Oracle® Enterprise Service Bus

Developer's Guide

10*g* (10.1.3.4.0)

E12638-02

October 2008

ORACLE®

Oracle Enterprise Service Bus Developer's Guide, 10*g* (10.1.3.4.0)

E12638-02

Copyright © 2006, 2008, Oracle. All rights reserved.

Primary Author: Tulika Das, Vimmika Dinesh, Rima Dave

Contributors: Abhimanyu Prabhavalkar, Amitabh Nandan, Aninda Sengupta, Ashwin Patel, Brian Volpi, Dave Berry, Demed L'Her, Dhaval Parikh, Eric Belden, Harish Vinayachandran, Ingrid Stuart, Mahesh Narayana, Maneesh Joshi, Nishad Desai, Prasad Dixit-Hardikar, Rakesh Saha, Santhosh Kumar, Seegler Ittyavirah, Sivaraj Subbaiyan, Zahid Syed, Atul Singh, Qiujie Li, Chandrashekar Venkatachar

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States 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, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

Preface xv

Audience xv
Documentation Accessibility xv
Related Documents xvi
Conventions xvi

What's New xvii

New Features in Release 10.1.3.4 xvii

1 Introduction to Oracle Enterprise Service Bus

Oracle Enterprise Service Bus Concepts Overview 1-1
Oracle Enterprise Service Bus Integration Features 1-2
Connectivity 1-2
Document Transformation 1-4
Content-Based and Header-Based Routing 1-4
Creating, Configuring, and Managing Oracle Enterprise Service Bus 1-5
Introduction to Oracle JDeveloper 1-6
Introduction to the Oracle Enterprise Service Bus Control 1-7
Oracle Enterprise Service Bus Architecture 1-8
Instance Tracking in Oracle Enterprise Service Bus 1-9
Activity Types 1-9
Activity Messages 1-10
Database tables 1-11
Sample Oracle Enterprise Service Bus Scenario 1-12
Starting, Stopping, and Accessing Oracle Enterprise Service Bus Components 1-13
Starting and Stopping the ESB Server 1-13
Windows Installation 1-14
Linux Installation 1-14
Opening Oracle JDeveloper 1-14
Opening the Oracle ESB Control 1-15

2 Developing the Enterprise Service Bus

Overview of Oracle JDeveloper 2-1

Overview of Connection Navigator Tab 2-1
Overview of the Application Navigator Tab 2-1
Overview of the Design Tab and Component Palette 2-2
Overview of Service Definitions and Routing Rules 2-4
Getting Started with Oracle JDeveloper 2-5
Creating and Testing Connections 2-6
Viewing Port Numbers 2-6
Creating an Application Server Connection 2-7
Creating an Integration Server Connection 2-8
Creating a Database Connection 2-8
Testing a Connection 2-10
Creating Applications and ESB Projects 2-11
Creating ESB Systems and Service Groups 2-12
Creating ESB Systems 2-12
Creating ESB Service Groups 2-13
Adding Project Content 2-14
Importing Files into a Project 2-14
Registering ESB Projects and Services with the ESB Server 2-15
Syncing Services From ESB Server 2-15
Deleting ESB Projects 2-16

3 Monitoring the Enterprise Service Bus

Overview of the Oracle ESB Control 3-1
Understanding the Layout of the Oracle ESB Control 3-3
Oracle ESB Control Services View 3-3
Oracle ESB Control Instances View 3-5
Oracle ESB Control Maps View 3-6
Creating, Viewing, and Updating Organizational Units for Services 3-7
Managing Oracle Enterprise Service Bus Systems and Service Groups 3-8
Creating an ESB System 3-8
Viewing or Modifying an Existing ESB System Definition 3-9
Creating Service Groups 3-10
Viewing or Modifying an Existing Service Group 3-11
Deleting Systems or Service Groups 3-11
Viewing and Updating Service Definitions 3-12
Enabling and Disabling Services 3-12
Viewing Service Definitions 3-12
Understanding and Managing Routing Rules 3-14
Creating or Modifying Routing Rules 3-14
Viewing Routing Rules 3-15
Deleting Routing Rules 3-16
Defining and Managing Tracking Fields 3-16
Understanding the Trackable Fields Tab 3-17
Defining and Updating Trackable Fields 3-17
Using the Expression Builder to Specify a Trackable Field Expression 3-18
Enabling and Disabling Trackable Fields 3-19
Deleting Trackable Fields 3-20

4 Creating Inbound and Outbound Services

Configuring Adapter Services with Oracle Enterprise Service Bus 4-1
Using Adapter Services 4-2
Creating Adapter Services 4-3
Modifying Adapter Services 4-4
Deleting Adapter Services 4-4
Example: Creating an Inbound File Adapter 4-4
Using SOAP Invocation Services 4-7
Creating a SOAP Service 4-7
Modifying SOAP Services 4-9
Deleting SOAP Services 4-9
Browsing for Deployed Services 4-10
Using Endpoint Properties 4-11
Specifying Endpoint Properties 4-11
Creating a BPEL Partner Link to an ESB Service 4-12
Calling an ESB Service From an External Service 4-13

5 Creating Routing Services and Routing Rules

Introduction to Routing Services and Routing Rules 5-1
Overview of Specifying the Routing Service WSDL 5-2
Modifying the Service WSDL File 5-4
Specifying Routing Service Properties 5-6
Overview of Specifying Routing Rules 5-7
Target Service and Operation Overview 5-9
Filter Expression Overview 5-10
Transformation Overview 5-12
Accept Messages From Overview 5-13
Routing Invocation Type Overview 5-13
Routing Rule Priority Overview 5-13
Creating and Modifying Routing Services 5-13
Opening the Create Routing Service Dialog 5-13
Specifying the WSDL File for a Routing Service 5-14
Generating the WSDL for a Routing Service from an Existing XSD File 5-14
Generating the WSDL to Create a Routing Service Based on a Sample File 5-16
Selecting an Existing WSDL to Create a Routing Service 5-17
Specifying Routing Rules 5-18
Specifying the Target Operations 5-19
Creating an XSL Map File for Data Structure Transformation 5-20
Using An Expression for Filtering Messages Based on Payload 5-20
Specifying the ESB Systems From which Messages are Accepted 5-26
Specifying Synchronous or Asynchronous Execution 5-27
Specifying Routing Rules Priority 5-27
Header Transformation and Filtering 5-27
Header Support Terminology 5-27
Header-based Transformation 5-28
Header-Based Filtering 5-31

Limitations of ESB Header Support 5-32
Modifying a Routing Service 5-33
Deleting a Routing Service 5-33

6 XSLT Data Mapper and Transformations

XSLT Data Mapper 6-1
Notes on the Mapper 6-3
Creating an XSL Map with Data Mapper 6-3
Using the XSLT Mapper 6-6
Simple Copy by Linking Nodes 6-7
Setting Constant Values 6-7
Adding Functions 6-8
Editing Function Parameters 6-9
Chaining Functions 6-10
Named Templates 6-11
Importing User-Defined Extension Functions 6-11
Creating User-Defined Extension Functions 6-11
Editing XPath Expressions 6-13
Adding XSLT Constructs 6-14
Conditional Processing with xsl:if 6-14
Conditional Processing with xsl:choose 6-15
Handling Repetition or Arrays 6-16
Automatically Mapping Nodes 6-17
Auto Map with Confirmation 6-18
Viewing Unmapped Target Nodes 6-19
Generating Dictionaries 6-20
Creating Map Parameters and Variables 6-20
Creating a Map Parameter 6-21
Creating a Map Variable 6-21
Searching Source and Target Nodes 6-22
Ignoring Elements in the XSLT Document 6-23
Replacing a Schema in the XSLT Mapper 6-24
Using Instance Id in the XSLT Mapper 6-24
Using the Mapper Test Utility 6-24
Testing a Map 6-24
Generating Reports 6-26
Correcting Memory Errors When Generating Reports 6-27
Sample XML Generation 6-27

7 Domain-Value Maps

Understanding Domain-Value Maps 7-1
Creating and Populating Domain-Value Maps 7-1
Creating a New Domain-Value Map from Scratch 7-2
Exporting a Domain-Value Map 7-4
Domain-Value Map Template and XSD Files 7-5
Importing an Existing Domain-Value Map File 7-7
Importing Rows Into a Domain-Value Map 7-8

Editing a Domain-Value Map 7-10
Editing the Name of a Domain-Value Map 7-10
Adding Rows or Columns to a Domain-Value Map 7-11
Deleting a Row from a Domain-Value Map 7-11
Deleting a Column from a Domain-Value Map 7-11
Renaming a Column in a Domain-Value Map 7-11
Reordering the Columns in a Domain-Value Map 7-11
Resetting a Domain-Value Map to Its Last Saved State 7-12
Resizing Columns in a Domain-Value Map 7-12
Deleting a Domain-Value Map 7-12
Using a Domain-Value Map in a Transformation 7-12

8 Creating Cross References

Introduction to Cross References 8-1
Introduction to the Cross Reference Command-Line Utility 8-4
Creating, Modifying, and Deleting Cross Reference Table 8-5
Modifying Cross Reference Tables 8-5
Deleting a Cross Reference Table 8-5
Populating Cross Reference Tables 8-6
xref:populateXRefRow Function 8-6
Using the xref:populateXRefRow Function 8-8
xref:populateXRefRow1M Function 8-10
Looking Up Cross Reference Tables 8-12
xref:lookupXRef Function 8-12
Using the xref:lookupXRef Function 8-12
xref:lookupXRef1M Function 8-14
Deleting Cross Reference Table Values 8-15
Importing and Exporting Cross References 8-17
Exporting Cross Reference Tables 8-17
Importing Cross Reference Tables 8-17
Schema Definition (XSD) File for Cross References 8-18

9 Administering the Enterprise Service Bus

Administrative Stages 9-1
Planning Resources for the ESB 9-2
Understanding Oracle Enterprise Service Bus Clusters 9-2
Providing Security 9-4
Setting Up Notification Channels 9-4
Specifying Notification Channels 9-4
Configuring the Email Notification 9-5
Configuring the Wireless Provider for Voice 9-7
Configuring Paging Notification 9-9
Configuring Mobile Notification 9-9
Configuring Phone Notification 9-9
Configuring the FAX Cover Page 9-9
Testing the ESB Services 9-9

Checking Log Files ... 9-11
Viewing Log Files ... 9-12
Configuring Log Files ... 9-13
Moving the ESB Instance to a Different Oracle Home ... 9-13
Using the ESB Import and Export Utilities ... 9-15
Configuring the InterConnect Adapter with ESB ... 9-16

10 Tracking Message Instances Across the Enterprise Service Bus

Overview of the Oracle ESB Control Instances View ... 10-1
Understanding Instances View Elements and Controls ... 10-2
Administering Message Instances ... 10-5
Enabling and Disabling Instance Tracking ... 10-6
Viewing Instance Details ... 10-6
Viewing Instance Statistics ... 10-7
Searching for Message Instances ... 10-7
Purging Message Instances ... 10-9

11 Support for Resequencing

Overview of the Resequencer ... 11-1
Resequencing Order ... 11-1
Resequencer Thread Pool ... 11-3
Using the Resequencer ... 11-3
Limitations in the Resequencer ... 11-7
Configuring Database Adapters and File Adapters for FIFO ... 11-8
File Adapter ... 11-8
Database Adapter ... 11-9

12 Error Handling

Overview of Error Handling ... 12-1
Managing Error Conditions ... 12-2
Inbound Adapter Error Handling ... 12-2
User Error Handling ... 12-2
Resubmitting Messages on Errors ... 12-4
ESB Error Handling Enhancement ... 12-5
Custom Rejection Handler for Third-Party Adapters ... 12-5
Shutdown of the Inbound Adapter ... 12-5
Enhance the Information in ERROR_TOPIC ... 12-6
Reverse Control ... 12-6
Primary Key ... 12-6
Removal of Message from the Queue on Rejection ... 12-7

13 Enterprise Service Bus Ant Deployment Feature

Overview of Ant Deployment Feature ... 13-1
Ant Installation ... 13-1
Deployment of Ant Tasks ... 13-2
ESB Ant Deployment Custom Tasks ... 13-4

deployESBProjects 13-4
undeployESBEntities 13-6
extractESBDeploymentPlan 13-7
deployESBSuitcase 13-9

A XPath Extension Functions

add-dayTimeDuration-to-dateTime A-2
compare A-2
compare-ignore-case A-3
create-delimited-string A-3
current-date A-4
current-dateTime A-4
current-time A-4
day-from-dateTime A-5
doc A-5
ends-with A-5
format-dateTime A-6
format-string A-6
generate-guid A-6
get-content-as-string A-7
get-localized-string A-7
getInboundResponseHeader A-8
getRequestHeader A-8
hours-from-dateTime A-8
implicit-timezone A-9
index-within-string A-9
last-index-within-string A-9
left-trim A-10
lookup-dvm A-10
lookup-table A-11
lookup-xml A-12
lower-case A-12
minutes-from-dateTime A-12
month-from-dateTime A-13
query-database A-13
right-trim A-14
seconds-from-dateTime A-14
sequence-next-val A-14
setOutboundHeader A-15
setResponseHeader A-15
square-root A-16
subtract-dayTimeDuration-from-dateTime A-16
timezone-from-dateTime A-16
upper-case A-17
year-from-dateTime A-17

B Oracle Enterprise Service Bus API

ConsoleClientFactory Class B-1
ConsoleClient Interface B-1
Perform Function B-1
Using Perform Function B-2
Get the list of instances B-2
Get the list of errored instances B-3
Get all the errors occurred in a flow id B-3
Get the XML to draw the instance diagram for a flow id B-4
Resubmit a list of instances by Ids B-4
Resubmit an instance by Id B-5
Resubmit an instance with modified/unmodified payload B-5
Get the current status of the instance tracking B-7
Enable/disable instance tracking B-7
Purge instances based on time B-7
XML Schema File B-8

C Resubmission APIs

Get the List of Instances C-1
Get the List of Instances That Raised an Error C-2
Get All the Errors That Occurred in a Given Flow C-2
Get the XML to Draw the Instance Diagram C-3
Resubmit a List of Instances by IDs C-3
Resubmit an Instance by ID C-4
Resubmit an Instance with Modified or Unmodified Payload C-4
Get the Current Status of Instance Tracking C-5
Enable or Disable Instance Tracking C-5
Purge the Instances Based on Time C-6
Retrieve All Service Metadata C-6
Update Service Status C-7

Index

List of Figures

1–1 Sample Data Mapper Tool 1-4
1–2 Sample Oracle JDeveloper 1-7
1–3 Oracle ESB Control - Services View 1-8
1–4 ESB Architecture - Single Instance Environment 1-9
1–5 Instance Tracking in ESB 1-10
1–6 Illustration of a Sample ESB Scenario 1-13
1–7 Oracle JDeveloper – Initial Display 1-15
1–8 Oracle ESB Control - Initial Display 1-16
2–1 Sample Application Navigator 2-2
2–2 Oracle JDeveloper Design Tab and Component Palette 2-3
2–3 Create Routing Service Dialog Box 2-4
2–4 Sample Property Sheet for a Routing Service 2-5
3–1 Oracle ESB Control - Services View Diagram Tab 3-3
3–2 Oracle ESB Control - Instances View Tracking Tab 3-5
3–3 Oracle ESB Control - Instances View Errors Tab 3-6
3–4 Oracle ESB Control - Maps View with Sample Domain-Value Map 3-7
3–5 Icons Used in the Oracle ESB Control 3-8
3–6 ESB System – Definition Tab 3-9
3–7 Services View – Trackable Fields Tab 3-18
3–8 Trackable Fields - Expression Builder 3-19
4–1 Adapter Configuration Wizard - Messages Page 4-6
4–2 Create File Adapter Service Dialog 4-6
4–3 Automatic Routing Service Creation 4-7
4–4 Oracle Application Server Control – Applications 4-8
4–5 Service Explorer Dialog 4-10
4–6 BPEL Partner Link 4-13
4–7 Oracle ESB Control – Definition Tab of Services View 4-14
5–1 Creating a Routing Service 5-2
5–2 Create Routing Service Dialog Box 5-3
5–3 Routing Service 5-4
5–4 Refresh WSDL Dialog 5-5
5–5 Populated Refresh WSDL Dialog 5-5
5–6 Modified Routing Service 5-6
5–7 Routing Rules Priority 5-7
5–8 Routing Rules Properties Page 5-7
5–9 Routing Rules – Request/Reply/Response Schema 5-8
5–10 Routing Rules Icons, Fields, and Options 5-8
5–11 Browse Target Service Operation 5-10
5–12 Expression Builder Window – Initial View 5-11
5–13 Type Chooser Dialog 5-15
5–14 Create Routing Service - Request Tab 5-16
5–15 Routing Rule Property Sheet Icons 5-18
5–16 Routing Rules for Request/Reply with Target Service Specified 5-20
5–17 Sample Expression Builder Tool – WSDL Message Element Selected 5-21
5–18 Sample Expression Builder Tool – WSDL Message Element Inserted 5-22
5–19 Sample Expression Builder Tool – Function Selected 5-23
5–20 Sample Expression Builder Tool – Function Inserted 5-24
5–21 Sample Expression Builder Tool – Value Entered 5-25
5–22 Expression Editing Icons 5-25
5–23 Request and Outbound Headers 5-28
5–24 Response and Inbound Response Headers 5-28
5–25 Create Variable Dialog With a Header Transformation Function 5-30
5–26 Expression Builder With a Header Transformation Function 5-32
6–1 Sample XSLT Data Mapper Tool 6-2

