSLEnable	The true or false value for this property means that the Oracle MQ Series Adapter is SSL enabled or SSL disabled.
userID	This property is used if credential mapping is not set.
password	This is the password to connect to the queue manager. This property is used if credential mapping is not set.
XATransaction	This property is used to enable or disable XA transactions. If set to true, then XA transaction is enabled.
receiveExit	This is the Receive Exit java class, which gets triggered when you receive a message on a particular queue.
securityExit	This is the Security Exit java class, which enables you to customize the security flows that occur when an attempt is made to connect to a queue manager.
sendExit	This is the Send Exit java class, which gets triggered when you send a message to a particular queue.

Table A–19 Binding Properties for Oracle MQ Series Adapter

Property	Description
`adapter.mq.inbound.queueName`	This property is used to specify the name of the inbound MQ queue.
`adapter.mq.inbound.binaryNulls`	This property is used for dequeuing the messages with binary zero value. The default value for this property is `true`.

A.7 Generic Oracle JCA Adapter Properties

This section describes the properties applicable to all Oracle JCA Adapters, including:

- Table A–20, " JCA Properties for all Oracle JCA Adapters"

For properties specific to each of the Oracle JCA Adapters, see:

- Section A.1, "Oracle File and FTP Adapters Properties"
- Section A.2, "Oracle Socket Adapter Properties"
- Section A.3, "Oracle AQ Adapter Properties"
- Section A.4, "Oracle JMS Adapter Properties"
- Section A.5, "Oracle Database Adapter Properties"
- Section A.6, "Oracle MQ Series Adapter Properties"

For more information, see Section 33.1.2.8, "Oracle JCA Adapters Endpoint Properties", in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

Table A–20 JCA Properties for all Oracle JCA Adapters

Property	Description

`wsdlLocation`	An optional `adapter-config` attribute of type `xs:string`.
	When set to the name of the WSDL associated with the adapter's JCA file, the Oracle Service Bus can automatically resolve the WSDL to allow bulk import of JCA files and related WSDL and schemas.
	The example below shows the `FulfillmentBatch_jms.jca` file with the `adapter-config` attribute `wsdlLocation` set to the name of the corresponding WSDL file `FulFillmentBatch.wsdl`.
	`<adapter-config name="FulfillmentBatch" adapter="Jms Adapter"`
	`xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata"`
	`wsdlLocation="FulfillmentBatch.wsdl" >`
	`...`
	`</adapter-config>`

B

Oracle JCA Adapter Valves

This appendix includes sample valves used by Oracle File and FTP Adapters. A valve is the primary component of execution in an FTP or File Adapter processing pipeline. A valve processes the content it receives and forwards the processed content to the next valve.

This chapter contains the following sections:

- Section B.1, "A Simple Unzip Valve"
- Section B.2, "A Simple Decryption Valve That Uses Staging File"
- Section B.3, "A Valve for Encrypting Outbound Files"
- Section B.4, "An Unzip Valve for processing Multiple Files"

B.1 A Simple Unzip Valve

The following sample is a simple Unzip Valve:

```
package valves;

import java.io.*;
import java.util.zip.*;

import oracle.tip.pc.services.pipeline.AbstractValve;
import oracle.tip.pc.services.pipeline.InputStreamContext;
import oracle.tip.pc.services.pipeline.PipelineException;

/**
 * A simple valve to process zip files.
 * The valve processes the first entry from the zip file.
 * If you need to process multiple files, you will need
 * a re-entrant valve
 **/
public class SimpleUnzipValve extends AbstractValve {

      public InputStreamContext execute(InputStreamContext inputStreamContext)
                  throws IOException, PipelineException {
            // Get the input stream that is passed to the Valve
            InputStream originalInputStream = inputStreamContext.getInputStream();

            // Create a new ZIP input stream
            ZipInputStream zipStream = null;
            try {
                  zipStream = new ZipInputStream(originalInputStream);
                  ZipEntry entry = null;
                  // In this sample valve, lets pick up the first entry
```

```
                  if ((entry = zipStream.getNextEntry()) != null) {
                        System.out.println("Unzipping " + entry.getName());
                        ByteArrayOutputStream bos = new ByteArrayOutputStream();
                        byte[] buf = new byte[4096];
                        int len = 0;
                        while ((len = zipStream.read(buf)) > 0) {
                              bos.write(buf, 0, len);
                        }
                        bos.close(); // no-op but still ...
                        ByteArrayInputStream bin = new ByteArrayInputStream(bos
                                    .toByteArray());
                        // This is where the Valve returns the inputstream to the
caller
                        // Example, Adapter
                        // return the newly created inputstream as a part of the
context
                        inputStreamContext.setInputStream(bin);

                        return inputStreamContext;
                  }
            } finally {
                  if (zipStream != null) {
                        zipStream.close();
                  }
            }
            // return null if no data
            return null;

      }

      @Override
      // Not required for this simple valve
      public void finalize(InputStreamContext in) {

      }

      @Override
      // Not required for this simple valve
      public void cleanup() throws PipelineException, IOException {

      }


}
```