6–2 Request Transformation Map Dialog 6-4
6–3 Auto Map Preferences Dialog 6-4
6–4 XSLT Data Mapper AutoMap 6-5
6–5 XSLT Data Mapper Context Menu 6-6
6–6 Linking Nodes 6-7
6–7 Set Text Window 6-8
6–8 Using the Concat Function 6-9
6–9 Editing Function Parameters 6-10
6–10 Chaining Functions 6-10
6–11 The XPath Building Assistant 6-14
6–12 Conditional Processing with xsl:if 6-15
6–13 Conditional Processing with xsl:choose 6-16
6–14 Handling Repetition or Arrays 6-17
6–15 Auto Mapping Candidates 6-18
6–16 Auto Map with Confirmation 6-19
6–17 Completion Status 6-20
6–18 Test XSL Map Dialog 6-25
6–19 Test Window 6-26
6–20 The Generate Report Dialog 6-27
7–1 Oracle ESB Control Map View – Create New Map 7-3
7–2 Domain-Value Map – Rows Created 7-4
7–3 Maps View – Import a New Map Dialog 7-8
7–4 Sample Domain-Value Map with a Conflict 7-8
7–5 Oracle ESB Control Map View – Import into an Existing Map Dialog 7-9
7–6 Oracle ESB Control Map View – Updated Map After Importing Rows 7-10
7–7 Arrows for Rearranging Columns In Domain-Value Maps 7-11
7–8 Data Mapper – Source to Target Map 7-13
7–9 Component Palette – lookup-dvm Function 7-13
7–10 Data Mapper – look-up dvm Function Added 7-14
7–11 Edit Function – lookup-dvm 7-14
7–12 Data Mapper – look-up dvm Function Defined 7-15
8–1 Common Value Integration Pattern Example 8-2
8–2 Expression Builder Dialog Box with Cross Reference Functions 8-3
8–3 XSLT Mapper Dialog Box with Cross Reference Functions 8-4
8–4 The Edit Function – populateXRefRow Dialog Box 8-9
8–5 The Populated Edit Function – populateXRefRow Dialog Box 8-10
8–6 The Edit Function – lookupXRef Dialog Box 8-13
8–7 Populated Edit Function – lookupXRef Dialog Box 8-14
8–8 Edit Function – markForDelete Dialog Box 8-16
8–9 Populated Edit Function – markForDelete Dialog Box 8-16
9–1 ESB Architecture - Clustered Environment 9-3
9–2 ESB System – Definition Tab 9-5
9–3 Oracle Enterprise Manager Web Services 9-10
9–4 Oracle Enterprise Manager Test Web Service 9-10
9–5 Oracle Enterprise Manager Test Page 9-11
9–6 Oracle Application Server Control – OC4J Log Files 9-12
9–7 Oracle Application Server Control – Logger Configuration 9-13
10–1 Instances View – Tracking Tab and Details Subtab 10-2
10–2 Instances View – Manage Panel 10-2
10–3 Instances View – Search Panel 10-3
10–4 Instances Panel Showing Status Codes 10-3
10–5 Oracle ESB Control – Instances View with Error 10-4
10–6 Instances View – Errors Tab 10-5
10–7 Sample Overlay Statistics in Instances View 10-7
12–1 Instances View – Tracking Tab 12-3

12–2 Error Details Dialog 12-3
12–3 Instances View - Errors Tab 12-4

List of Tables

1–1 Oracle Adapter Services 1-3
3–1 Comparison of Features Available in the Oracle ESB Control and Oracle JDeveloper 3-2
3–2 ESB System Definition Tab 3-10
3–3 ESB Service Group Definition Tab 3-11
3–4 Service Definition – General Region 3-13
3–5 Service Definition - Operations Region 3-13
3–6 Service Definition - Operation Details Region 3-14
3–7 Routing Rules Tab in Oracle ESB Control 3-15
4–1 Summary of Oracle Technology Adapters 4-2
8–1 Cross Reference Table Sample 8-1
8–2 Cross Reference Table with the Common Column 8-2
8–3 xreftool Commands 8-5
8–4 xreftool Commands for Modifying a Cross Reference Table 8-5
8–5 xref:populateXRefRow Function Modes 8-7
8–6 xref:populateXRefRow Function Results with Different Modes 8-8
8–7 A Cross Reference Table with Multiple Column Values 8-10
8–8 xref:populateXRefRow1M Function Modes 8-11
8–9 xref:populateXRefRow1M Function Results with Different Modes 8-11
9–1 XML Elements for the E-mail Notification Configuration File 9-6
9–2 XML Elements for the Voice Notification Configuration File 9-7
10–1 Specifying Message Instance Search Criteria 10-7
12–1 Properties to Be Set for Shutting Down the Inbound Adapter 12-5
12–2 Property Relationship Table 12-8

Preface

This guide provides information about Oracle Enterprise Service Bus. The information includes how to use Oracle JDeveloper and Oracle ESB Control to create and manage Oracle Enterprise Service Bus.

Audience

This document is intended for all users interested in learning about and using Oracle Enterprise Service Bus.

Documentation Accessibility

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

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, 7 days a week. For TTY support, call 800.446.2398. Outside the United States, call +1.407.458.2479.

Related Documents

For more information, see these Oracle resources:

- *Oracle Enterprise Service Bus Quick Start Guide*
- *Oracle Enterprise Service Bus Installation Guide*
- *Oracle Application Server Adapter Concepts*
- *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide*
- *Oracle Application Server Adapter for Oracle Applications User's Guide*
- *Oracle BPEL Process Manager Developer's Guide*
- *Oracle Application Server Administrator's Guide*
- *Oracle Application Server Performance Guide*
- *Oracle Application Server Enterprise Deployment Guide*
- *Oracle Application Server High Availability Guide*
- *Oracle Web Services Manager Administrator's Guide*
- *Oracle Application Server Installation Guide for Microsoft Windows*
- *Oracle Application Server Installation Guide for Linux x86*

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN):

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

Conventions

The following text conventions are used in this document:

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

What's New

This chapter describes the new features of Enterprise Service Bus (ESB) in Oracle Application Server release 10.1.3.4.

New Features in Release 10.1.3.4

In this release, Oracle ESB supports the following new features:

- Support for Creating User-Defined Extension Functions
- Support for Resequencing
- ESB Error Handling Enhancement
- Enterprise Service Bus Ant Deployment

Support for Creating User-Defined Extension Functions

This release enables you to create your own set of user-defined extension functions. For more information refer "Creating User-Defined Extension Functions" on page 6-11.

Support for Resequencing

This release provides support for resequencer to rearrange a stream of related, but out of sequence messages back into order. For more information refer "Overview of the Resequencer" on page 11-1.

ESB Error Handling Enhancement

This release significantly enhances the Error Handling feature. For more information refer "ESB Error Handling Enhancement" on page 12-5.

Enterprise Service Bus Ant Deployment

This release provides support for Enterprise Service Bus Ant deployment. For more information refer "Overview of Ant Deployment Feature" on page 13-1.

1

Introduction to Oracle Enterprise Service Bus

This chapter introduces Oracle Enterprise Service Bus features, concepts, and components.

This chapter contains the following topics:

- Oracle Enterprise Service Bus Concepts Overview on page 1-1
- Creating, Configuring, and Managing Oracle Enterprise Service Bus on page 1-5
- Oracle Enterprise Service Bus Architecture on page 1-8
- Sample Oracle Enterprise Service Bus Scenario on page 1-12
- Starting, Stopping, and Accessing Oracle Enterprise Service Bus Components on page 1-13

Oracle Enterprise Service Bus Concepts Overview

An enterprise service bus moves data among multiple endpoints, both within and outside of an enterprise. It uses open standards to connect, transform, and route business documents as Extensible Markup Language (XML) messages, among disparate applications. It enables monitoring and management of business data, with minimal impact on existing applications. An enterprise service bus is the underlying infrastructure for delivering a service-oriented architecture (SOA) and event-driven architecture (EDA).

Oracle Enterprise Service Bus is the foundation for services using SOA and EDA. At its core, it is a loosely coupled application framework that provides your business with increased flexibility, reusability, and overall responsiveness in a distributed, heterogeneous, message-oriented environment using industry standards.

Oracle Enterprise Service Bus is a component of Oracle SOA Suite. Oracle SOA Suite is a standards-based suite that provides an integrated design-time environment and a common architecture for developing enterprise applications. Oracle SOA Suite enables services to be created, managed, and orchestrated into composite applications and business processes.

Oracle Enterprise Service Bus contains the following components:

- ESB Server

  The ESB Server is the runtime server which listens on the control topic for updates from the ESB Metadata Server and updates its cache.

- Oracle ESB Control

The Oracle ESB Control provides a Web-based interface for managing, administering, and monitoring services that you have registered with the ESB Metadata Server.

- ESB Metadata Server

  The database that holds your ESB metadata such as schemas, transformations, and routing rules. The ESB Metadata Server is the server to which you register the ESB services that you have designed using Oracle JDeveloper and configured using Oracle ESB Control.

- Oracle JDeveloper

  Oracle JDeveloper is a graphical and user-friendly way to model, edit, and design the services that comprise an Oracle Enterprise Service Bus system.

Oracle Enterprise Service Bus Integration Features

Oracle Enterprise Service Bus features that provide the ability to integrate applications fall into three broad categories:

- Connectivity on page 1-2
- Document Transformation on page 1-4
- Content-Based and Header-Based Routing on page 1-4

Connectivity

Connectivity is provided through adapter services and Simple Object Access Protocol (SOAP) invocation services, as described in the list that follows:

- SOAP invocations services

  SOAP invocation services provide connectivity with external SOAP clients, such as Oracle BPEL Process Manager, Apache Axis, and Microsoft .NET. You can call Oracle Enterprise Service Bus services from such clients, or you can call those products from Oracle Enterprise Service Bus.

  You might call Oracle Enterprise Service Bus from Oracle BPEL Process Manager, for example, to take advantage of the document routing features that Oracle Enterprise Service Bus provides, or you might call Microsoft .NET from Oracle Enterprise Service Bus to integrate a legacy Microsoft .NET infrastructure.

- WSIF

  Oracle Enterprise Service Bus utilizes WSIF bindings in a WSDL document to perform native Java calls to external java interfaces. WSIF is also used internally by the JCA framework.

- Adapter services

  Oracle Application Server adapters provide bidirectional, real-time data access to virtually any data source in your enterprise.

  An adapter either listens for, or polls for, events in the source application it supports. When listening for events, an adapter registers as a listener for the application that is configured to push events to the adapter. The adapter can also poll the back-end application, such as a database or file, for the events required by Oracle Enterprise Service Bus.

  By registering adapters with Oracle Enterprise Service Bus (using a wizard), you integrate external data sources with Oracle Enterprise Service Bus, and ultimately, with each other.

Oracle Enterprise Service Bus Server enables you to define inbound and outbound adapter services. An inbound adapter service receives data from an external data source 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 target application. Oracle Enterprise Service Bus Server currently supports the Oracle adapters described in Table 1–1.

Table 1–1 Oracle Adapter Services

Adapter Service	Description
File/FTP adapter service	An inbound file/FTP adapter service reads data from a local or remote file system, transforms the file data into an XML message and sends it to Oracle Enterprise Service Bus when a new text file appears in a local or remote file system. An outbound file/FTP adapter service transforms the contents of an XML messages to a text file and writes it to a local or remote file system.
Database adapter service	An inbound database adapter service sends an XML message to Oracle Enterprise Service Bus when a SQL insert, update, or delete operation is performed against a database. An outbound database adapter transforms the contents of an XML message into a SQL insert, update, or delete operation on the target database.
JMS adapter service	An inbound JMS adapter service listens on a Java Message Service (Oracle and non- Oracle) destination and forwards incoming messages to the Oracle Enterprise Service Bus. An outbound JMS adapter service writes messages from Oracle Enterprise Service Bus to a Java Message Service external to Oracle Enterprise Service Bus.
MQ adapter service	An inbound Native MQSeries adapter service sends an XML message to Oracle Enterprise Service Bus when new XML message is received by a queue. An outbound Native MQSeries adapter service writes messages from Oracle Enterprise Service Bus to a message queue.
AQ adapter service	An inbound AQ adapter service sends an XML message to Oracle Enterprise Service Bus when a new message is received by an Oracle Advanced Queuing single or multiconsumer queue. An outbound AQ adapter service sends messages from Oracle Enterprise Service Bus to an Oracle Advanced Queuing single or multiconsumer queues.
Oracle Applications (OA) Adapter services	An inbound OA adapter sends XML messages to Oracle Enterprise Service Bus on receiving messages from an Oracle E-Business Suite interface. An outbound OA Adapter inserts data from Oracle Enterprise Service Bus into Oracle Applications using interface tables, APIs, and concurrent programs.
Custom adapter service	A custom adapter service for configuring third party adapters.

Any service, except an inbound adapter service, created in Oracle Enterprise Service Bus service, such as an outbound adapter service or routing service (described in "Content-Based and Header-Based Routing" on page 1-4), is automatically exposed as a SOAP service without requiring you to provide configuration details. Oracle ESB Control (described in "Introduction to the Oracle Enterprise Service Bus Control" on page 1-7) lists the concrete WSDL URL for these services on the Definitions tab. You

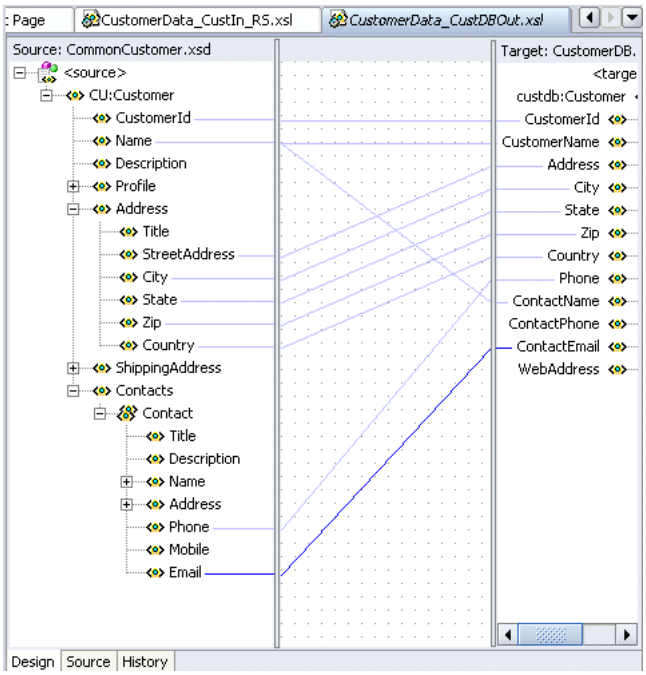
can use the concrete WSDL URL to invoke the service using SOAP over Hypertext Transfer Protocol (HTTP) from Oracle JDeveloper or Microsoft .Net. On the Definitions tab, you also specify whether a service can be invoked by an application (external) outside of Oracle Enterprise Service Bus.

Document Transformation

Oracle Enterprise Service Bus includes a standards-based data mapper launched from within Oracle JDeveloper. The data mapper specifies an `XSL` file to transform data from one XML schema to another, thus enabling data interchange among applications using different schemas. Multiple transformations may be required to achieve the desired result. These transformations can be reused, as needed, across your enterprise.

Figure 1–1 shows an example of the data mapper being used to map data from one schema to another.

Figure 1–1 Sample Data Mapper Tool



Content-Based and Header-Based Routing

Data contained within XML messages are distributed from the source application to a target application using routing services. As the name suggests, a routing service determines how a message gets from one point to another within the Oracle Enterprise Service Bus environment as defined by the routing rules it applies on the XML message. As you can define rules to route messages based on the message content, that routing is known as a content-based routing service. Routing that is based on header transformation and filtering is known as header-based routing.