B.2 A Simple Decryption Valve That Uses Staging File

The following is a sample decryption valve that uses a staging file:

```
package valves;

import java.io.*;
import javax.crypto.*;
import javax.crypto.spec.*;

import oracle.tip.pc.services.pipeline.AbstractStagedValve;
import oracle.tip.pc.services.pipeline.InputStreamContext;
import oracle.tip.pc.services.pipeline.PipelineException;
import oracle.tip.pc.services.pipeline.PipelineUtils;
```

```
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

/**
 * Simple Decryption valve that uses DES algorightm
 *
 * You must note that this class uses AbstractStagedValve. By using the
 * AbstractStagedValve, the valve notifies the pipeline that the valve will take
 * care of its own staging and cleanup
 *
 */
public class SimpleDecryptValve extends AbstractStagedValve {

      // Staging file where the intermediate decrypted content is kept
      private File stagingFile = null;

      /**
       * Called by the adapter. All the binding/reference properties in the
       * composite are available to the pipeline via the pipeline context For
       * example <service name="FlatStructureIn"> <interface.wsdl
       *
interface="http://xmlns.oracle.com/pcbpel/adapter/file/FlatStructureIn/#wsdl.inter
face(Read_ptt)"/>
       * <binding.jca config="FlatStructureIn_file.jca"> <property
       * name="myCipherKey" source="" type="xs:string" many="false"
       * override="may">somekey</property> </binding.jca> </service>
       *
       */
      public InputStreamContext execute(InputStreamContext inputStreamContext)
                  throws IOException, PipelineException {

            // Read the cipher key from the adapter binding property 'myCipherKey'
            String cipherKey = (String) getPipeline().getPipelineContext()
                        .getProperty("myCipherKey");

            // If key is blank, default to some hard-coded value
            if (PipelineUtils.isBlank(cipherKey)) {
                  System.out.println("using default ciper key");
                  cipherKey = "desvalve";
            }
            // Create an instance of the Cipher
            byte key[] = cipherKey.getBytes();
            SecretKeySpec secretKey = new SecretKeySpec(key, "DES");
            Cipher decrypt = null;
            try {
                  decrypt = Cipher.getInstance("DES/ECB/PKCS5Padding");
            } catch (NoSuchPaddingException nspe) {
                  throw new PipelineException("Unable to get cipher instance",
nspe);
            } catch (NoSuchAlgorithmException nsae) {
                  throw new PipelineException("Invalid cipher algorithm", nsae);
            }
            try {
                  decrypt.init(Cipher.DECRYPT_MODE, secretKey);
            } catch (InvalidKeyException ike) {
                  throw new PipelineException("Invalid secret key", ike);
            }
            // original input stream from caller. For example, adapter
            InputStream originalInputStream = null;
            CipherInputStream cis = null;
```

```
            try {
                  originalInputStream = inputStreamContext.getInputStream();
                  cis = new CipherInputStream(originalInputStream, decrypt);
            } catch (Exception e) {
                  throw new PipelineException("Unable to create cipher stream",
e);
            }
            // Since we're using a staged valve, we will store the decrypted
content
            // in a staging file
            // In this case, we're leveraging the File/Ftp Adapter control
directory
            // to store the content, but, the staging file can be placed anywhere
            this.stagingFile = PipelineUtils.getUniqueStagingFile(getPipeline()
                        .getPipelineContext().getStagingDirectory());

            // Write the decrypted content to the staging file
            OutputStream os = new FileOutputStream(this.stagingFile);
            byte[] b = new byte[8];
            int i = cis.read(b);
            while (i != -1) {
                  os.write(b, 0, i);
                  i = cis.read(b);
            }
            os.flush();
            os.close();
            cis.close();

            // Open a stream to the staging file and return it back to the caller
            InputStream in = new FileInputStream(this.stagingFile);
            // close the input stream passed in this invocation
            inputStreamContext.closeStream();
            // set the input stream to staging file and return
            inputStreamContext.setInputStream(in);
            return inputStreamContext;

      }

      /*
       * (non-Javadoc)
       *
       * @see oracle.tip.pc.services.pipeline.AbstractStagedValve#getStagingFile()
       */
      public File getStagingFile() {
            return stagingFile;
      }

      /*
       * Delete the staging file if there is one (non-Javadoc)
       *
       * @see
oracle.tip.pc.services.pipeline.AbstractValve#finalize(oracle.tip.pc.services.pipe
line.InputStreamContext)
       */
      public void finalize(InputStreamContext ctx) {
            try {
                  cleanup();
            } catch (Exception e) {
            }
      }
```

```
      /*
       * Use this method to delete the staging file (non-Javadoc)
       *
       * @see oracle.tip.pc.services.pipeline.AbstractStagedValve#cleanup()
       */
      public void cleanup() throws PipelineException, IOException {

            if (stagingFile != null && this.stagingFile.exists()) {
                  this.stagingFile.delete();
            }
            this.stagingFile = null;
      }

}
```

B.3 A Valve for Encrypting Outbound Files

The following is a simple encryption valve that extends `AbstractValve`.

```
package valves;

import java.io.*;
import javax.crypto.*;
import javax.crypto.spec.*;

import oracle.tip.pc.services.pipeline.AbstractValve;
import oracle.tip.pc.services.pipeline.InputStreamContext;
import oracle.tip.pc.services.pipeline.PipelineException;
import oracle.tip.pc.services.pipeline.PipelineUtils;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

/**
 * Simple Encryption valve that uses DES algorightm
 *
 */
public class SimpleEncryptValve extends AbstractValve {


/**
      * Called by the adapter. All the binding/reference properties
      * in the composite are available to the pipeline via
      * the pipeline context
      * For example
      * <service name="FlatStructureOut">
      *     <interface.wsdl
interface="http://xmlns.oracle.com/pcbpel/adapter/file/FlatStructureOut/#wsdl.inte
rface(Write_ptt)"/>
      *     <binding.jca config="FlatStructureOut_file.jca">
      *         <property name="myCipherKey" source="" type="xs:string"
many="false" override="may">somekey</property>
      *   </binding.jca>
      * </service>
      *
      */
     public InputStreamContext execute(InputStreamContext inputStreamContext)
               throws IOException, PipelineException {
```

```
          //Read the cipher key from the adapter binding property 'myCipherKey'
          String cipherKey = (String) getPipeline().getPipelineContext()
                    .getProperty("myCipherKey");

          //If key is blank, default to some hard-coded value
          if (PipelineUtils.isBlank(cipherKey)) {
               System.out.println("using default ciper key");
               cipherKey = "desvalve";
          }
          //Create an instance of the Cipher

pt.init(Cipher.ENCRYPT_MODE, secretKey);
          } catch (InvalidKeyException ike) {
               throw new PipelineException("Invalid secret key", ike);
          }
          //original input stream from caller. For example, adapter
          ByteArrayOutputStream bos = new ByteArrayOutputStream();
          try {

               encryptStream(inputStreamContext.getInputStream(), bos, encrypt);
          } catch (Exception e) {
               throw new PipelineException("Unable to encrypt", e);
          }
          byte[] bytes = bos.toByteArray();
          InputStream in = new ByteArrayInputStream(bytes);
          //close the input stream passed in this invocation
          inputStreamContext.closeStream();
          //set the input stream and return
          inputStreamContext.setInputStream(in);
          return inputStreamContext;

     }

     private static void encryptStream(InputStream in, OutputStream out, Cipher
encrypt) {
          try {
               byte[] buf = new byte[4096];
               // Bytes written to out will be encrypted
               out = new CipherOutputStream(out, encrypt);

               // Read in the cleartext bytes and write to out to encrypt
               int numRead = 0;
               while ((numRead = in.read(buf)) >= 0) {
                    out.write(buf, 0, numRead);
               }
               out.close();
          } catch (java.io.IOException e) {
          }
     }

     /*
      * Delete the staging file if there is one
      * (non-Javadoc)
      * @see
oracle.tip.pc.services.pipeline.AbstractValve#finalize(oracle.tip.pc.services.pipe
line.InputStreamContext)
      */
     public void finalize(InputStreamContext ctx) {
          try {
               cleanup();
```

```
            } catch (Exception e) {
            }
        }

        /*Use this method to delete the staging file
         * (non-Javadoc)
         * @see oracle.tip.pc.services.pipeline.AbstractStagedValve#cleanup()
         */
        public void cleanup() throws PipelineException, IOException {

        }

        public static void main(String[] args) throws Exception{
            String cipherKey = "desvalve";

            //Create an instance of the Cipher
            byte key[] = cipherKey.getBytes();
            SecretKeySpec secretKey = new SecretKeySpec(key, "DES");
            Cipher encrypt = null;
            try {
                encrypt = Cipher.getInstance("DES/ECB/PKCS5Padding");
            } catch (NoSuchPaddingException nspe) {
                throw new PipelineException("Unable to get cipher instance", nspe);
            } catch (NoSuchAlgorithmException nsae) {
                throw new PipelineException("Invalid cipher algorithm", nsae);
            }
            try {
                encrypt.init(Cipher.ENCRYPT_MODE, secretKey);
            } catch (InvalidKeyException ike) {
                throw new PipelineException("Invalid secret key", ike);
            }
            //original input stream from caller. for example, adapter

            FileInputStream fin = new FileInputStream(args[0]);
            FileOutputStream fout = new FileOutputStream(args[1]);
            try {

                encryptStream(fin, fout, encrypt);
            } catch (Exception e) {
                throw new PipelineException("Unable to encrypt", e);
            }
            fin.close();
            fout.close();

        }

}
```

B.4 An Unzip Valve for processing Multiple Files

The following is the sample of an unzip valve for processing multiple files:

```
package valves;

import java.io.*;
import java.util.zip.*;
import java.util.*;

import oracle.tip.pc.services.pipeline.AbstractStagedValve;
```

```
import oracle.tip.pc.services.pipeline.InputStreamContext;
import oracle.tip.pc.services.pipeline.PipelineException;
import oracle.tip.pc.services.pipeline.PipelineUtils;

/**
 * A re-entrant valve is one that can be invoked multiple times
 * and on each invocation it must return a new stream.
 * This concept is used here in this sample to process
 * a zipped file containing multiple entries.
 *
 * If a valve is marked as re-entrant, then the caller (adapter),
 * calls hasNext() on the valve to check if there are more
 * streams available
 */
public class ReentrantUnzipValve extends AbstractStagedValve {

      //member variables
      private boolean initialized = false;

      private List<String> files = null;

      private File currentFile = null;

      private File unzipFolder = null;

      /**
       * On the first invocation, this valve unzips the zip file into
       * a staging area and returns a stream the first unzipped file
       * On subsequent iterations, the valve returns streams to
       * subsequent files.
       */
      public InputStreamContext execute(InputStreamContext inputStreamContext)
                  throws IOException, PipelineException {
            String fileName = "";
            //the first time that the valve is invoked, unzip the file into
            //the staging area
            if (!initialized) {
                  files = new ArrayList<String>();
                  //Get hold of the File/Ftp adapter control directory
                  File controlDirectory = getPipeline().getPipelineContext()
                              .getStagingDirectory();
                  //Create if required
                  if (!controlDirectory.exists()) {
                        controlDirectory.mkdirs();
                  }
                  //Generate a unique folder to store the staging files
                  String digestPath = "";
                  try {
                        digestPath = PipelineUtils.genDigest(inputStreamContext
                                    .getMessageOriginReference());
                  } catch (Exception e) {
                        digestPath = String.valueOf(inputStreamContext
                                    .getMessageOriginReference().hashCode());
                  }
                  unzipFolder = new File(controlDirectory, digestPath);
                  if (!unzipFolder.exists())
                        unzipFolder.mkdirs();
                  //unzip the files into the staging folder
                  unzipToDirectory(inputStreamContext.getInputStream(),
unzipFolder);
```

```
                  //store the file names into the list
                  PipelineUtils.listFiles(unzipFolder, files);
                  //close the input stream
                  inputStreamContext.closeStream();
            }
            initialized = true;
            //return the next one in the list
            if (files != null && files.size() > 0) {
                  fileName = files.remove(0);
                  currentFile = new File(fileName);
                  System.out.println("Returning file[" + fileName + "]");
                  //Open a stream to the file and return to caller. For example,
adapter
                  FileInputStream fis = new FileInputStream(currentFile);
                  inputStreamContext.setInputStream(fis);
                  /*For re-entrant valves, setting the message key is
                   important since this allows the caller to distinguish
                   between parts for the same message. for example, in the
                   case of zip file in this example, the
                   messageOriginReference will be same, but, the individual
                   message keys will vary. For example, the messageOriginReference
                   will be "/input/in.zip", whereas message key might be something
                   like "dir1/address-csv1.txt", "dir1/address-csv2.txt" and so on
                   */
                  inputStreamContext.setMessageKey(fileName);
                  return inputStreamContext;
            } else {
                  //return null if no more files
                  return null;
            }
      }

      /*
       * Adapter calls this to check if there are more files
       * @see oracle.tip.pc.services.pipeline.AbstractValve#hasNext()
       */
      public boolean hasNext() {
            return (files != null && files.size() > 0);
      }

      /*
       * Returns the current file being processed
       * @see oracle.tip.pc.services.pipeline.AbstractStagedValve#getStagingFile()
       */
      public File getStagingFile() {
            return currentFile;
      }

      /*
       * delete the current file once the entry has been published to binding
component
       * @see
oracle.tip.pc.services.pipeline.AbstractValve#finalize(oracle.tip.pc.services.pipe
line.InputStreamContext)
       */
      public void finalize(InputStreamContext ctx) {

            if (currentFile != null && currentFile.exists()) {
                  currentFile.delete();
            }
```

```
        }

        /*
         * Cleanup intermediate files
         * @see oracle.tip.pc.services.pipeline.AbstractStagedValve#cleanup()
         */
        public void cleanup() throws PipelineException, IOException {
              PipelineUtils.deleteDirectory(unzipFolder);
              initialized = false;
              if (currentFile != null && currentFile.exists()) {
                    currentFile.delete();
              }
              files = null;
        }

        /*
         * Unzip to the directory
         */
        private void unzipToDirectory(InputStream in, File directory)
                    throws IOException {
              ZipInputStream zin = new ZipInputStream(in);
              ZipEntry entry = null;
              if ((entry = zin.getNextEntry()) != null) {
                    do {
                          String entryName = entry.getName();
                          if (!entry.isDirectory()) {
                                File file = new File(directory, entryName);
                                unzipFile(zin, file);
                          }
                    } while ((entry = zin.getNextEntry()) != null);
              }
              zin.close();
        }

        private void unzipFile(InputStream in, File file) throws IOException {
              if (!file.getParentFile().exists()) {
                    file.getParentFile().mkdirs();
              }
              OutputStream os = new FileOutputStream(file);
              byte[] buf = new byte[4096];
              int len = 0;
              while ((len = in.read(buf)) > 0) {
                    os.write(buf, 0, len);
              }
              os.close();
        }

  }
```