Routing rules specify the set of services (referred to as target services) that Oracle Enterprise Service Bus invoke when the routing service receives a message. When you configure routing rules, you specify the following details:

- Whether a filter expression is applied.

  A filter expression specifies that the contents (payload) of a message be analyzed before any service is invoked. For example, using the scenario described in

"Sample Oracle Enterprise Service Bus Scenario" on page 1-12, you might apply a filter expression that specifies that the database adapter service be invoked only if the message includes customer contact information.

- Whether a document transformation is applied.

  See Document Transformation on page 1-4 for information about transformations.

- Whether execution is synchronous or asynchronous

  If you specify that execution is synchronous, then Oracle Enterprise Service Bus invokes the target service immediately; control is not returned to the routing service until the message has been received by the target service for processing.

  If you specify that execution is asynchronous, then Oracle Enterprise Service Bus uses JMS for delivering the message to the target service, which will be invoked at a later time. Control is returned to the routing service immediately, before the target service has received the message.

- The priority level for execution

  The priority level, determined by the order in which routing rules are ordered, determines the order in which outbound service invocation actions are executed.

For information about routing rules, see "Specifying Routing Rules" on page 5-18. For information about header transformation and filtering, see "Header Transformation and Filtering" on page 5-27.

Creating, Configuring, and Managing Oracle Enterprise Service Bus

The two main tools for creating, configuring, and managing Oracle Enterprise Service Bus are the following:

- Oracle JDeveloper

  Oracle JDeveloper is primarily intended for specifying the overall enterprise service bus creation and configuration. It enables you to:

  – Create inbound and outbound adapter services

    Oracle JDeveloper provides an adapter configuration wizard that assists you in creating inbound and outbound adapter services.

  – Create routing services from inbound adapter services

    When you complete the adapter configuration wizard for an inbound adapter service, Oracle JDeveloper gives you the opportunity to create a routing service from the newly created inbound adapter service.

  – Select the routing service that will route to an outbound service

    When you complete the adapter configuration wizard for an outbound adapter service, Oracle JDeveloper gives you the opportunity to specify the routing service that will route to the newly created outbound service.

  – Specify or create document transformation files (`XSL` files)

    As part of creating routing services from an inbound adapter service or specifying an existing routing service to an outbound adapter service, you can specify if a transformation is needed.

    If a transformation is needed, you can specify that an existing transformation file be used, or specify that you want to create a new transformation file. If you

specify that you want to create a new transformation file, Oracle JDeveloper opens the data mapper tool to enable you to do so.

- Oracle ESB Control

  Oracle ESB Control is primarily intended for run time use. It enables you to:

  - View the Oracle Enterprise Service Bus configuration graphically.

    As shown in Figure 1–3, Oracle ESB Control provides a graphical representation of the inbound and outbound adapter services, the routing services and the connections among them.

  - Adjust routing rules

    Oracle ESB Control provides property pages that enable you to specify or adjust the routing rules for routing services. For example, if you have write privileges you can specify a filter expression, or add or change the document transformation file associated with a routing operation.

  - View run-time statistics

    Oracle ESB Control enables you to view run-time statistics for messages processed by the various services.

  - Track message instances across the enterprise service bus

    Similar to the configuration diagram, Oracle ESB Control provides a graphical representation of the message flow through the services of Oracle Enterprise Service Bus.

  - You can also use Oracle ESB Control to create systems and service groups. Systems and service groups are described in "Introduction to the Oracle Enterprise Service Bus Control" on page 1-7.

The following two sections introduce these tools in more detail:

- Introduction to Oracle JDeveloper on page 1-6
- Introduction to the Oracle Enterprise Service Bus Control on page 1-7

Introduction to Oracle JDeveloper

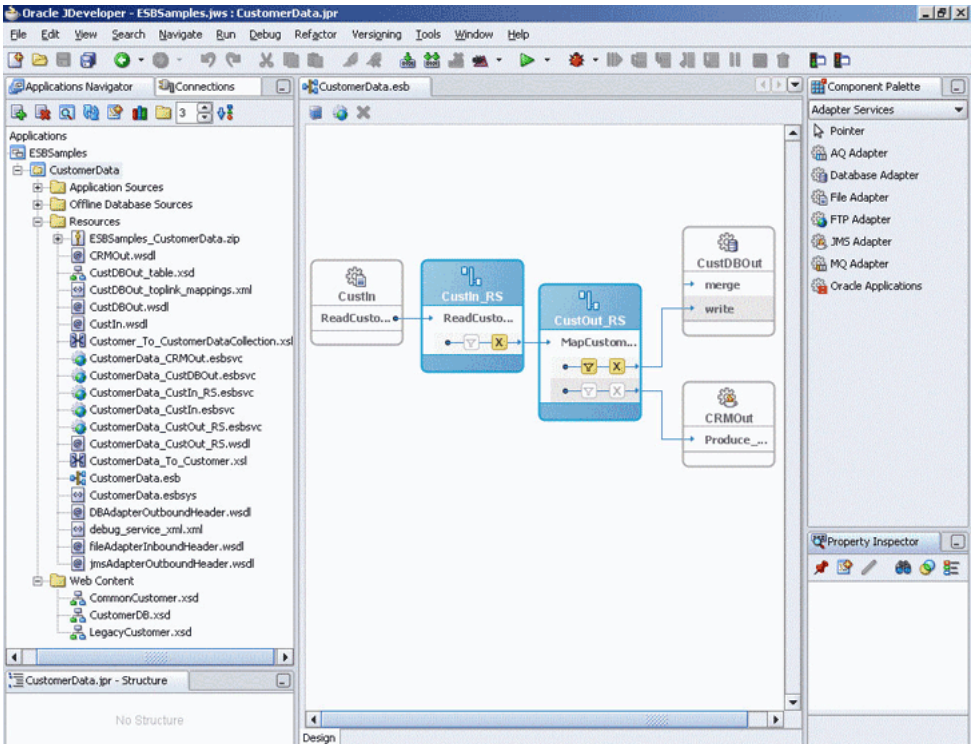
Oracle JDeveloper is an integrated development environment (IDE) for building applications and Web services using Java, XML, and SQL standards. Oracle JDeveloper supports the entire development life cycle with integrated features for designing, coding, debugging, testing, profiling, tuning, and registering applications. A visual and declarative development approach and the Oracle Application Development Framework (ADF) work together to simplify application development and reduce coding tasks.

Oracle Enterprise Service Bus includes support for the following in Oracle JDeveloper:

- Transformations and routing from inbound and outbound adapter services
- Adapters
- SOAP Services

Figure 1–2 shows Oracle JDeveloper with an ESB project.

Figure 1–2 Sample Oracle JDeveloper



The Applications Navigator displays the project files that you have created. In Figure 1–2, for example, the Applications Navigator includes an application named ESBSamples, which contains the project node entitled CustomerData. When the CustomerData node is expanded you can see the WSDL files that define the adapter services for the application, and the XSD files that define the structure of the data that will be routed across the Oracle Enterprise Service Bus.

For more information, see Chapter 2, "Developing the Enterprise Service Bus".

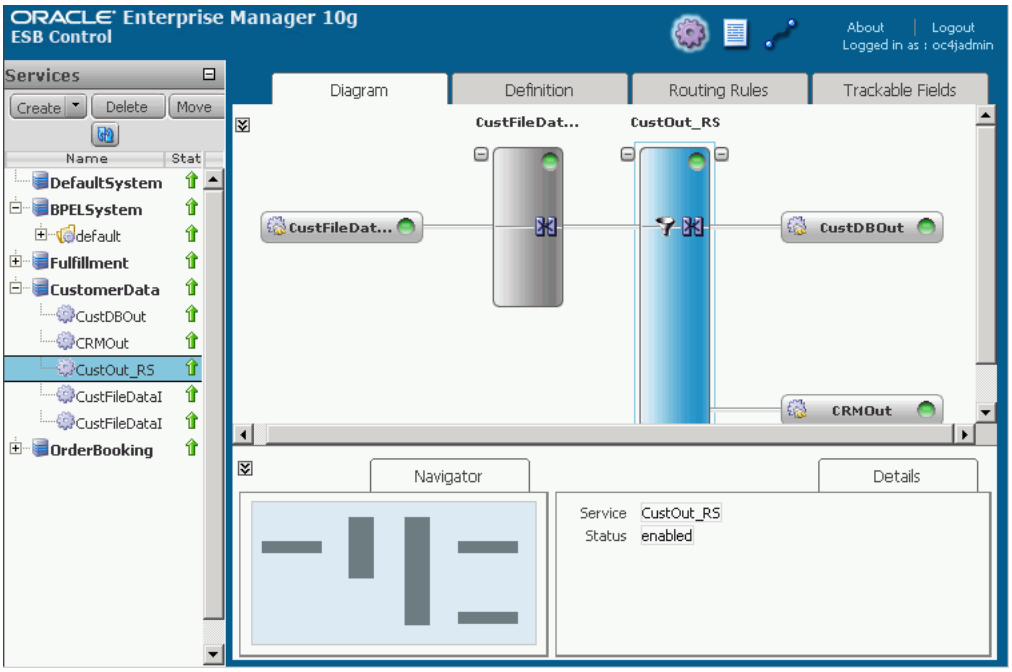
Introduction to the Oracle Enterprise Service Bus Control

You monitor and make run-time adjustments to the Oracle Enterprise Service Bus configuration using the Oracle ESB Control. The Oracle ESB Control provides the Services, Instances, and Maps views, which you select by clicking the icons that run across the top of the page.

- The Services view enables you to view service definitions, update routing rules, define trackable fields, and view a schematic diagram of the services. For more information, see "Oracle ESB Control Services View" on page 3-3.
- The Instances view enables you to view details about message processing across an ESB system. For more information, see "Oracle ESB Control Instances View" on page 3-5.
- The Maps view enables you to create, update, and delete domain-value maps, as well as view, export, and import existing domain-value maps. For more information, see "Oracle ESB Control Maps View" on page 3-6.

Figure 1–3 is an example of the Services view of Oracle ESB Control.

Figure 1–3 Oracle ESB Control - Services View



For more information, see Chapter 3, "Monitoring the Enterprise Service Bus".

Oracle Enterprise Service Bus Architecture

ESB services are designed and configured with Oracle JDeveloper and Oracle ESB Control user interfaces. The ESB project which contains the services is registered to the ESB Server. The ESB Server supports multiple protocol bindings including HTTP/SOAP, JMS, JCA, WSIF, and Java that ensure guaranteed, reliable message delivery using synchronous/asynchronous, request/reply or publish/subscribe models. However, the ESB Server does not support Remote Method Invocation (RMI).

When an ESB project is registered with the ESB server, ESB files created in Oracle JDeveloper or Oracle ESB Control are deployed to the design time metadata server. The following JMS topics are running on the design time metadata server: control, monitor, resubmit, and defer. Also running on the metadata server are the following servlets: console, WSIL, design time, WebDav, and SOAP provider.

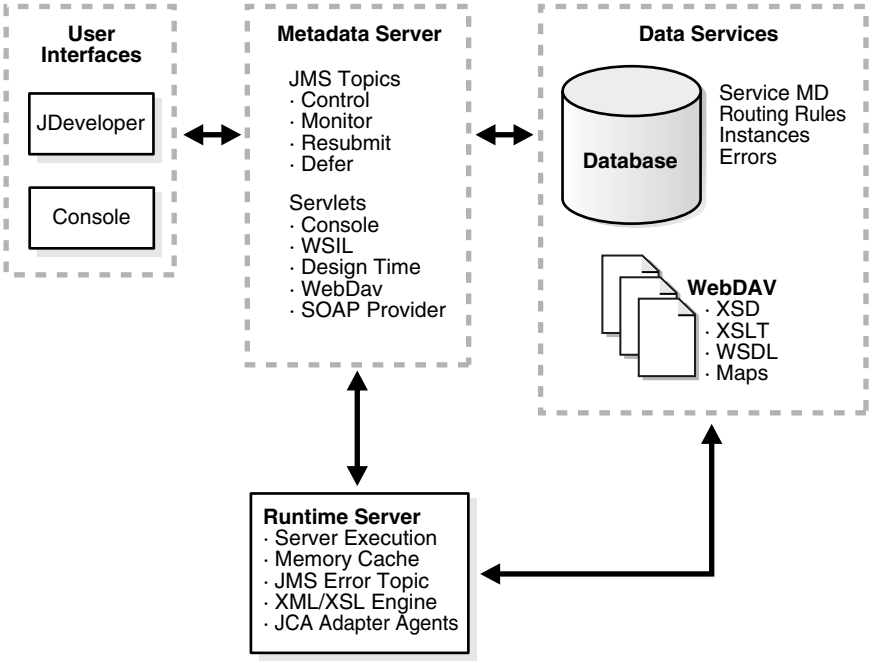
The created or updated service definition files are translated and captured in a relational form in the `ORAESB` schema in the database repository while the XSD, XSL, WSDL, and map files are written to the file system. The service definition files have pointers to the XSD, XSL, WSDL, and map files.

An ESB runtime server, or multiple servers in a cluster, accesses the control topic file on the design metadata server to cache the information for the ESB runtime services. ESB runtime servers listen on the control topic to get notified of any metadata changes. These notifications result into reload of cached metadata for the entities that changed. The ESB runtime server contains the following: server execution, memory cache, JMS error topic, XML/XSL engine, and JCA adapter agents.

At runtime, the ESB message flow is initiated by an inbound adapter polling or listening for an event, such as a file copied to the directory specified for an inbound file adapter. The ESB flow is also initiated when an external SOAP/HTTP process invokes the Web service associated with an ESB routing service.

Figure 1–4 is an illustration of ESB architecture running on a single instance. For an illustration of ESB architecture on a clustered environment, see Figure 9–1 on page 9-3.

Figure 1–4 ESB Architecture - Single Instance Environment



Instance Tracking in Oracle Enterprise Service Bus

This section discusses the following aspects of instance tracking by Oracle Enterprise Service Bus:

- Activity Types
- Activity Messages
- Database tables

Activity Types

The following list includes all the activities that Oracle Enterprise Service Bus captures during a message flow, along with their associated return status:

> **Note:** The value of the `type` attribute of the `activityMessage` element corresponds to the activity types described in this section.

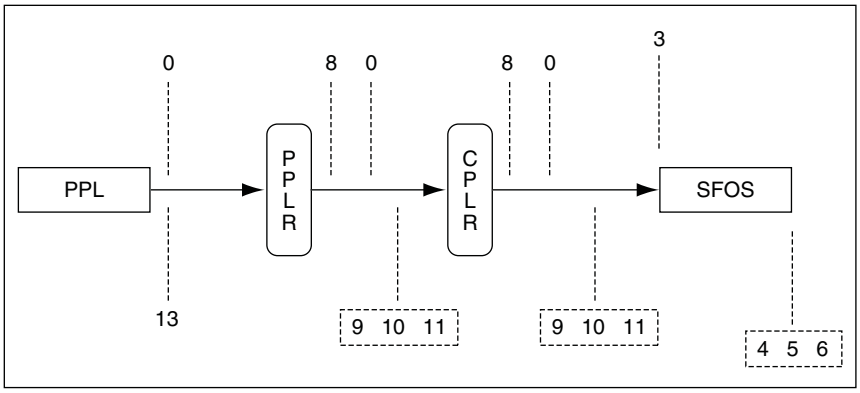
- 0 – A message has been raised
- 2 – Message has been rejected by the inbound adapter service
- 3 – An outbound adapter service operation is about to be executed
- 4 - An outbound adapter service operation has been executed successfully
- 5 - An outbound adapter service operation has generated a fault message
- 6 - An outbound adapter service operation has raised an exception

- 8 – A routing rule did not get executed due to the following reasons:
  - Not Acceptable because it was raised on the same system
  - Filtered as the filter expression was evaluated to false
  - Not executed because the target service is disabled, or the system where the routing rule is to be executed, is disabled
- 9 – A routing rule is about to be executed
- 10 – A routing rule has been executed successfully

  This activity also ensures that if a response or fault was generated then the following activities were performed:
  - The response or fault has been sent to the caller
  - The response or fault has been sent to the response handler or the fault handler
  - The response has been ignored
- 11 – A routing rule has generated a fault message
- 12 – A routing rule has raised an exception
- 13 – A retryable routing rule has raised an exception
- 14 – The message raised has failed
- 15 – The global transaction, in which the message was processed, has been committed
- 16 – The global transaction has been rolled back due to an error

The Oracle Enterprise Service Bus server calls the callback APIs provided for each activity type. Figure 1–5 shows the points at which the activities are captured:

Figure 1–5 Instance Tracking in ESB



Activity Messages

All the activity messages are wrapped in an element called `activityMessages` and is published to ESB Monitor topic. The following list includes all the activity messages along with description:

- Message raised

  Indicates that an event has been raised in Oracle Enterprise Service Bus
- Message raise failed

Indicates that the event raised has thrown an exception

- Outbound operation is about to be executed

  Indicates that an outbound adapter service operation is going to be executed

- Outbound operation has been executed successfully

  Indicates that an outbound adapter service operation execution was successful

- Outbound operation has generated a fault message

  Indicates that an outbound adapter service operation has generated a WSDL fault message

- Output operation has raised an exception

  Indicates that the operation has thrown an exception back to the caller

- No routing rule to execute

  Indicates that there were no routing rules defined for a routing service

- Routing rule is about to be executed

  Indicates the a routing rule is going to be executed

- Routing rule has been executed successfully

  Indicates that the routing rule execution was successful

- Routing rule has generated a fault message

  Indicates that the execution of the routing rule has produced a fault message

- Routing rule has raised an exception

  Indicates that the execution of the routing rule was not successful and generated an exception

- Routing rule has not been executed

  Indicates that the routing rule was not executed for some reason

- Transaction committed

  Indicates that the transaction in which the message was processed has been committed

- Transaction committed

  Indicates that the transaction in which the message was processed has been rolled back

Database tables

Oracle Enterprise Service Bus uses the following tables for instance tracking:

- `ESB_ACTIVITY`
- `ESB_TRACKING_FIELD_VALUE`
- `ESB_FAULTED_INSTANCE`
- `ESB_TRANSACTION_STATUS`

See Also: Chapter 10, "Tracking Message Instances Across the Enterprise Service Bus"

Sample Oracle Enterprise Service Bus Scenario

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 using Oracle Enterprise Service Bus to deliver appropriate real-time data access to all applications that update or have a common interest in the same data.