Index

A

ability
  SequencingPollingStrategy (Last Read or Last Updated, 9-45
Accessing Distributed Destinations (Queues and Topics) on the WebLogic Server JMS, 8-56
  compatibility and migration, 8-59
  JMS Adapter with Distributed Queues and Distributed Topics, 8-57
  one copy cf a message per adapter endpoint, 8-58
  one copy of a message per application, 8-57
  providing JMS adapter access to distributed destinations, 8-56
Accessing Distributed Destinations (Queues and Topics) on the WebLogic Server JMS specifying the message selector when defining an activation spec, 8-58
Accessing Queues and Topics Created in 11g from the OC4J 10.1.3.4 Server, 8-53
activation agent
  singleton, 2-20
ActivationSpec parameters, 4-49
adapter
  adapter support of global transactions, 2-16
  adapter support of local transactions, 2-16
Adapter Configuration Wizard
  configuring the database adapter, 9-133
  starting, 4-2, 5-2, 9-9
  stored procedures, 9-79
  using with the AQ adapter, 7-12
  using with the file adapter, 4-3, 4-4
  using with the FTP adapter, 4-3, 4-4
  using with the MQ Series adapter, 10-6
  using with the socket adapter, 5-4
Adapter Configuration wizard
  using with the file adapter, 10-7
  using with the MQ Series adapter, 10-7
adapter services
  event notification, 1-18
  metadata interaction, 1-19
  request-response, 1-18
adapters
  adding a connection factory, 2-25
  asynchronous example, 2-18
  asynchronous transaction flow, 2-17
batching and debatching support, 2-25
checking for rejected messages from database, 2-33
checking for rejected messages from Fusion Middleware Control Console, 2-33
composite availability and inbound adapters, 2-20
CorrelationID of Receive message not matching Invoke, 2-22
creating an application server connection, 2-7
creating fault policies, 2-31
defining adapter interface by importing an existing WSDL, 2-2
deploying an Application Profile, 2-10
deploying JDeveloper, 2-10
error handling, 2-31
features, 1-1
global transaction retries and rollbacks and fault policies, 2-16
handling deployment plan, 2-14
hot-standby state, 2-21
how adapters ensure no memory loss, 2-15
inbound transactions, 2-16, 2-19
installation, 2-2
integration with Oracle SOA Composite, 3-14
local transaction and global (XA) transactions, 2-15
manually deploying a RAR file, 2-13
message error rejection handlers, 2-34
migrating repositories, 2-14
monitoring, 3-16
multiple activations of the same adapter endpoint, 2-21
outbound transactions, 2-19
Payload Native Size, 2-23
physical deployment, 2-6
properties
  generic, A-22
  Oracle AQ Adapter, A-8
  Oracle DB Adapter, A-13
  Oracle File Adapter, A-1
  Oracle FTP Adapter, A-1
  Oracle JMS Adapter, A-10
  Oracle MQ Series Adapter, A-15
  Oracle Socket Adapter, A-7
  wsdlLocation, A-22

recommended setting for data sources, 2-29
rejecting nonmatching Native Correlation IDs, 2-22
setting the trace levels, 2-43
singleton (active/passive) inbound endpoint lifecycle support, 2-20
specifying Inbound retry properties in the composite.xml file, 2-37
starting and stopping, 2-2
streaming large payload, 2-24
synchronous transaction flow, 2-18
testing applications, 2-42
types, 1-2
types of adapter services, 1-17
using header properties, 2-5
viewing logs, 2-44
XA transaction support, 2-15
adding a connection factory
creating a connection pool, 2-26
creating a data source, 2-25
adding a third-party JMS provider for non-Web Logic server JMS and non-AQJMS connection instances, 8-24
adding or updating an adapter connection factory, 2-28
ADT payload types
AQ adapter, 7-6
after-read strategy
database adapter, 9-24
APIs, 9-90
AQ adapter
ADT payload types, 7-6
AQ header properties, 7-4
correlation identifier, 7-3
dequeue mode, 7-4
dequeuetimeout property, 7-9
DOM 2 compliance, 7-8
enqueue-specific features, 7-3
error handling support, 7-10
features, 7-2
inbound retries, 7-9
integration with Oracle BPEL Process Manager, 7-1
integration with Oracle Mediator, 7-2
introduction, 7-1
message priority, 7-3
message-size aware, 7-8
multiconsumer queue, 7-3
multiple inbound polling threads, 7-9
multiple receiver threads, 7-8
normalized message support, 7-7
payload schema, 7-7
performance tuning, 7-10
stream payload support, 7-9
use cases, 7-12, 7-48, 7-65
AQ header properties
AQ adapter, 7-4
arrayIdentifierLength
construct, 6-10
arrayLength
construct, 6-10
arrays
native schema, 6-20
arrayTerminatedBy
construct, 6-10
assembling or wire inbound adapter, BPEL and Outbound adapter reference, 4-174
assign
construct, 6-10

B

batch processing
support with the file and FTP adapters, 4-40, 4-68
batching
definition, 4-45
BPEL Process Manager
integration with inbound interaction, 3-8
integration with outbound interaction, 3-7
overview, 3-4
byteOrder
construct, 6-9

C

cellSeparatedBy
construct, 6-10
choiceCondition
construct, 6-10
chunkedRead
definition, 4-7
supported with file and FTP adapters, 4-7
clauses
clauses to add to impact the sign position in COBOL Copybook, 6-7
supported COBOL Copybook clauses, 6-5
COBOL Copybook
clauses to add to impact the sign position, 6-7
definition, 6-4
multiple root levels, 6-121
Native Format Builder wizard support, 6-4
numeric types, 6-133
single root level, virtual decimal, fixed length array, 6-129
supported clauses, 6-5
use cases, 6-121
user inputs, 6-5
variable length array, 6-132
com.ibm.mq.jar, 10-60
complex structure
native schema, 6-83
conditional processing
native schema, 6-26
conditionValue
construct, 6-10
configuration properties
for socket adapter, 5-18
configuring
socket adapter, 5-17
configuring a custom adapter, 2-45

configuring oracle JMS adapter use case, 8-8
creating an application and a SOA project, 8-8
constructs
arrayIdentifierLength, 6-10
arrayLength, 6-10
arrayTerminatedBy, 6-10
assign, 6-10
byteOrder, 6-9
cellSeparatedBy, 6-10
choiceCondition, 6-10
conditionValue, 6-10
dataLines, 6-10
dateFormat, 6-11
encodeLineTerminators, 6-10
encoding, 6-9
fieldValidation, 6-10
headerLines, 6-9
headerLinesTerminatedBy, 6-9
identifierLength, 6-11
itemSeparatedBy, 6-11
leftsurroundedBy, 6-11
length, 6-11
listTerminatedBy, 6-11
lookAhead, 6-11
native schema, 6-9
outboundHeader, 6-10
paddedBy, 6-11
padStyle, 6-11
parseBom, 6-10
quotedBy, 6-11
rightsurroundedBy, 6-11
skip, 6-11
skipLines, 6-11
skipUntil, 6-11
standalone, 6-9
startsWith, 6-11
stream, 6-9
style, 6-11
surroundedBy, 6-11
terminatedBy, 6-11
uniqueMessageSeparator, 6-9
useArrayIdentifiers, 6-10
validateNxsd, 6-10
validation, 6-10
variable, 6-11
variables, 6-11
version, 6-9
xmlversion, 6-10
control table
database adapter, 9-67
correlation identifier
AQ adapter, 7-3
Correlation Schemas, 10-49
Correlation Support Within Adapters, 2-21
creating a custom adapter, 2-45
custom adapter screen flow, 2-46
creating database connections, 9-100
creating relationships
database adapter, 9-17
CSV
native schema, 6-63
cursor variable, 9-38
Custom Java handler, 2-35

D

database adapter
advanced use cases for outbound invoke operations, 9-62
after-read strategy, 9-24
concepts, 9-49
control table, 9-67
creating relationships, 9-17
creating the attribute filter, 9-21
datatype conversions, 9-114, 9-115
defining keys, 9-16
defining the WHERE clause, 9-22
deleting the rows that were read, 9-25
deployment, 9-70
design overview, 9-3
entering the SQL string for the Pure SQL operation, 9-33
features, 9-1, 9-34
performance tuning, 9-46
proxy authentication support, 9-40
Pure SQL - XML type support, 9-37
row set support (strongly typed XSD), 9-38
row set support (weakly typed XSD), 9-38
scalability, 9-41
schema validation, 9-41
streaming large payload, 9-40
transaction support, 9-34
function return values, 9-116
importing and selecting tables, 9-15
integration with Oracle BPEL Process Manager, 9-2
introduction, 9-1
last updated, 9-67
logical delete, 9-64
mapping any relational schema to any XML schema types, 9-55
merge operations, 9-60
null values, 9-116
operation type, 9-12
physical delete, 9-62
polling strategies, 9-62
processing ResultSets, 9-116
query by example operations, 9-60
querying over multiple tables, 9-55
relational types to XML schema types, 9-54
relational-to-XML mappings, 9-50
returning an INTEGER status value, 9-117
row set support
strongly typed XSD, 9-118
weakly typed XSD, 9-127
sequencing table, 9-27, 9-67
specifying advanced options, 9-31
specifying polling options, 9-29
SQL operations as Web services, 9-60
stored procedures and functions, 9-79, 9-112

support for PL/SQL Boolean, PL/SQL record, and PL/SQL table types, 9-129
support third-party databases, 9-92
update a field in the table, 9-25
use cases for outbound invoke operations, 9-61
use cases for Pure SQL, 9-61
value binding, 9-112
walkthrough of the Adapter Configuration Wizard, 9-5
database connection
configuring in the weblogic-ra.xml file, 9-10
database operations
DML operations, 9-60
dataLines
construct, 6-10
datatype conversions
database adapter, 9-114, 9-115
dateFormat
construct, 6-11
dates
native schema, 6-34
debatching
definition, 4-7, 4-45
supported with file and FTP adapters, 4-7
defining keys
database adapter, 9-16
deleting rows that were read
database adapter, 9-25
delimited format
Native Format Builder wizard support, 6-3
deployment
of database adapter, 9-70
packaged-application dapters, 1-17
technology adapters, 1-7
dequeue mode
AQ adapter, 7-4
design time
technology adapters, 1-5
detectOmissions Feature, 9-46
disributed destination
FactoryProperty, 8-56
distirbiuted destination
one copy of message per adapter endpoint, 8-58
distribiuted destination
SubscriptionSharingPolicy, 8-56
distributed, 8-56
distributed destination, 8-56
ClientIDPolicy property, 8-56
durable subscription, 8-57
enabling Client ID sharing, 8-56
endpoint deactivation, 8-57
FactoryProperty
EXCLUSIVE, 8-56
SHARABLE, 8-56
inbound queues, 8-57
JMS messages, 8-57
message selector defined in an activation spec example, 8-59
one copy of a message per adapter endpoint
TopicMessageDistributionAll, 8-58
one copy of a message per application, 8-57
NO JMS_WL_DDForwarded, 8-58
outbound errors, 8-57
specifying one copy of a message per application or per endpoint, 8-56
distributed destinations
sharing Durable subscriptions, 8-56
distributed polling
SKIP LOCKED, 9-41
tuning on single node first, 9-45
Distribution List Support, 10-49
DML operations
merge, 9-60
query by example, 9-60
with the database adapter, 9-60
DTD format
Native Format Builder wizard support, 6-4
dynamic outbound directory and file name specification, 4-8

E

encodeLineTerminators
construct, 6-10
encoding
construct, 6-9
definition, 5-12
supported with socket adapter, 5-12
enqueue-specific features
AQ adapter, 7-3
error handling
message errors, 2-33
rejected messages, 2-31