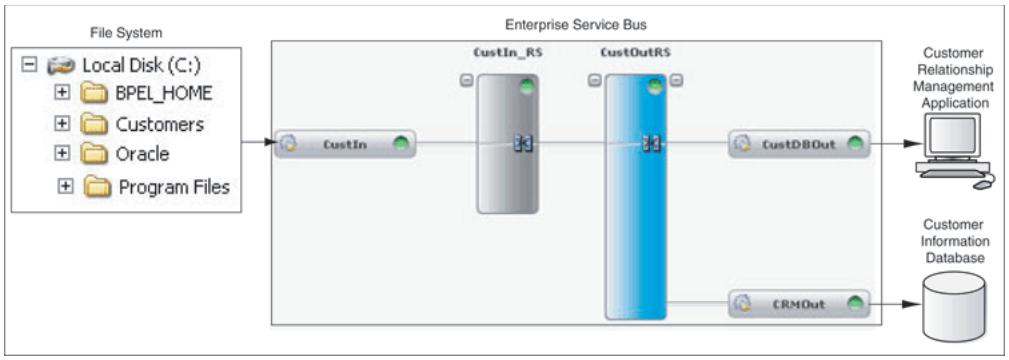
For example, Oracle Enterprise Service Bus might accept data contained in a text file, transform it to a format appropriate for updating a database that serves as a customer repository, and route and deliver the data to that database.

To accomplish all the required tasks, Oracle Enterprise Service Bus follows these basic steps, which are shown in Figure 1–6. These steps are simplified for the purposes of the introductory example.

Figure 1–6 illustrates a scenario in which Oracle Enterprise Service Bus:

1. Receives the customer data from a file system as a text file, through an inbound file adapter service named `CustIn`. The `CustIn` adapter service sends the message to a routing service named `CustIn_RS`.
2. The `CustIn_RS` routing service transforms the data format from the file adapter's schema into the canonical XML schema, and sends the message to the routing service named `CustOut_RS`.
3. The `CustOut_RS` routing service:
   - Routes the message in the canonical format to the `CRMOut` file adapter service.
   - Applies a filter expression to the XML message payload to determine whether the message should be routed to the outbound adapter service for the Customer Information Database, `CustDBOut`.
   - Invokes the appropriate adapter services (as determined by the filter expression). Routing rules specify that messages bound for the `CustDBOut` service be sent synchronously, while those bound for the CRMOut service be sent asynchronously.
     - If the receiving adapter service is `CustDBOut`, then the `CustDBOut` service is invoked immediately and control is not returned to the `CustOut_RS` service until the message has been received by `CustDBOut`.
     - If the receiving adapter service is `CRMOut`, then the message is sent to JMS and control is immediately returned to the `CustOut_RS` service.
4. The outbound adapter service delivers the message to its associated external application.

Figure 1–6 Illustration of a Sample ESB Scenario



See Also: *Oracle Enterprise Service Bus Quick Start Guide* for step-by-step instructions on implementing this scenario

Starting, Stopping, and Accessing Oracle Enterprise Service Bus Components

The key Oracle Enterprise Service Bus components are the ESB Server, Oracle JDeveloper, and Oracle ESB Control.

This section contains the following topics:

- Starting and Stopping the ESB Server on page 1-13
- Opening Oracle JDeveloper on page 1-14
- Opening the Oracle ESB Control on page 1-15

For information about the system requirements for and the installation of Oracle Enterprise Service Bus, see *Oracle Enterprise Service Bus Installation Guide*.

For information about the system requirements for and the installation of Oracle Application Server 10g release 3 (10.1.3.1.0) which provides the Oracle SOA Suite, see the Oracle Application Server 10g release 3 (10.1.3.1.0) installation guide for your system.

For information about getting started with the Oracle SOA Suite components of Oracle Application Server 10*g* release 3 (10.1.3.1.0), see *Oracle Application Server Administrator's Guide*.

Note: The Oracle Application Server SOA Suite Basic Install type is recommended for developers. For information about the Oracle SOA Suite installation, see the Oracle Application Server 10*g* release 3 (10.1.3.1.0) installation guide for your system. For information about stopping and starting Oracle SOA Suite components of Oracle Application Server, see *Oracle Application Server Administrator's Guide*.

Starting and Stopping the ESB Server

This section discusses starting and stopping the ESB Server for the Oracle Enterprise Service Bus standalone Developers installation type. For information about the Oracle Enterprise Service Bus installation types, see *Oracle Enterprise Service Bus Installation Guide*.

The process of starting and stopping the ESB Server depends on the operating system where the ESB server is located.

Windows Installation

To start the ESB Server on Windows, use one of the following methods:

- From the desktop **Start** button, select **Programs** > **Oracle – *Oracle_Home*** > **Oracle ESB** > **Start ESB Server,** where *Oracle_Home* is the name of the Oracle home where you installed Oracle Enterprise Service Bus.
- From a command window, run *Oracle_Home*/opnm/bin/opmnctl startall where *Oracle_Home* is the name of the Oracle home where you installed Oracle Enterprise Service Bus

To stop the ESB Server on Windows, use one of the following methods:

- From the desktop **Start** button, select **Programs** > **Oracle – *Oracle_Home*** > **Oracle ESB** > **Stop ESB Server,** where *Oracle_Home* is the name of the Oracle home where you installed Oracle Enterprise Service Bus.
- From a command window, run *Oracle_Home*/opnm/bin/opmnctl stopall where *Oracle_Home* is the name of the Oracle home where you installed Oracle Enterprise Service Bus

Linux Installation

To start the ESB Server on Linux:

1. Start a command prompt and navigate to the *Oracle_Home*/opmn/bin folder.
2. Run the following from the operating system prompt:

   ```
   opmnctl startall
   ```

To stop the ESB Server on Linux:

1. Start a command prompt and navigate to the *Oracle_Home*/opmn/bin folder.
2. Run the following from the operating system prompt:

   ```
   opmnctl stopall
   ```

To check the status of the ESB Server on Linux:

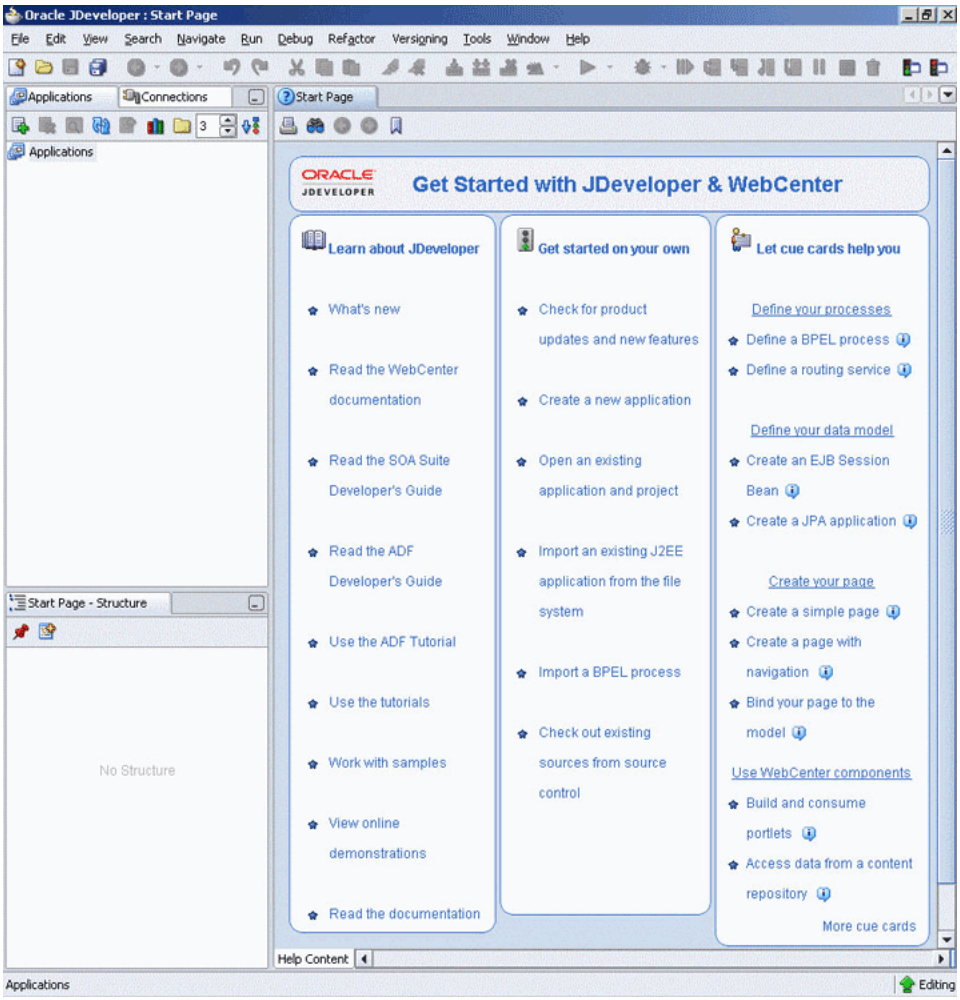
1. Start a command prompt and navigate to the *Oracle_Home*/opmn/bin folder.
2. Run the following from the operating system prompt:

   ```
   opmnctl status
   ```

Opening Oracle JDeveloper

To launch Oracle JDeveloper, run the jdeveloper executable in the JDeveloper home directory. The Oracle JDeveloper is started, as shown in Figure 1–7.

Figure 1–7 Oracle JDeveloper – Initial Display



To close Oracle JDeveloper, click **Exit** in the **File** menu.

Opening the Oracle ESB Control

The Oracle ESB Control is launched from a Web browser. The Oracle ESB Control can be accessed with the following URL:

`http://host_name:port_number/esb`

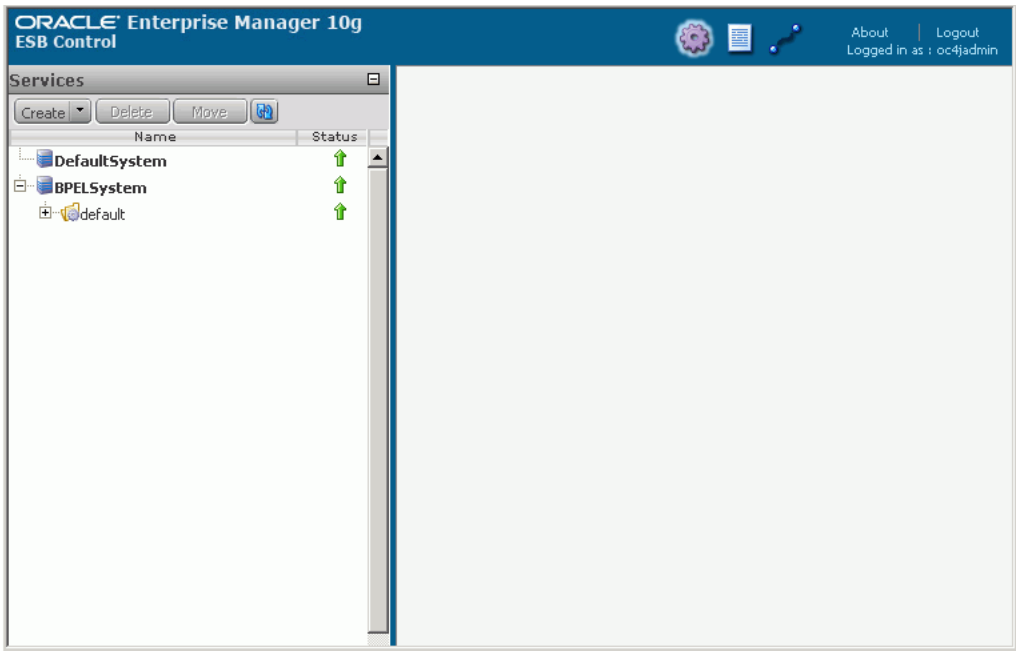
In the URL example, *`host_name`* is the host name where the ESB Server is running and *`port_number`* is the Oracle HTTP Server Listen port number, which is listed in the output from the `opmnctl status -l` command. For more information about viewing port numbers, see "Viewing Port Numbers" on page 2-6.

On Windows, you can launch the Oracle ESB Control from the Start menu. Select **All Programs** > **Oracle – *`Oracle_Home`*** > **Oracle ESB** > **ESB Control** from the desktop **Start** button, where *Oracle_Home* is the name of the Oracle home where you installed Oracle Enterprise Service Bus.

Before opening the Oracle ESB Control, you must first start the ESB Server as described in "Starting and Stopping the ESB Server" on page 1-13.

The Oracle ESB Control opens, as shown in Figure 1–8.

Figure 1–8 Oracle ESB Control - Initial Display



In Figure 1–8, The BPEL System is default location where BPEL services display up if they are on the same SOA Suite container. The `default` service group matches the BPEL Domain name for that BPEL process.

2

Developing the Enterprise Service Bus

The primary interface for developing an enterprise service bus is Oracle JDeveloper. This chapter provides an introduction to Oracle JDeveloper, an overview of the design process, and detailed descriptions of how to accomplish the tasks required to design an Oracle Enterprise Service Bus.

This chapter contains the following topics:

- Overview of Oracle JDeveloper on page 2-1
- Getting Started with Oracle JDeveloper on page 2-5

Overview of Oracle JDeveloper

Oracle JDeveloper is an integrated development environment that provides tools to help you design the enterprise service bus. It includes wizards and a design panel that enable you to view the services that you create visually.

When you open Oracle JDeveloper, the interface appears as shown in Figure 1–7, "Oracle JDeveloper – Initial Display" on page 1-15.

The following sections introduce the key components of Oracle JDeveloper:

- Overview of Connection Navigator Tab on page 2-1
- Overview of the Application Navigator Tab on page 2-1
- Overview of the Design Tab and Component Palette on page 2-2
- Overview of Service Definitions and Routing Rules on page 2-4

Overview of Connection Navigator Tab

The Connection Navigator tab, which appears in the upper left corner of Oracle JDeveloper, enables you to create connections from Oracle JDeveloper to various servers, such as an application or integration server. You can also create a connection to a database. If the Connection Navigator is not visible in your console, from the **View** menu, select **Connection Navigator**.

See "Creating and Testing Connections" on page 2-6.

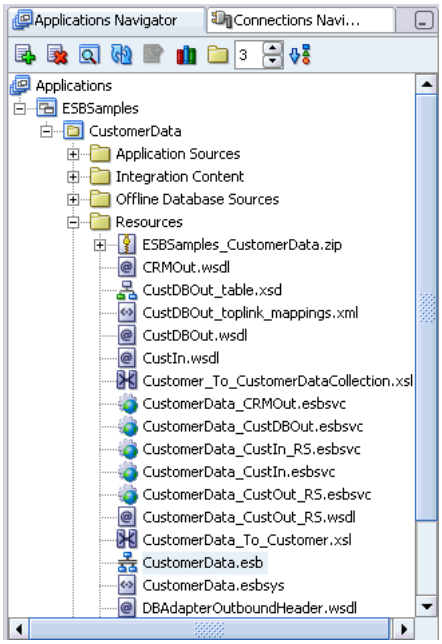
Overview of the Application Navigator Tab

The Application Navigator tab, which appears in the upper left corner of Oracle JDeveloper, is the starting point of the design process. If the Applications Navigator is not visible in your console, from the **View** menu, select **Application Navigator**.