F

fieldValidation
construct, 6-10
File adapter
creating a synchronous BPEL composite using the File Adapter, 4-241
file adapter
ActivationSpec parameters, 4-49
architecture, 4-1
archiving successfully processed files, 4-40
batch processing, 4-68
batching multiple inbound messages, 4-45
batching multiple outbound files, 4-61
concepts, 4-37
file listing, 4-65
file listing operation, 4-66
file listing using an invoke activity, 4-77
file polling, 4-45
file read, 4-37
file read operation, 4-6
file write, 4-50
synchronous reads using an invoke activity, 4-64
configuring
for high availability, 4-78

dynamic outbound directory names, 4-54
dynamic outbound file names, 4-60
features, 4-5, 4-6
batch processing, 4-40
binary data format support, 4-6
Cobol Copybook format support, 4-6
delimited format support, 4-6
file formats supported, 4-6
fixed positional format support, 4-6
high availability, 4-24
opaque support, 4-6
supported formats, 4-6
supported naming patterns, 4-42
inbound direction, 4-37
inbound headers, 4-49
including and excluding files, 4-42, 4-69
introduction, 4-1
limitations on outbound file name lengths, 4-54
logical and physical directory paths, 4-39, 4-53, 4-67
outbound file directory, 4-52
outbound file naming conventions, 4-57
outbound headers, 4-63
outbound WSDL file, 4-63
securing Enterprise Information System credentials, 4-29
supported naming patterns, 4-69
translating native data, 4-48
write read operation, 4-7
file error handler, 2-36
file listing
using the file adapter in an invoke activity, 4-77
file names
limitations on file adapter outbound file name lengths, 4-54
file polling, 4-45
file sorting
definition, 4-8
supported with file and FTP adapters, 4-8
file-based triggers
supported with file and FTP adapters, 4-10
fixed length data
native schema, 6-12
fixed length structure
native schema, 6-76
fixed-length positional format
Native Format Builder wizard support, 6-4
FTP adapter
ActivationSpec parameters, 4-49
architecture, 4-1
archiving successfully processed files, 4-40
batch processing, 4-68
batching multiple inbound messages, 4-45
batching multiple outbound files, 4-61
concepts, 4-37
file listing operation, 4-66
file polling, 4-45
file read, 4-37
file read operation, 4-6
file write, 4-50
filelisting, 4-77
put file, 4-75
specifying connection information to an FTP server, 4-70
synchronous get file, 4-76
configuring
for high availability, 4-78
creating an Oracle Wallet, 4-88
dynamic outbound directory names, 4-54
dynamic outbound file names, 4-60
features, 4-5
batch processing, 4-40
binary data format support, 4-6
Cobol Copybook format support, 4-6
delimited format support, 4-6
file formats supported, 4-6
fixed positional format support, 4-6
high availability, 4-24
opaque support, 4-6
supported formats, 4-6
supported naming patterns, 4-42
XML format support, 4-6
file inbound file directory, 4-52
file modification times, 4-73
file write operation, 4-7
inbound direction, 4-37, 4-71
inbound headers, 4-49
including and excluding files, 4-42, 4-69
installing and configuring OpenSSL for Solaris and Linux, 4-86
installing and configuring OpenSSL for Windows, 4-90
installing and configuring vsftpd, 4-87
introduction, 4-1
logical and physical directory paths, 4-39, 4-53, 4-67
outbound direction, 4-76
outbound file naming conventions, 4-57
outbound headers, 4-63
outbound WSDL file, 4-63
restrictions on use of RESTART and RECOVERY commands, 4-71
secure FTP overview, 4-84
securing Enterprise Information System credentials, 4-29
serverType property for determining line separations during file transfers, 4-71
SSL, 4-84
supported naming patterns, 4-69
translating native data, 4-48
using secure FTP, 4-84
FTP server
specifying connection information to, 4-70
function return values
database adapter, 9-116

H

header variables
specifying, 4-54, 4-60

headerLines
construct, 6-9
headerLinesTerminatedBy
construct, 6-9
high availability
configuring file and FTP adapters, 4-78
supported with file and FTP adapters, 4-24
high availability in Inbound operations with File and FTP Adapters, 4-79
high-availability
adapters, 2-20
composites, 2-20
inbound adapters, 2-20
singleton (active/passive) inbound endpoint lifecycle support, 2-20

I

identifierLength
construct, 6-11
importing tables
database adapter, 9-15
inbound interaction, 1-18
inbound interaction error handling, 2-34
inbound non-retryable errors, 2-39
Inbound Retryable Errors, 2-36
inbound retryable errors
configuring inbound adapters to handle retryable errors, 2-36
inbound service
inbound direction for file and FTP adapters, 4-49
installing and configuring OpenSSH for Windows, 4-94
integration
Oracle Fusion Middleware, 3-3
wls, 3-1
integration with BPEL Process Manager
use case, 3-10
Integration with CICS, 10-51
invoke activity
file listing using the file adapter, 4-77
synchronous reads using the file adapter, 4-64
itemSeparatedBy
construct, 6-11

J

Java pattern letters
supported with file and FTP adapters, 4-58
JDK regular expressions
supported with the file and FTP adapters, 4-42, 4-43, 4-69
JMS adapter
accessing queues and topics, 8-43
AQ JMS text message, 8-45
concepts, 8-6
destination property, 8-7
header properties, 8-8
point-to-point, 8-6
publish/subscribe, 8-6
structure of a JMS message, 8-7
configuring JMS adapter with IBM WebSphere MQ JMS, 8-29
Non-XA Data Sources, 8-29
XA Data Sources, 8-30
configuring JMS adapter with TIBCO JMS, 8-26
Direct Connection, 8-28
NonDirect Connection, 8-26
features, 8-2
generated WSDL files, 8-21
integration with Oracle BPEL Process Manager, 8-1
integration with Oracle Mediator, 8-1
introduction, 8-1
performance tuning, 8-6
produce message procedure, 8-24
synchronous/asynchronous request reply interaction pattern, 8-44
use cases, 8-8
weblogic-ra.xml file, 8-22
WLS JMS text message, 8-32
JMS Queue, 2-35

L

last updated
database adapter, 9-67
leftsurroundedBy
construct, 6-11
legacy adapters
architecture, 1-8
deployment, 1-12
design-time components, 1-10
run-time components, 1-12
length
construct, 6-11
lists
native schema, 6-19
listTerminatedBy
construct, 6-11
logical delete
database adapter, 9-64
logical directory paths, 4-39, 4-53, 4-67
lookAhead
construct, 6-11

M

MarkReservedValue, 9-45
merge
DML operations, 9-60
Message Delivery Failure Options, 10-50
message priority
AQ adapter, 7-3
Message Properties
Message Expiry, 10-48
Message Format, 10-48
Message Persistence, 10-49
Message Priority, 10-48
Messages Types, 10-47