You use the Application Navigator tab to create an application workspace and an ESB project. Within an ESB project you can create multiple ESB systems. As you build the enterprise service bus, the files you create for the services, transformation maps, and so on, are presented in the Applications Navigator within the project folder.

A sample Application Navigator tab is shown in Figure 2–1. In this sample, an application workspace named ESBSamples contains an ESB project named CustomerData. See "Creating Applications and ESB Projects" on page 2-11 for step-by-step instructions.

Figure 2–1 Sample Application Navigator



When you right-click a node in the Application Navigator tab, a menu of commands is displayed. The menu commands displayed depend on the node selected. For example, when you right-click an ESB project, such as CustomerData in Figure 2–1, the following commands are included in the menu:

- New

  This command opens the New Gallery dialog box, which provides access to dialog boxes and wizards that enable you to create new projects, create inbound and outbound adapter services, and access the document transformation tool.

- Register with ESB

  This command enables you to register the adapter services, transformations, and routing services that you design using Oracle JDeveloper with Oracle Enterprise Service Bus. When you register the ESB project with the ESB Server, as described in "Registering ESB Projects and Services with the ESB Server" on page 2-15, the files are written to the ESB repository and are available from the Oracle ESB Control.

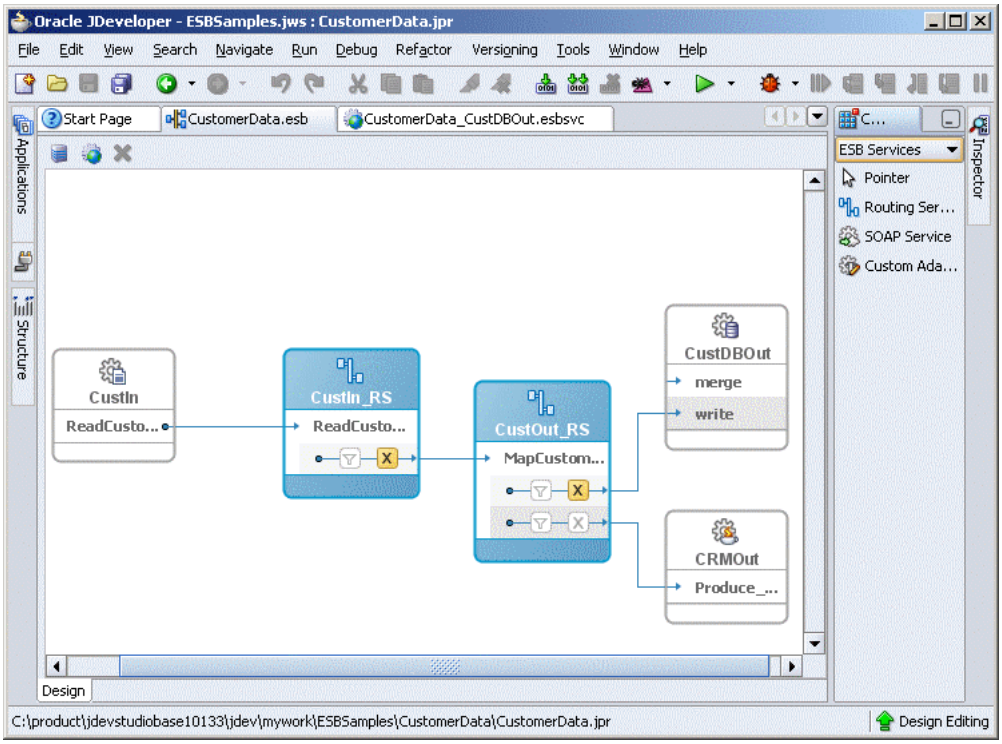
Overview of the Design Tab and Component Palette

When you double-click an ESB project in the Applications Navigator, the project properties dialog displays. Click the diagram component (esb file) in the Resources folder within the project to display the Design tab and the Component Palette. If you

click the down arrow under Component Palette and select ESB Services, the types of ESB services from which you can choose to build your enterprise service bus are displayed, as shown in Figure 2–2. Similarly, if you click the down arrow under the Component Palette and select Adapter Services, you can choose the type of adapter service you want to create to move messages into and out of the enterprise service bus.

The Design tab initially appears in the empty area in the middle of the figure; the Components Palette appears to its right. In Figure 2–2, the CustomerData project is selected in the Applications view and the Design tab displays the adapter and routing services in the project.

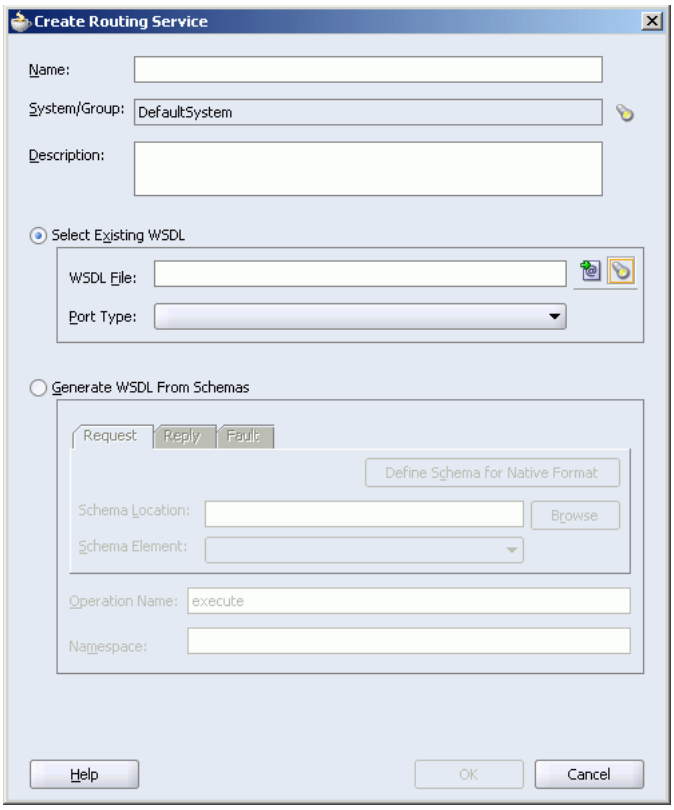
Figure 2–2 Oracle JDeveloper Design Tab and Component Palette



To create an ESB service, you drag a desired ESB service type from the Components palette and drop it onto the Design tab. When you do so, a wizard or dialog box opens to assist you in creating the desired service. Alternatively, you can create ESB services and Adapter Services using dialog boxes only, but the Design tab provides a visual method for building the enterprise service bus.

For example, you can drag and drop a Routing Service from the Component Palette to the Design tab. When you do so, the Create Routing Service dialog box opens, as shown in Figure 2–3. When you complete this dialog box and click OK, an icon appears on the Design tab, similar to the CustOut_RS in Figure 2–2.

Figure 2–3 Create Routing Service Dialog Box



You can use this building block approach to add the component services to the enterprise service bus design one-by-one. Figure 2–2, for example, shows a completed simple ESB configuration.

See Chapter 4, "Creating Inbound and Outbound Services" for conceptual information and step-by-step instructions on creating adapter services. See Chapter 5, "Creating Routing Services and Routing Rules" detailed conceptual information and step-by-step instructions on creating routing services and specifying routing rules.

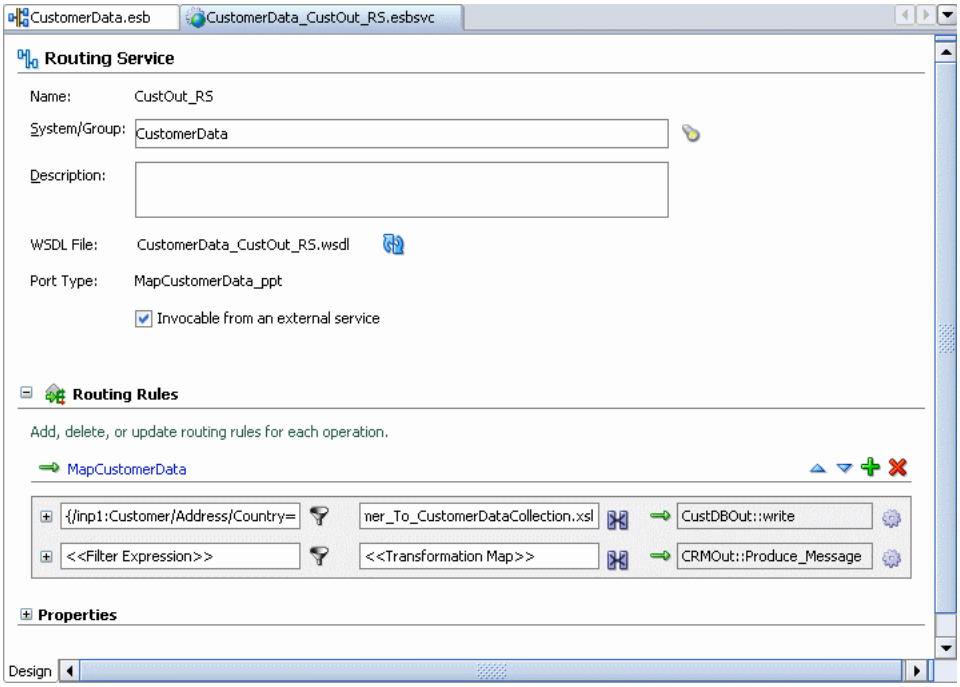
Overview of Service Definitions and Routing Rules

After you have created some services, you can view the definition of, and define properties for, each service. You can also view routing rules for a routing service.

For example, suppose you create the routing service whose icon is in the Design tab shown in Figure 2–2. When you double-click the top of the icon, property sheets replace the visual presentation of the enterprise service bus configuration. The Routing Rules panel is only displayed when you are viewing the property sheets for a routing service.

As shown in Figure 2–4, the Definition section presents the service name, ESB system (and group, if applicable), a description of the service, the name of the WSDL file that defines the service, the port type and an indication of whether the service can be invoked from a service that is external to the ESB system in which the service was created.

Figure 2–4 Sample Property Sheet for a Routing Service



The Routing Rules section, shown in Figure 2–4, presents icons that provide access to the tools that enable you to specify a filter expression, a transformation map, and the target operation (defined within the target service WSDL). It also enables you to limit the ESB systems from which messages will be accepted and whether execution of the target operation is to be synchronous or asynchronous.

See Chapter 5, "Creating Routing Services and Routing Rules" for step-by-step instructions about creating routing services.

Getting Started with Oracle JDeveloper

The overall process and tasks required for designing the enterprise service bus are outlined in the following list. Links to step-by-step instructions are specified for each task:

1. Start the ESB Server and open Oracle JDeveloper

   The key Oracle Enterprise Service Bus design components are the ESB Server, Oracle JDeveloper, and the database that serves as the ESB Metadata Server. For information about starting ESB components, see "Starting, Stopping, and Accessing Oracle Enterprise Service Bus Components" on page 1-13.

2. Create the necessary server and database connections.

   See "Creating and Testing Connections" on page 2-6.

3. Create the Oracle JDeveloper applications and projects

   See "Creating Applications and ESB Projects" on page 2-11.

4. Create ESB systems and, optionally, service groups.

   See "Creating ESB Systems and Service Groups" on page 2-12

5. Add or import into an ESB project.

See "Adding Project Content" on page 2-14 and "Importing Files into a Project" on page 2-14.

6. Create the inbound and outbound services, or invoke external services.

   See Chapter 4, "Creating Inbound and Outbound Services".

7. Create routing services and configure the routing rules to:

   - Apply conditional expressions to the message instance payload
   - Transform the structure of the message payload
   - Specify where a response in a request/response scenario be sent
   - Specify where faulted messages be sent

   See Chapter 5, "Creating Routing Services and Routing Rules".

8. Register services with the ESB repository.

   See "Registering ESB Projects and Services with the ESB Server" on page 2-15.

9. Sync services with the ESB Server.

   See "Syncing Services From ESB Server" on page 2-15.

If you no longer need a service, you can remove it. See "Deleting ESB Projects" on page 2-16.

Creating and Testing Connections

To begin configuring an Oracle Enterprise Service Bus, you need to create the necessary server and database connections.

This section discusses the following topics:

- Viewing Port Numbers on page 2-6
- Creating an Application Server Connection on page 2-7
- Creating an Integration Server Connection on page 2-8
- Creating a Database Connection on page 2-8
- Testing a Connection on page 2-10

Viewing Port Numbers

To view port numbers that are used when setting up connections, you can use the following:

- On Windows, click the desktop **Start** button, select **All Programs** > **Oracle – *`Oracle_Home`*** > **Oracle Process Manager** > **Oracle Assigned Port Numbers**, where *`Oracle_Home`* is the name of the Oracle home where you installed Oracle Enterprise Service Bus.
- Run *`Oracle_Home`*`/opmn/bin/opmnctl.exe status -l`, where *`Oracle_Home`* is the name of the Oracle home where you installed Oracle Enterprise Service Bus.
- In Oracle Enterprise Manager Application Server Control, open the **Cluster Topology** > **Runtime Ports** page.
- View the port numbers in the *`Oracle_Home`*`/opmn/config/opmn.xml` file.

Creating an Application Server Connection

To create a new application server connection, follow these steps:

1. Select Connection Navigator under the View menu to display the Connections Navigator.
2. Right-click Application Server in the tree and select New Application Server Connection to launch the Create Application Server Connection wizard. The information you need to provide varies depending on the type of connection you choose on the Type page. Follow the instructions in the wizard to complete each of the following pages:
   a. Welcome page
   b. Type page

      Connection Name: Enter a name, such as `LocalApplicationServer`

      Connection Type: Select a type, such as `Oracle Application Server 10g 10.13`
   c. Authentication page

      Username: Enter the OC4J username

      Password: Enter the password

      Deploy Password: Specify whether to deploy the password
   d. Connection page

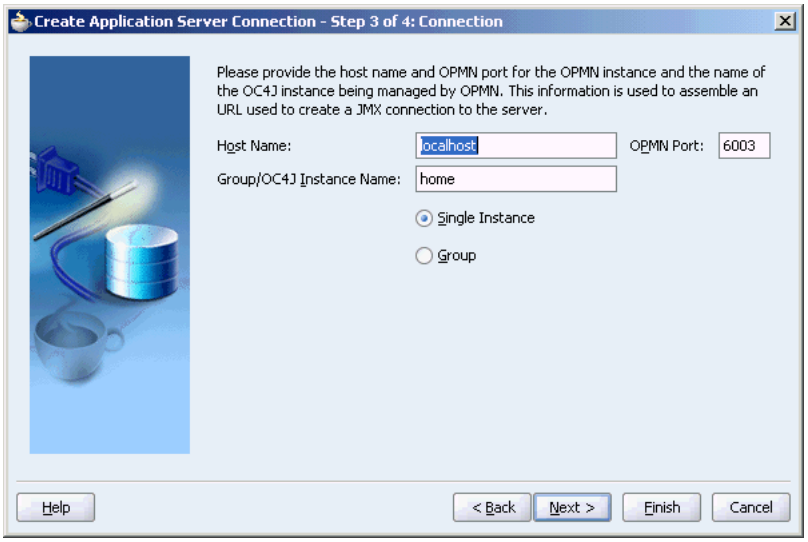
      Enter the information for the specific connection type. For example, these are options for the `Oracle Application Server 10g 10.13` type:

      Host Name: Enter the host name or `localhost`

      OPMN Port: Enter the port number, such as `6003`

      OC4J Instance Name: Enter the OC4J instance name

      Connect To: Select `Single Instance` or `Group`



   e. Test page
3. On the Test page, click the Test Connection button to test the information you provided. If the test succeeds, click the Finish button. If the test fails, review the

information for the connection and correct as necessary. Before testing, make sure the ESB Server is running.

Creating an Integration Server Connection

To create a new integration server connection, follow these steps:

1. Select Connection Navigator under the View menu to display the Connections Navigator.
2. Right-click Integration Server tree and select New Integration Server Connection to launch the Create Integration Server Connection wizard. Follow the instructions in the wizard to complete each of the following pages:
   a. Welcome page
   b. Name page

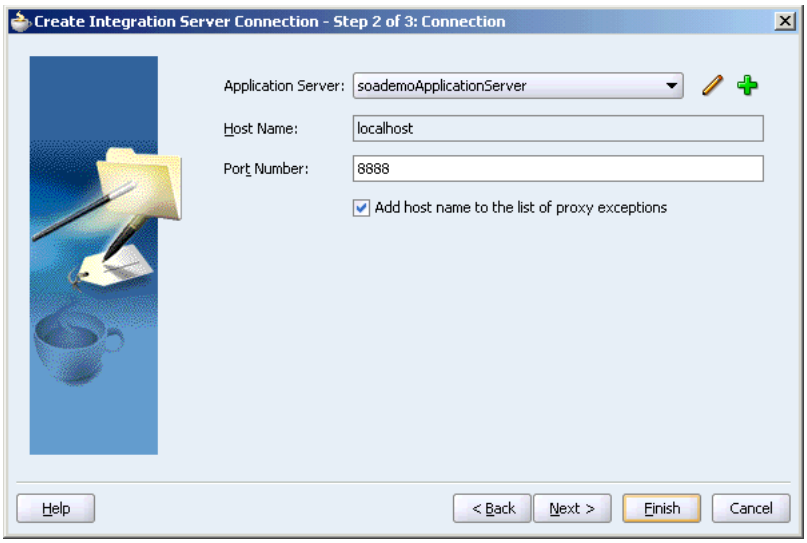
      Connection Name: Enter a name, such as `LocalIntegrationServer`
   c. Connection page

      Application Server: Select an application server that has been previously set up from the list

      Host Name: Enter the host name or `localhost`

      Port Number: Enter the port number, such as `8888`

      Add Host Name: Specify whether you want to add the host name to the proxy exceptions



   d. Test page
3. On the Test page, click the Test Connection button to test the information you provided. If the test succeeds, click the Finish button. If the test fails, review the information for the connection and correct as necessary. Before testing, make sure the ESB Server is running.

Creating a Database Connection

To create a new database server connection, follow these steps:

1. Select Connection Navigator under the View menu to display the Connections Navigator.

2. Right-click Database tree and select New Database Connection to launch the Create Database Connection wizard.

Note: If you have selected the Basic Installation on Windows, an Oracle Lite database has been installed with a Windows Service named `Oracle Lite Multiuser Service`.

Follow the instructions in the wizard to complete each of the following pages. The Oracle Lite database is used an example in the steps.

a. Welcome page

b. Type page

Connection Name: Enter a name for the database connection, such as `Olite`. For the Oracle Lite database, it is strongly recommended to use `Olite` to matched the predefined JNDI name (`eis/DB/Olite`) and in the name in connector XML file.

Connection Type: Select a connection type from the list, such as `Oracle Lite`

c. Authentication

Username: Enter the username for the database account

Password: Enter the password for the username

Role: Enter the database role for the user

Deploy Password: Specify whether to deploy the password

d. Connection page

Provide the connections details for the type of database connection selected on the Type page. For example, if Oracle Lite database type is selected you would enter data for the fields described in this step.

Driver type and connection details for an Oracle Lite database:

Driver: Select the driver, such as `Type 4 Driver`

Host Name: Enter the host name where the database is installed

JDBC Port: Enter the port number for database, such as `1531`

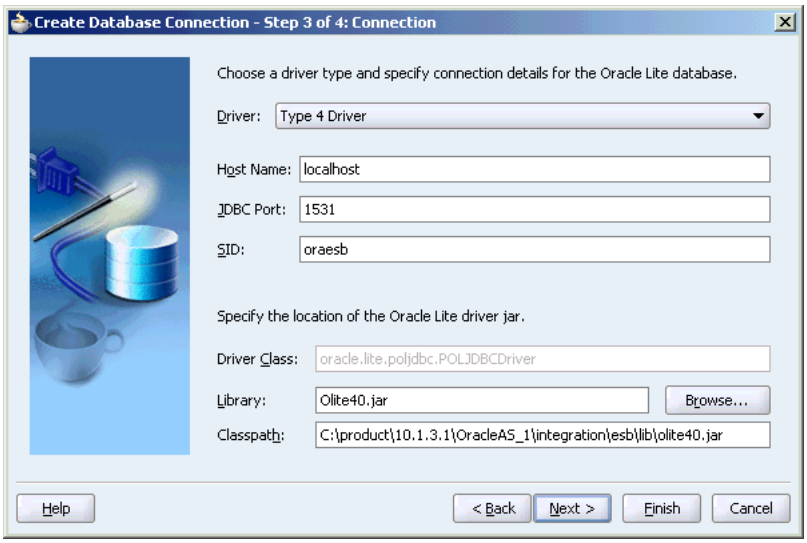
SID: Enter the service ID of the database, such as `oraesb`

Location of the Oracle Lite driver jar:

Driver Class: Enter the class name or accept the default class name, such as `oracle.lite.poljdbc.POLJDBCDriver`

Library: Enter the library name, such as `Olite40.jar`

Classpath: Enter the classpath or accept the default classpath, such as `c:\product\10.1.3.1\OraceleAS_1\integration\esb\lib\olite40.jar`



e. Test page

3. On the Test page, click the Test Connection button to test the information you provided. If the test succeeds, click the Finish button. If the test fails, review the information for the connection and correct as necessary. Before testing, make sure the database is running.

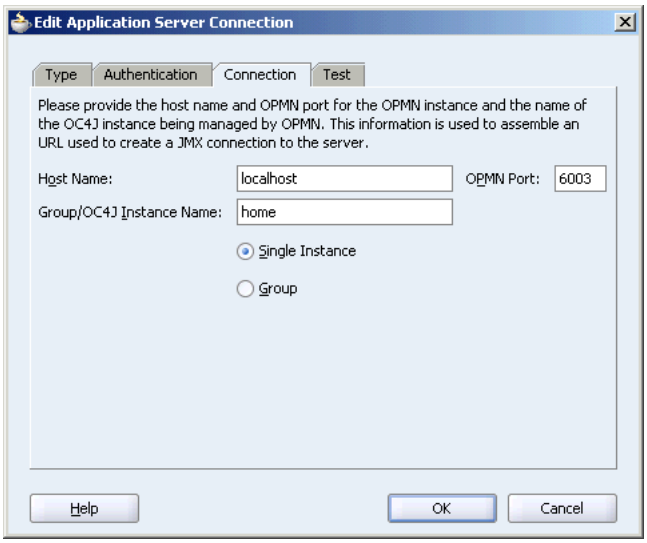
Testing a Connection

To review and test the connections in Oracle JDeveloper, perform the following steps:

1. Select Connection Navigator under the View menu to display the Connections Navigator.
2. In the Connections Navigator, expand the connection tree that you want to test.

   For example, expand the Application Server tree.
3. Right-click a connection in the tree that has been already been set up and select Properties to display the Edit Connection property sheet. You can also double-click the connection to display the property sheet.

   For example, right-click an application server under the expanded Application Server tree and select Properties. Click the Connection tab to view the connection properties.



4. Review the information in the property tabs for the connection.
5. In the Test tab, click Test Connection to test the connection.
6. If the test fails, then correct the information in the property tabs and retest. Make sure that the necessary server is running before testing the connection.

Creating Applications and ESB Projects

To begin configuring an Oracle Enterprise Service Bus, you must create an application and an ESB project. An application contains one or more projects. An ESB *project* is a collection of related files, for example, all the Oracle Enterprise Service Bus services and associated files within a single Oracle Enterprise Service Bus.

To create an application in Oracle JDeveloper follow these steps:

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

   The New Gallery dialog box opens.
2. In the **Categories** panel, select **General.**
3. In the **Items** panel, double-click **Application.**

   The Create Application dialog box opens.
4. In the **Application Name** field, enter a name for the application.
5. In the **Directory Name** field, accept the default value or enter the directory specification for the directory on the local file system where you want to store the project files.
6. Click OK.

   The Create Project dialog box opens.
7. In the Create Project dialog box, click **Cancel.** You will use a different dialog box to create an ESB project.

To create an ESB project, follow these steps after you have created an application:

1. In the **Applications Navigator,** right-click the application to which you want to add a project and then click **New Project.**

   The New Gallery dialog box opens.

2. Select **ESB Project** under **General Projects**, and then click **OK.**

   The Create ESB Project dialog box opens.

3. In the **Project Name** field, enter the name of the project.
4. In the **Directory Name** field, specify the directory where the project files will be stored, if you do not want to accept the default directory.
5. In the **Diagram Name** field, optionally enter a name for the diagram that will schematically represent the project, if you do not want to accept the default diagram name.
6. Click **OK.**

   A design area is presented with a label of *`esb_project-name`*`.esb`, or *`diagram-name`*`.esb` if you specified an optional name for the diagram in the previous step.

You use this design area to model the enterprise service bus system. It enables you to drag and drop ESB Services (adapter, routing, and SOAP services) from the Component Palette panel in the upper-right corner into the design area.

Creating ESB Systems and Service Groups

ESB systems and service groups are units for organizing the services for the Oracle Enterprise Service Bus.

The following sections contain information about and step-by-step instructions on creating these organizational units:

- Creating ESB Systems on page 2-12
- Creating ESB Service Groups on page 2-13

Creating ESB Systems

A system is a representation of a single application, proxy for an application, or a technical system. Examples of systems are:

- An Oracle Applications instance
- A set of transformations, Oracle BPEL Process Manager services, and adapter services for an SAP instance
- A set of transformations, BPEL Services, and database services for a custom database application
- A standalone Oracle Business Activity Monitoring instance, complex event processing (CEP) services, and other related services
- An Oracle B2B engine (that serves as a proxy for trading partners) and related transformation services and other services
- A set of services, adapter services, and Oracle BPEL Process Manager services representing a bridge to a third party integration infrastructure (such as webMethods, Inc, and IBM infrastructures)

If you do not explicitly specify an ESB system, then the services created in the ESB project are added to the default system. The default system is provided for getting started quickly.

Follow these steps to create an ESB system using Oracle JDeveloper:

1. In the **Applications Navigator,** open the ESB Project in which you want to create the ESB system.
2. In the **Design** tab, click the **Create System/Group** icon on the top of the Design window. This is the icon on left in the group of three icons. See Figure 2–2 on page 2-3 for a screenshot that shows the icons at the top of the Design tab.

   The Create ESB System or Service Group dialog box opens.
3. In the **Create ESB System or Service Group** dialog box, follow these steps:
   a. For the **Create** option, select **System**.
   b. In the **Name** field, enter a unique name.

      The name must be unique across the ESB project in which you are creating it.
   c. Optionally, in the **Description** field, enter a description of the ESB service.
   d. Click **OK.**
4. In the **Applications Navigator,** click the refresh icon.

   An entry for the ESB system in the form, *system-name*.esbsys, is added to the Resources folder under the project name. For example, Figure 2–1 on page 2-2 shows a Oracle JDeveloper application that contains an ESB project named CustomerData and an ESB system also named CustomerData.

For information about creating an ESB system from the Oracle ESB Control, see "Managing Oracle Enterprise Service Bus Systems and Service Groups" on page 3-8.

Creating ESB Service Groups

A service group is an organization unit for ESB services. Its purpose is much like a directory in a file structure – it is a mean to organize ESB services in groups.

A service group is similar to a directory in a directory structure. Just as a file can belong to only one directory, a service can belong to only one service group. Different services, with the same name can belong to different service groups, just as different files with the same file name can reside in different directories in a file system. In addition, you can create one service group within another, resulting in nested service groups.

Before you can create an ESB service group, you must create an ESB system to contain it, as described in "Creating ESB Systems" on page 2-12, then follow these steps to create the ESB service group:

1. In the **Applications Navigator,** open the ESB Project in which you want to create the ESB system.
2. In the **Design** tab, click the **Create System/Group** icon on the top of the Design window. This is the icon on the left in the group of three icons. See Figure 2–2 on page 2-3 for a screenshot that shows the icons at the top of the Design tab.

   The Create ESB System or Service Group dialog box opens.
3. In the Create ESB System or Service Group dialog box, follow these steps:
   a. For the **Create** option, select **Service Group**.
   b. In the **Name** field, enter a unique name.

      The name must be unique across the ESB system in which you are creating it.
   c. In the **System/Group** field, enter the ESB system or an existing service group in which you want to create this service group.

   d. Optionally, in the **Description** field, enter a description of the ESB service group.
   e. Click **OK.**

4. In the **Applications Navigator,** click the refresh icon.

   An entry for the ESB service group in the form, *service-group-name*.esbgrp, is added to the Resources folder under the project name.

Adding Project Content

You can add project content to an add existing project.

To add content to a project in Oracle JDeveloper, follow these steps:

1. In the **Applications Navigator,** right-click the ESB Project to which you want to add project content.
2. Click **Add to Project Content** in the menu.

   The Project Properties dialog opens.
3. Expand Project Content in the tree to view the categories of content that can be added. These include:
   - Integration Content
   - Modelers
   - Offline Database
   - Resources
   - TopLink
   - Web Application
4. Make your selections for Project Content additions or for additions in the categories under Project Content.
5. Click **OK**.

Note: If you add an XSD file to the project with the **Java Content** feature, the XSD file is not copied to the project directory if it is not already present in the project directory. Although the file shows up in the project tree, it does not get deployed to the ESB Server with the **Register with ESB** menu option.

Importing Files into a Project

You can import files into an existing project.

To import files into a project in Oracle JDeveloper, follow these steps:

1. In the **Applications Navigator,** select the ESB Project to which you want to import files.
2. Select **Import** from the **File** menu.

   The Import dialog opens.

3. Select the type of file you want to import from the list in the dialog, then click OK to launch the wizard for the specific file type. Follow the instructions provided by each wizard.

   For example, select **Web Source** to launch the Web Source wizard. Browse the local files system to locate schema files (XSD) that you want to copy into the project, then click **OK**.

Registering ESB Projects and Services with the ESB Server

You use the integration server connection to register the ESB project and services that you have created with Oracle JDeveloper. This registration with the ESB Metadata Server is required to run these services, and to view and manage these services in the Oracle ESB Control. Before registering an ESB project and services, make sure you have a working integration server connection. See "Creating and Testing Connections" on page 2-6.

To register services using an integration server connection, follow these steps:

1. Right click the ESB project in the Application Navigator.
2. Select the **Register with ESB** from the menu.
3. Click a local ESB Server option or on a connection to another integration server that has been set up.
4. When the **ESB Registration Summary** dialog displays, click **OK**.

When you refresh the Oracle ESB Control, the ESB services that you registered with the ESB Metadata Server display in the Oracle ESB Control Service Navigation tree.

Note: You can also deploy one or multiple ESB projects to the ESB Metadata Server by using the custom ant tasks such as `deployESBProjects`. The files required for using the custom ant tasks are available in the following folder of your installation directory:

ORACLE_HOME`\integration\esb\deployment`

The `deployment` folder also contains the documentation about using the ant tasks.

Syncing Services From ESB Server

Use the **Sync from ESB Server** option to copy the metadata from the ESB Server and replace the corresponding metadata in Oracle JDeveloper. For example, if a routing rule has been modified in Oracle ESB Control, you can update that routing rule in Oracle JDeveloper.

This option only replaces Oracle JDeveloper metadata that currently exists in the ESB Server. When registering services, Oracle JDeveloper can determine that a change has occurred and gives you the option of syncing services. For information about registering services, see "Registering ESB Projects and Services with the ESB Server" on page 2-15.

To sync metadata from the ESB Server:

1. Click the Sync from ESB Server icon at the top of the Design tab. This is the middle icon (gear with sphere overlay) in the group of three icons. See Figure 2–2 on page 2-3 for a screenshot that shows the icons at the top of the Design tab.

2. Click **Yes** in the **Confirm Sync from ESB Server** dialog.

Deleting ESB Projects

To delete an ESB project in the **Application Navigator**, follow these steps:

1. Select the project or service in the **Application Navigator**.
2. Select **Delete** from the **Edit** menu.
3. Click **Yes** when the **Removing Files from IDE** dialog displays.

Note that deleting projects or services from the Oracle JDeveloper does not remove the object from Oracle ESB Control or the ESB Server.

3

Monitoring the Enterprise Service Bus

This chapter describes how to use Oracle ESB Control, a Web-based interface for managing, administering, and monitoring an Oracle Enterprise Service Bus.