Message Segmentation, 10-51
Messages Type
Normal Message, 10-47
Reply Message, 10-48
Report Message, 10-48
Request Message, 10-48
Messaging Scenarios
Dequeue Message, 10-20
Request-Response Synchronous (Oracle BPEL Process Manager as Server), 10-29
Enqueue Message, 10-16
Request-Reply Asynchronous (Oracle BPEL Process Manager as Server), 10-34
Request-Response (Oracle BPEL Process Manager as a Client, 10-23
Request-Response Synchronous (Oracle Enterprise Service Bus as Server), 10-41
MQ Series
creating an adapter service that will dequeue the message from a queue, 10-83
creating an outbound adapter service, 10-90
designing the SOA composite, 10-82
MQ Series Adapter Configuration, 10-59
Adding com.ibm.mq.jar, 10-60
backout queue, 10-15
performance tuning, 10-16
RFH2 header, 10-8
weblogic-ra.xml, 10-60
XA transaction, 10-11
XA recovery, 10-12
MQ Series Concepts, 10-1, 10-3
Application Data, 10-5
Cluster, 10-2
Enqueue/Dequeue, 10-2
Message, 10-1
Message Channel, 10-2
Message Group, 10-2
Message Header, 10-4
Message Queue, 10-1
Message Segment, 10-2
Messaging, 10-1
Optional Header, 10-5
Queue Manager, 10-2
Request/Response, 10-2
Transmission Queue, 10-2
MQ Series wiring services and activities, 10-77
MQ Seroes
inbound Synchronous Request-Reply, 10-81
multiconsumer queue
AQ adapter, 7-3

N

namespaces
schema must have a namespace, 4-48, 10-20
naming patterns
supported with the file and FTP adapters, 4-42, 4-69
native correlation
about, 2-21
rejectUncorrelatedMessages, 2-22
Native Format Builder wizard
COBOL Copybook format support, 6-4
creating native schema files, 6-1
delimited format support, 6-3
DTD format support, 6-4
fixed-length positional format, 6-4
starting, 4-48, 10-20
supported formats, 6-2
Native MQ Series Adapter, 10-5
Concepts, 10-16
Configuring, 10-59
Correlation Schemas, 10-49
Distribution List Support, 10-49
Integration with CICS, 10-51
Integration with Oracle BPEL Process Manager, 10-6
Integration with Oracle Mediator, 10-7
Message Delivery Failure Options, 10-50
Message Properties, 10-47
Message Segmentation, 10-51
Messaging Scenarios, 10-16
Need, 10-5
Report Messages, 10-49
native schema files
creating, 6-1
native schemas
arrayIdentifierLength construct, 6-10
arrayLength construct, 6-10
arrays, 6-20
arrayTerminatedBy construct, 6-10
assign construct, 6-10
byteOrder construct, 6-9
cellSeparatedBy construct, 6-10
choiceCondition construct, 6-10
complex structure, 6-83
conditional processing, 6-26
conditionValue construct, 6-10
constructs, 6-9
CSV, 6-63
dataLines construct, 6-10
dateFormat construct, 6-11
dates, 6-34
encodeLineTerminators construct, 6-10
encoding construct, 6-9
fieldValidation construct, 6-10
fixed length data, 6-12
fixed length structure, 6-76
headerLines construct, 6-9
headerLinesTerminatedBy construct, 6-9
identifierLength construct, 6-11
itemSeparatedBy construct, 6-11
leftsurroundedBy construct, 6-11
length construct, 6-11
lists, 6-19
listTerminatedBy construct, 6-11
lookAheadconstruct, 6-11
outboundHeader construct, 6-10
paddedBy construct, 6-11
padStyle construct, 6-11

parseBom construct, 6-10
quotedBy construct, 6-11
rightsurroundedBy construct, 6-11
separated value file structure, 6-75
skip construct, 6-11
skipLines construct, 6-11
skipUntil construct, 6-11
standalone construct, 6-9
startsWith construct, 6-11
stream construct, 6-9
style construct, 6-11
surrounded data, 6-17
surroundedBy construct, 6-11
terminated data, 6-15
terminatedBy construct, 6-11
uniqueMessageSeparator construct, 6-9
useArrayIdentifiers construct, 6-10
validateNxsd construct, 6-10
validation construct, 6-10
variable construct, 6-11
variables, 6-36
variables construct, 6-11
version construct, 6-9
xmlversion construct, 6-10
normalized messages
native correlation, 2-21
null values
database adapter, 9-116

O

opaque format
supported with file and FTP adapters, 4-6
OpenSSL
installing and configuring for Solaris and Linux, 4-86
installing and configuring for Windows, 4-90
Oracle AQ Adapter Properties, A-8
Oracle Database Adapter Properties, A-13
Oracle File and FTP Adapters
File Formats, 4-6
FTP Servers, 4-6
Inbound and Outbound Interaction, 4-6
Oracle File and FTP Adapters Properties, A-1
Oracle FTP Adapter
configuring a pipeline, 4-11
configuring batch notification handler, 4-19
Nontransactional, 4-8
Payload Support, 4-9
Security, 4-8
Oracle FtP Adapter
proxy support, 4-8
Oracle Fusion Middleware
integration with adapters, 3-5
Oracle JCA Adapter for Files
See file adapter, 4-1
Oracle JCA Adapter for FTP
See FTP adapter, 4-1
Oracle JCA Adapter for Sockets
See socket adapter, 5-1
Oracle JMS Adapter Properties, A-10
Oracle Mediator
overview, 3-5
Oracle SOA Composite
integration with adapters, 3-13
overview, 3-13
Oracle Socket Adapter Properties, A-7
Oracle Wallet
installing and configuring, 4-88
OracleAS Adapter for AQ
See AQ adapter, 7-1
Oralcle File and FTP Adapters
File Debatching, 4-7
outbound adapter interaction error handling, 2-40
outbound interaction, 1-18
outboundHeader
construct, 6-10
OutputCompletedXml Feature, 9-48
overloading, 9-90

P

packaged-application adapters
architecture, 1-13
deployment, 1-17
design-time components, 1-15
run-time components, 1-16
paddedBy
construct, 6-11
padStyle
construct, 6-11
parseBom
construct, 6-10
payload
size threshold, 2-23
streaming, 2-24
payload schema
AQ adapter, 7-7
payload size
changing global payload size to a finite value, 2-24
limitations, 2-23
payloadSizeThreshold, 2-23
performance tuning
AQ adapter, 7-10
database adapter, 9-46
file adapters, 4-24
FTP adapters, 4-24
JMS adapter, 8-6
MQ Series adapter, 10-16
socket adapter, 5-13
physical delete
database adapter, 9-62
polling strategies
database adapter, 9-62
pre-processing and post-processing
supported with file and FTP adapters, 4-10
produce message procedure
JMS adapter, 8-24