This chapter contains the following topics:

- Overview of the Oracle ESB Control on page 3-1
- Understanding the Layout of the Oracle ESB Control on page 3-3
- Creating, Viewing, and Updating Organizational Units for Services on page 3-7
- Viewing and Updating Service Definitions on page 3-12
- Understanding and Managing Routing Rules on page 3-14
- Defining and Managing Tracking Fields on page 3-16

Overview of the Oracle ESB Control

The Oracle ESB Control is typically used at run time to monitor and fine tune the enterprise service bus. The Oracle ESB Control enables you to:

- Create, modify, and delete ESB systems and service groups
- View and modify routing rules for routing services, including filter expressions, transformations, execution type (synchronous or asynchronous), and so on.
- View and modify adapter and SOAP services
- View the connections among the various Oracle Enterprise Service Bus services in a schematic diagram
- View run-time statistics
- View system, group, and service definition details, including URLs required to call ESB services from applications that are external to the enterprise service bus
- Track message instances across the service bus in a schematic diagram
- View error conditions within a schematic, including the messages that did not reach their destination, and resubmit messages for retryable errors.
- View and define domain-value maps

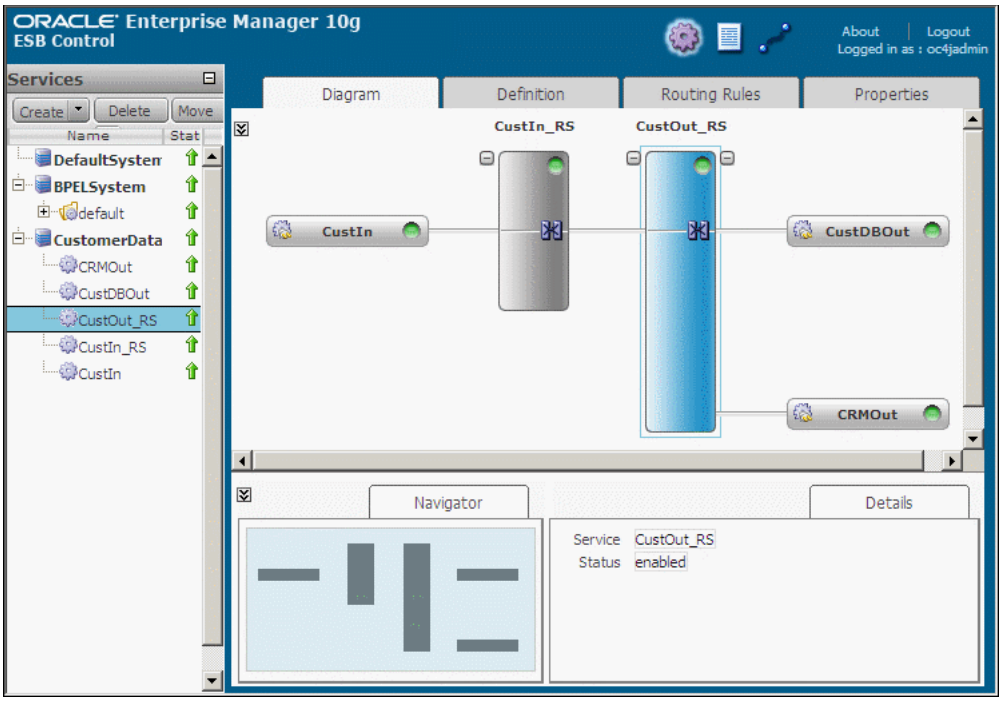
Note that many of these features are also available in Oracle JDeveloper. Table Table 3–1 summarizes the features available in each of these tools:

Table 3–1 Comparison of Features Available in the Oracle ESB Control and Oracle JDeveloper

Feature	Available in Oracle JDeveloper?	Available in Oracle ESB Control?
Create, modify, and delete ESB systems	Yes	Yes
Create, modify, and delete ESB routing service definitions	Yes	Yes
Create, modify, and delete routing rules (transformations, filter expression and so on)	Yes	Yes
Create a transformation file	Yes	No
Create, modify, and delete ESB adapter services	Yes	No
Create, modify, and delete SOAP invocation services	Yes	No
View service connections schematically	Yes	Yes
View run-time statistics	No	Yes
Specify notification channels	No	Yes
View message instances schematically	No	Yes
Resubmit messages that have failed due to an error	No	Yes
Define and view trackable fields	No	Yes
Define domain-value maps	No	Yes

Figure 3–1 shows the Oracle ESB Control with the schematic that is generated when you complete the tutorial documented in the *Oracle Enterprise Service Bus Quick Start Guide*. On the left side of the Oracle ESB Control is a navigation tree that enables you to select a system, group, or service; on the right side of the Oracle ESB Control are tabbed pages that display, and enable you to change, properties for an object selected in the navigation tree.

Figure 3–1 Oracle ESB Control - Services View Diagram Tab



For information about starting ESB components and opening the Oracle ESB Control, see "Opening the Oracle ESB Control" on page 1-15

Understanding the Layout of the Oracle ESB Control

The Oracle ESB Control provides three views, which you select by clicking the links that run across the top of the page. The currently selected view is indicated by a blue bubble that encloses the link. In Figure 3–1, for example, the currently selected view is Services.

Each of the views is described in the following subsections:

- Oracle ESB Control Services View on page 3-3
- Oracle ESB Control Instances View on page 3-5
- Oracle ESB Control Maps View on page 3-6

Note: All the tasks described in this section assume that the ESB Server is started and the database that is serving as the ESB repository is up and running. For information about starting the ESB Server, see "Starting, Stopping, and Accessing Oracle Enterprise Service Bus Components" on page 1-13.

Oracle ESB Control Services View

The Services view of the Oracle ESB Control, enables you to view service definitions, update routing rules, define trackable fields, and view a schematic diagram of the services.

As shown in Figure 3–1 on page 3-3, the Services view is divided into two main regions: a services panel on the right and tabbed pages on the left. The services panel

provides a navigation tree of ESB systems and the service groups and services within each, and buttons to create, delete, and move objects within the navigation tree.

The services panel appears on the left side of the Oracle ESB Control. It provides buttons for creating and deleting ESB systems, groups, and services and a navigation tree to present these items (and their current status) after they have been created.

The tabbed panel presents details about services selected in the navigation tree. Depending on the service selected, the following tabs are available:

- On the Diagram tab, view the relationship of the selected service to other objects within the ESB system. Oracle Enterprise Service Bus creates this diagram automatically, based on the definitions you specify for each service.

  The selected service is represented in blue within the diagram, whereas the other services are represented in gray. In the diagram, the following conventions are used:

  - Adapter services are represented by rectangles with a long horizontal plane.
  - Routing services are represented by rectangles with long vertical planes.
  - Communication between services is represented by the lines between the services.
  - A routing service filter expression is indicated by a funnel.
  - A transformation is indicated by two rectangles with an X overlay.

  The navigator tab indicates which area of the diagram is currently being displayed in relation to the entire diagram. This is useful for large diagrams that extend beyond the area that can be presented in the diagram region. When you move the scroll bar in the diagram region, notice that the blue window in the navigator scrolls also, to highlight the portion of the diagram that is currently visible.

  The detail tab provides the name of the selected service and the status of the selected service in the Diagram tab. Overlay metrics enable you to select which metrics, if any, you want to be superimposed on the diagram.

  See "Creating, Viewing, and Updating Organizational Units for Services" on page 3-7.

- On the Definition tab, view the definition of the selected service, enable it, or disable it. For outbound adapter services and routing services, these details including the concrete WSDL URL, which you can use to invoke the service using SOAP over HTTP from another application.

  See "Viewing and Updating Service Definitions" on page 3-12.

- On the Routing Rules tab, view, update, delete, and set the priority of routing rules for the selected routing service. The routing rules tab is presented only when you select a routing service from the navigation tree. It shows the rules that govern how a message is routed by the routing service.

  See "Understanding and Managing Routing Rules" on page 3-14 for more information about routing rules.

- On the Properties tab, view, update, and delete endpoint properties for the selected adapter or SOAP service. The Properties tab is presented only when you select an adapter or SOAP service from the navigation tree.

  Click the + on right side of the screen to add endpoint properties. Select the endpoint from the list under the Name column and enter a value in the Value column. Setting endpoints is similar to the actions in the endpoint properties panel

in Oracle JDeveloper. For more information about using endpoint properties, see "Using Endpoint Properties" on page 4-11.

- On the Trackable Fields tab, define trackable fields for the selected service. See "Defining and Managing Tracking Fields" on page 3-16.

Oracle ESB Control Instances View

The Instances view enables you to view details about message processing across an ESB system.

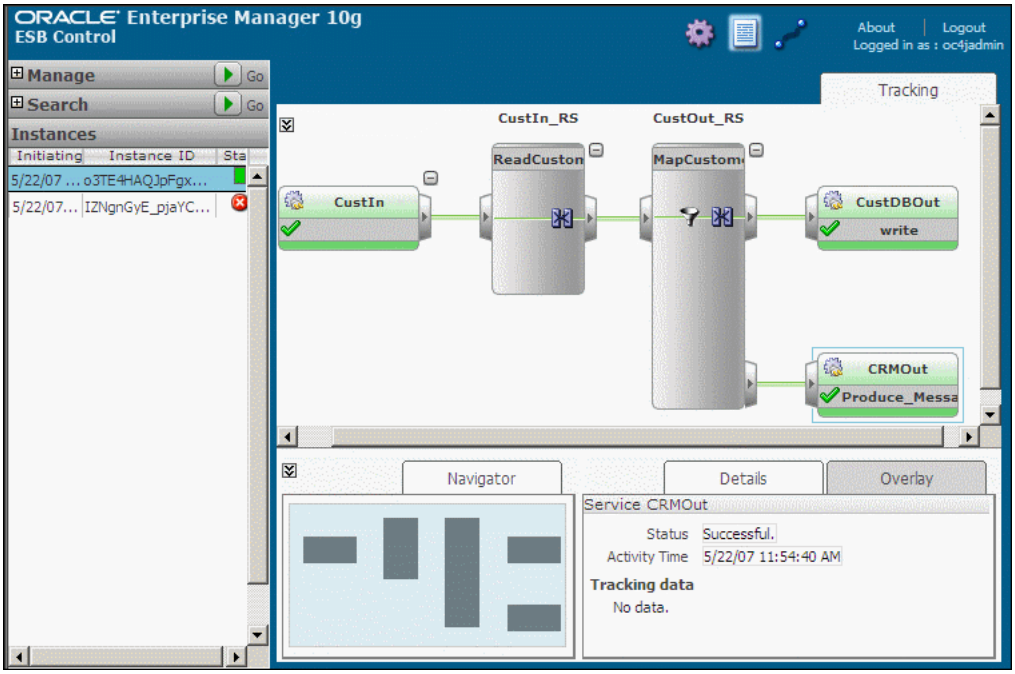
It enables you to filter messages based on any of the following properties:

- The service that processed them
- The status of the messages (Any, Error, Faulted, Processing, or Completed)
- Tracking name and tracking value
- Message IDs
- Time frame during which the message was processed

When you select a service for a message from the Instances panel, the message instance's path through the enterprise service bus is presented in the Tracking tab diagram (and the selected service is enclosed by dotted lines in the diagram). Within the diagram, endpoints that successfully processed the message are represented in green, services where an error occurred are represented in red, and services that processed the message successfully, but were rolled back due to an error are represented in yellow. Endpoints that were not invoked in the processing of the message are represented in gray.

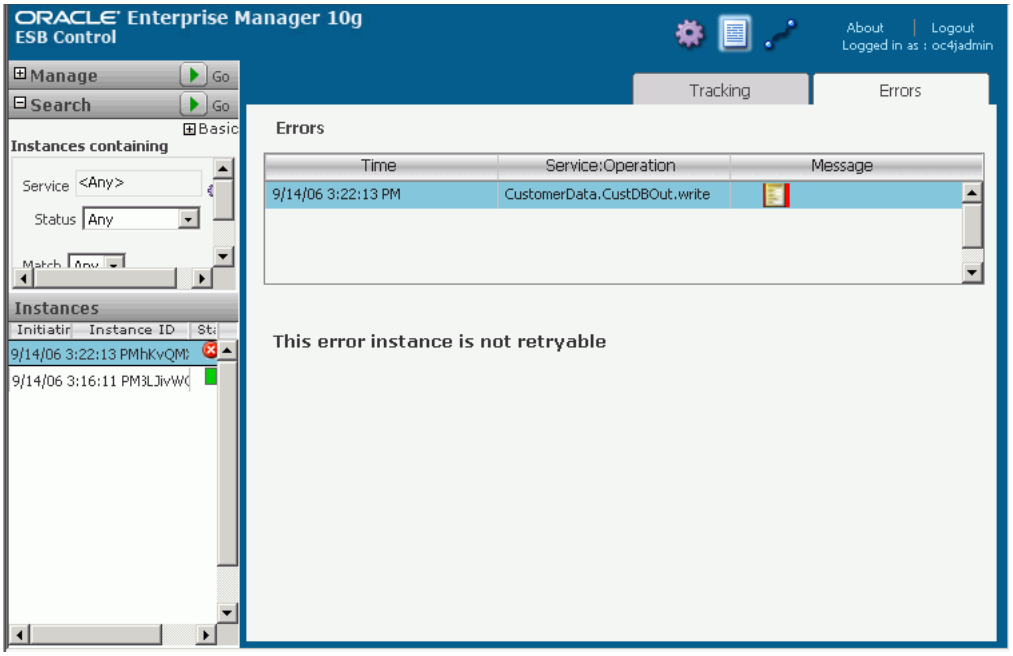
Figure 3–2 shows that the CRM adapter service (CRMOut) was successfully invoked and that the database adapter service (CustDBOut) was not invoked because a filter expression excluded the message from delivery to the database adapter service.

Figure 3–2 Oracle ESB Control - Instances View Tracking Tab



If you select a service where an error occurred for the message, an Error tab is presented, as shown in Figure 3–3. The Errors tab contains an error table that lists the time, service:operation, and message associated with each error. Click the Error Details icon in the Message column to view the error message and the stack trace.

Figure 3–3 Oracle ESB Control - Instances View Errors Tab



For more information message tracking, see Chapter 10, "Tracking Message Instances Across the Enterprise Service Bus". For more information about error handling, see Chapter 12, "Error Handling".

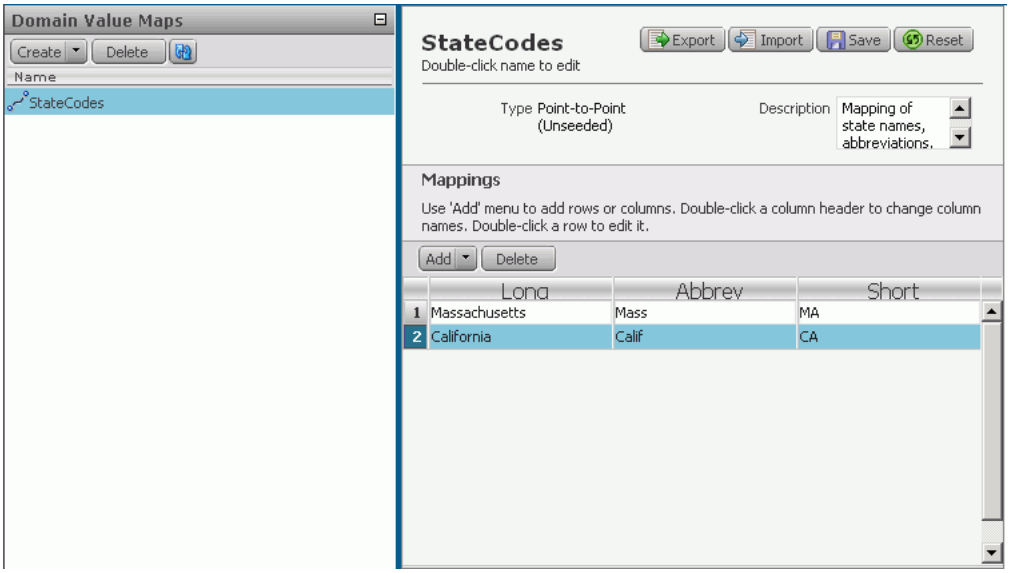
Oracle ESB Control Maps View

The Maps view enables you to create, update, and delete domain-value maps, as well as view, export, and import existing domain-value maps.

Applications that you want to integrate using Oracle Enterprise Service Bus likely use different values to represent the same information. For example, one application may represent the state of Massachusetts as `MA` whereas another application might represent it as `Massachusetts` or `Mass`. A domain-value map enables you to associate values from one application to values from another. Each domain-value map typically holds a specific category of value mappings among multiple applications. For example, one domain-value map might hold mappings for country codes and another might hold mappings for units of measurement.

Figure 3–4 shows a sample domain-value map in the Maps view of Oracle ESB Control. The names of the saved maps are listed in the left panel of the page and right panel shows the details about the mappings in a specific map. In this sample map, mappings are set up for the long, abbreviated, and short name of states in the columns of the map.

Figure 3–4 Oracle ESB Control - Maps View with Sample Domain-Value Map



For more information domain-value maps, see Chapter 7, "Domain-Value Maps".