Q

query by example
DML operations, 9-60
quotedBy
construct, 6-11

R

REF CURSOR support, 9-38
regex constructs
supported, 4-42, 4-43, 4-69
rejected messages
checking, 2-32
configuring, 2-31
rejectUncorrelatedMessages, 2-22
relational-to-XML mappings
for database adapter, 9-50
Report Messages, 10-49
Confirmation on Arrival, 10-50
Confirmation on Delivery, 10-50
Exception Report, 10-50
Expiry Report, 10-50
Negative Action Notification, 10-50
Positive Action Notification, 10-50
rightsurroundedBy
construct, 6-11
row set
about, 9-38
cursor variable, 9-38
strongly typed XSD, 9-38, 9-118
weakly typed XSD, 9-38, 9-127
row set support
database adapter, 9-118, 9-127
strongly typed XSD, 9-118
weakly typed XSD, 9-127
run time
packaged-application adapters, 1-16
technology adapters, 1-7

S

scal, 9-45
scalability
distributed polling
MarkReservedValue, 9-45
SKIP LOCKED, 9-41
tuning on single node first, 9-45
schemas
must have a namespace, 4-48, 10-20
secure FTP
using with the FTP adapter, 4-84
separated value file structure
native schema, 6-75
sequencing table
database adapter, 9-67
sequencing table updates
database adapter, 9-27
serverType property
determines line separations during file transfers, 4-71
service names
creating, 4-3, 4-4, 5-4, 10-6, 10-7
setting the Trace Level of Oracle JCA Adapters, 2-43
setting up Oracle FTP Adapter for SFTP, 4-95
SFTP Overview, 4-93
singleton, 2-20
singleton (active/passive) inbound endpoint lifecycle support, 2-20
skip
construct, 6-11
SKIP LOCKED, 9-41
skipLines
construct, 6-11
skipUntil
construct, 6-11
Socket Adapter
creating an outbound Oracle Socket Adapter reference, 5-32
specifying a TCP port in a configuration plan, 5-40
socket adapter
architecture, 5-1
communication modes, 5-4
inbound receive, 5-6
inbound synchronous request/response, 5-5
outbound invoke, 5-6
outbound synchronous request/response, 5-5
concepts, 5-4
byte order, 5-12
character encoding, 5-12
defining protocols, 5-6
configuring, 5-17
configuration properties, 5-18
designing XSL using the XSLT mapper tool, 5-20
modeling a handshake, 5-19
weblogic-ra.xml file, 5-17
defining protocols
handshake mechanism using custom Java code, 5-9
handshake mechanism using style sheet, 5-6
without handshake mechanism, 5-11
designing XSL
for inbound synchronous request/reply, 5-20
for outbound synchronous request/reply, 5-31
features, 5-4
integration with Oracle BPEL Process Manager, 4-5, 5-2, 5-4
integration with Oracle Mediator, 5-2
integration with SOA composite, 4-5, 5-4
introduction, 5-1
modeling an inbound handshake, 5-19
modeling an outbound handshare, 5-19
performance tuning, 5-13
sorting
definition, 4-8
supported with file and FTP adapters, 4-8
SSL
creating an Oracle Wallet, 4-88
installing and configuring OpenSSL for Solaris and

Linux, 4-86
installing and configuring OpenSSL for Windows, 4-90
installing and configuring vsftpd, 4-87
secure FTP overview, 4-84
using secure FTP with the FTP adapter, 4-84
standalone
construct, 6-9
startsWith
construct, 6-11
stored procedures and functions
artifcat generation, 9-103
complex user-defined types, 9-109
JCA file, 9-105
object references, 9-110
object type inheritance, 9-109
Oracle data types, 9-106
referencing types in other schemas, 9-110
user-defined types, 9-107
WSDL and XSD, 9-104
XSD attributes, 9-106
XSD pruning optimization, 9-111
database adapter, 9-79
invocation at run time, 9-115
overloaded procedures, 9-90
run time, 9-112
searching for, 9-84
third-party database functionality at run time, 9-116
stream
construct, 6-9
style
construct, 6-11
supported third-party databases
DB2 Data Types, 9-95
IBM DB2 AS/400, 9-97
Microsoft SQL Server 2000 and 2005, 9-94
MySQL v5.x and v6.x, 9-98
terms used, 9-93, 9-94
surrounded data
native schema, 6-17
surroundedBy
construct, 6-11
Synchronous, 2-18
synchronous reads
using the file adapter in an invoke activity, 4-64

T

technology adapters
architecture, 1-4
deployment, 1-7
design-time components, 1-5
run-time components, 1-7
terminated data
native schema, 6-15
terminatedBy
construct, 6-11
threading models
supported with file and FTP adapters, 4-22
translation
native data, 4-62
not required, 4-48, 10-20
of native data, 4-48
requires native schemas, 6-1
translator
converts native data to XML and back, 4-6

U

uniqueMessageSeparator
construct, 6-9
updating a field in a table
database adapter, 9-25
use cases
database adapter, 9-133
file and FTP adapters, 4-104
JMS adapter, 8-8
socket adapter, 5-41
stored procedures, 9-135
using the AQ adapter, 7-12
using the AQ adapter ADT queue, 7-48
using the AQ adapter RAW queue, 7-65
useArrayIdentifiers
construct, 6-10
using the Native Format Translator for removing or adding namespaces to XML with no namespace, 6-100

V

validateNxsd
construct, 6-10
validation
construct, 6-10
value binding
database adapter, 9-112
variable
construct, 6-11
variables
construct, 6-11
native schema, 6-36
version
construct, 6-9
vsftpd server
installing and configuring, 4-87

W

Web Service rejection handler, 2-34
Web services
SQL operations as, 9-60
weblogic-ra.xml file, 10-60
configuring a database connection in, 9-10
for socket adapter, 5-17
JMS connection factory definitions, 8-22
modifying, 5-17
WHERE clause
database adapter, 9-22
WLS
integration with adapters, 3-3

overview, 3-1
WLS JMS Text Message
meeting prerequisites, 8-32
creating an application and an SOA project, 8-34
creating an Application Server connection, 8-34
creating an outbound adapter service, 8-39
wiring services and activities, 8-39
creating an inbound adapter service, 8-35
creating queues in the Oracle WebLogic Server administration console, 8-32
creating the Q2Qorders.xsd file, 8-33
deploying with JDeveloper, 8-53
monitoring using the Fusion Middleware Control Console, 8-53
WSDL, 1-13
WSDL file
outbound direction for file and FTP adapters, 4-63
WSDL files
JMS adapter, 8-21
specifying logical directory paths, 4-39, 4-53, 4-67
wsdlLocation, A-22

X

XML format support, 4-6
xmlversion
construct, 6-10