Creating, Viewing, and Updating Organizational Units for Services

Oracle Enterprise Service Bus provides system and service group structures for organizing services. ESB systems are required; service groups are optional.

Every service you create must be defined as a child of either a service group or a system. A service is uniquely identified by its full path within the system/service group/service or system/service structure. A service is represented by a gear icon in the services navigation tree.

- ESB systems

  An Oracle Enterprise Service Bus system is an organizational unit, typically used to contain the services associated with of a single application, proxy for an application, or a technical system, such as an Oracle Applications instance. Examples of systems are:

  - An Oracle Applications instance
  - A set of transformations, Oracle BPEL Process Manager services, and adapter services for an SAP instance
  - A set of transformations, BPEL Services, and database services for a custom database application
  - A standalone Oracle Business Activity Monitoring instance, complex event processing (CEP) services, and other related services
  - An Oracle B2B engine (that serves as a proxy for trading partners) and related transformation services and other services
  - A set of services, adapter services, and Oracle BPEL Process Manager services representing a bridge to a third party integration infrastructure (such as webMethods, Inc, and IBM infrastructures)

  When you create a service, you must create it within the context of an Oracle Enterprise Service Bus system. A service cannot belong to multiple ESB systems.

You can specify that an administrator be alerted if an error or fault occurs for a service contained within an ESB system.

A system is represented in the Oracle ESB Control by a stacked disks icon.

For more information see "Creating an ESB System" on page 3-8 or "Viewing or Modifying an Existing ESB System Definition" on page 3-9.

- Service groups

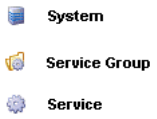
  Similar to systems, service groups are units for organizing services. Unlike systems, however, you are not required to create a service group before you can create a service. A service group must be created within the context of an ESB system. A service can belong to, at most, one service group.

  A service group is represented in the navigation tree by a folder icon on which a gear icon is superimposed.

  For more information, see "Creating Service Groups" on page 3-10 or "Viewing or Modifying an Existing Service Group" on page 3-11

Icons used to represent objects in the Oracle ESB Control are shown in Figure 3–5.

Figure 3–5 Icons Used in the Oracle ESB Control



Managing Oracle Enterprise Service Bus Systems and Service Groups

You can use the Oracle ESB Control to create new ESB systems, or view and modify existing ESB systems. The following sections provide step-by-step instructions:

- Creating an ESB System on page 3-8
- Viewing or Modifying an Existing ESB System Definition on page 3-9
- Creating Service Groups on page 3-10
- Viewing or Modifying an Existing Service Group on page 3-11
- Deleting Systems or Service Groups on page 3-11

Creating an ESB System

To create a system, follow these steps:

1. In the **Service** panel, click **Create.**

   A dialog box opens.

2. In the **What do you want to create?** field, select **System.**

   The dialog box refreshes to present only the fields required to create a system.

3. In the **Specify Name** field, enter a unique name for the system.

4. Click **OK.**

   The named system is added to the Services panel (with an downward pointing arrow that indicates that the system is disabled), and the Definition tab is presented.

5. On the **Definition** tab, optionally specify the following:
   - The Java Message Service (JMS) topic where you want messages that could not be delivered due to an errors be stored.
   - The JMS topic for storage of messages that are delivered asynchronously.
   - The notification channels that you want the ESB Server to use in the event that an error occurs. See "Setting Up Notification Channels" on page 9-4 for details.
6. On the **Definition** tab, click **Create.**

   An Update Service window opens to indicate that the system has been created. The arrow next to the named service in the Services panel now points upward to indicate that the service is enabled.
7. In the Update Service window, click **OK.**

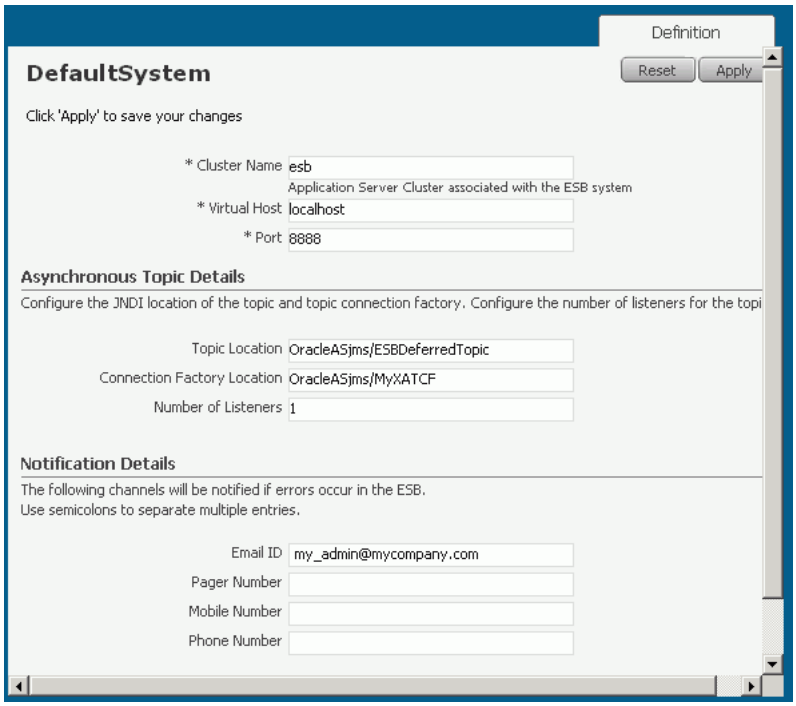
Viewing or Modifying an Existing ESB System Definition

To view or modify the definition for a previously created ESB system, follow these steps:

1. Click **Services** if the Services view is not currently displayed.
2. In the **Services** panel, click the name of the ESB system of interest.

   The Definition tab displays similar to Figure 3–6.

Figure 3–6 ESB System – Definition Tab



3. Review the properties presented on the **Definition** tab and described in Table 3–2.
4. Make any desired updates. All properties can be updated except the Asynchronous Topic and the Error Topic.
5. If you are satisfied with your changes, click **Apply;** otherwise, click **Reset** to return the properties to the settings that were presented when you opened the page.

Table 3–2 ESB System Definition Tab

Page Element	Description
Name	The name used in the Oracle ESB Control to identify the system.
Cluster Name	The name of the Oracle Application Server cluster associated with the system.
Virtual Host	The name of the host where the Oracle Application Server is running.
Port	The port number on the host where the Oracle Application Server is running.
Topic Location	The JNDI location of the JMS topic to which messages are published when messages are routed asynchronously.
Connection Factory Location	The connection factory location to which messages are published when messages are routed asynchronously.
Number of Listeners	The number of listeners for the JMS topic to which messages are published when messages are routed asynchronously.
Notification Details	The notification details specify the communication channel or channels that will be used to alert an administrator that an error has occurred within the Oracle Enterprise Service Bus system. You can specify notifications with the following channels: ▪ Email ID ▪ Pager number ▪ Mobile number ▪ Phone number See "Setting Up Notification Channels" on page 9-4.

Creating Service Groups

Similar to systems, service groups are units for organizing services. Unlike systems, however, you are not required to create a service group before you can create a service. A service can belong to, at most, one service group.

To create a service group:

1. Click **Services** if the Services view is not currently displayed.
2. In the **Service** panel, click **Create.**

   A dialog box opens.
3. In the **What do you want to create?** field, select **Service Group.**

   The dialog box refreshes to present only the fields required to create a system.
4. In the **Specify Name** field, enter a name for the service group that is unique within the system or parent service group in which you are creating it.
5. To the right of the **Choose the parent System or Service Group** field, click the icon.

   The Choose Parent System or Service Group dialog box opens.
6. In the Choose Parent System or Service Group dialog box follow these steps:
   a. Select the name of the parent system or service group.
   b. Click **Select.**
7. Click **OK.**

The dialog box closes and the service group name is added to the navigation tree. The status column indicates that the service group is disabled.

8. On the **Definition** tab, in the **Description** field, optionally enter a description for the service group.
9. On the Definition tab, click **Create.**

   A window opens to indicate that the changes have been successfully applied.
10. Click **OK.**

    The status column in the Services panel indicates that the status of the newly created group is up.

Viewing or Modifying an Existing Service Group

After you have created a service group, you can change its name or description, if desired. However you cannot change its parent.

To view or modify a service group, follow these steps:

1. Click **Services** if the Services view is not currently displayed.
2. In the **Services** panel, click the name of the service group of interest.
3. Review the properties presented on the **Definition** tab and described in Table 3–3.
4. Make any desired updates. All properties can be updated except the Identifier.
5. If you are satisfied with your changes, click **Apply;** otherwise, click **Reset** to return the properties to the settings that were presented when you opened the page in Step 2.

Table 3–3 ESB Service Group Definition Tab

Page Element	Description
Identifier	The name by which the ESB Server identifies the service group. When you create a service group, ESB Server creates an identifier using the format *systemname*System.*servicegroupname*, where *systemname* is the name of the system in which the service group was created and *servicegroupname* is the name that was originally specified for the service group when it was created. If you change the name of the service group, the name used in the Services panel is updated to reflect your change, but the identifier remains the same. The ESB Server uses the identifier, not the name, to refer to the service group in error messages.
Name	The name used in the Oracle ESB Control to identify the service group.
Parent Name	The ESB system that contains the service group.
Description	A user-specified description of the service group.

Deleting Systems or Service Groups

To delete a system or service group, select the system or service group and click **Delete**. You need to confirm the action to delete the object from the Oracle ESB Control.

Note that deleting a system or service group from the Oracle ESB Control does not remove the object from Oracle JDeveloper. If you register the ESB project again with

the ESB Server in Oracle JDeveloper, it will appear in Oracle ESB Control after a refresh.

Viewing and Updating Service Definitions

Although you cannot create services using the Oracle ESB Control, you can view the service definitions, modify some properties, and create or modifying routing rules for existing services, as described in the following sections:

- Enabling and Disabling Services on page 3-12
- Viewing Service Definitions on page 3-12

For information about creating services, see Chapter 4, "Creating Inbound and Outbound Services" and Chapter 5, "Creating Routing Services and Routing Rules".

Enabling and Disabling Services

To enable or disable a service, follow these steps:

1. Click **Services** if the Services view is not currently displayed.
2. In the Services panel, navigate to the service of interest, and then select it.
3. Click the **Definition** tab.
4. Click **Enable** or **Disable**, as desired.

Note: By default, all services are enabled. When you explicitly disable a service, an error is thrown back to the invoker.

In case of a routing service, the routing rule errors out when the target service is disabled. This is applicable to synchronous as well as asynchronous routing rules.

Viewing Service Definitions

To view the definition of an ESB service, follow these steps:

1. Click **Services** if the Services view is not currently displayed.
2. In the Services panel, navigate to the service of interest, and then select it.
3. Click the **Definition** tab.

The Definition page summarizes the contents of the WSDL file for the selected service. It consists of the following three regions for all services, except inbound adapter services. Inbound adapter services present the overall section only, and only a subset of the fields for that region.

- General

  This region of information, which is not labeled, appears at the top of the page. The elements that are presented in this region of the page are described in Table 3–4. For inbound adapter services, only the following elements are presented: Name, Type, Description, and Inbound Adapter WSDL URL.

Table 3–4 Service Definition – General Region

Page Element	Description
Name	The name that was entered to identify the routing service when it was created.
Type	The type of this ESB service. The value of this field is always Routing Service for a routing service.
Description	An optional user-specified description of the routing service.
Invocable from an external service	Indicates whether the service can be invoked from an external service.
Concrete WSDL URL	The concrete WSDL URL for the service. Use this URL to call the service from any external Web Services client (such as .Net). Click the URL to view the WSDL in your default browser.
Port Type	A name that refers to the set of operations performed by this service. This name corresponds to the name attribute of the portType element in the WSDL that defines this routing service.
Namespace	The XML namespace specified in the WSDL that defines this routing service.
WSDL URL	The abstract WSDL URL that ESB Server used to configure the service. Click the URL to view the WSDL in your default browser.
SOAP Endpoint URI	The URl of the SOAP endpoint. Click the URL to open the Test Web Service page. **Note**: For information on using the Test Web Service page, see "Testing the ESB Services" on page 9-9 and *Oracle SOA Suite Developer's Guide*.

- Operations

  This region of the page lists the operations that the service WSDL describes and their type. Table 3–5 describes this region of the page.

Table 3–5 Service Definition - Operations Region

Page Element	Description
Name	Lists the names of the operations specified in the service WSDL URL.
Type	Lists the type for each operation specified in the service WSDL URL, such as One Way, Request Response, or Request Response Fault.

- Operation Details

  This region of the page provides details on an operation selected in the Operations region. Table 3–6 describes this region of the page. Which tabs are presented in this region depend on the operation definition, as follows:

  – One-way operations

    If the selected operation is defined as a one-way operation (only input elements are defined for the operation in the WSDL), then only the Request tab is active.

  – Request/response operation

If the selected operation is defined as a request/response operation (both input and output elements are defined for the operation in the WSDL), then both the Request and Response tabs are active.

– Faults

If a fault is defined for the operation (fault elements are defined for the operation in the WSDL), then the fault tab is active. Note that a fault is simply a special response that can be directed to the requester.

Table 3–6 Service Definition - Operation Details Region

Page Element	Description
Validate Payload at Runtime	Enables you to specify whether or not you want the message payload validated against the schema (XSD) definition at runtime. Select to enable, deselect to disable. If you select this option and a message payload received by the service is invalid, the message handled as described in "Managing Error Conditions" on page 12-2.
Element	The message payload element name. This corresponds to the <part> element's `element` attribute specified in the WSDL URL.
Schema Location	Specifies the location of the schema, if any, used by the service.

Understanding and Managing Routing Rules

While a routing service determines the operations that will be performed on the messages received by the routing service, whether or not it can be invoked by an external service, and whether or not the message payload will be validated by the routing service at run time, routing rules determine the following:

- The set of target services to which message instances will be sent from the routing service
- The operation that will be applied to a message instance upon reaching the target service
- Whether a transformation will be applied to the structure of a message instance before being sent to a given target service
- Whether some message instances will not be sent to a target service, on the basis of a filter applied to the message payload

This section discusses the following topics:

- Creating or Modifying Routing Rules on page 3-14
- Viewing Routing Rules on page 3-15
- Deleting Routing Rules on page 3-16

For information about creating routing services, see Chapter 5, "Creating Routing Services and Routing Rules".

Creating or Modifying Routing Rules

Although routing rules are typically specified as part of the development phase using Oracle JDeveloper, the Oracle ESB Control also provides tools for creating and modifying routing rules at run time. Although the method for accessing the controls are different for each tool, the actual controls are the same.

To access the controls for creating a routing rule using the Oracle ESB Control, follow these steps:

1. In the **Services** panel, click the routing service for which you want to create or modify routing rules.
2. Click the Routing Rules tab.

The controls on Routing Rules tab in the Oracle ESB Control are similar to the Routing Rules tab in Oracle JDeveloper. For information about using these controls, see "Specifying Routing Rules" on page 5-18.

Viewing Routing Rules

To view the current routing rules for a routing service, follow these steps:

1. In the **Services** panel, click the routing service for which you want to view the routing rules.
2. Click the **Routing Rules** tab.

   Table 3–7 describes the page elements on this tab.

Table 3–7 Routing Rules Tab in Oracle ESB Control

Page Element	Description
Priority arrows	The green arrows at the top of the routing rules table enable to you to set the priority of the routing rules relative to other synchronously executed routing rules.
Filter Expression	Specifies an expression by which you want messages to be filtered for the selected rule. You might specify a filter expression, for example, if you only want messages written to a database for a customer record if that customer address is in the United States. This field is optional. Click the icon to the right of this field to open the Expression Builder. See "Using An Expression for Filtering Messages Based on Payload" on page 5-20 for more information.
Transformation Map File	Specifies the XSLT file that defines the transformation you want to apply to the data format of message before it is delivered to the service to which it is being routed. You might specify an XSLT file to transform message data from a canonical format to the format expected by the database receiving that data. This field is optional. Click the icon to the right of this field to open the Map Browser dialog box, which enables you to search for and browse among defined and registered Data Map (.xslt) files or to create a new one. See "Creating an XSL Map File for Data Structure Transformation" on page 5-20 for more information.
Target Operation	Specifies the service and operation to which messages will be sent. Click the icon to the right of this field to open the Service Operation Browser dialog box, which enables you to search for and browse among registered service operations.

Table 3–7 (Cont.) Routing Rules Tab in Oracle ESB Control

Page Element	Description
Accept Messages From	Specifies the ESB systems from which messages will be accepted by the target service, as follows: ■ Any System Select this option to specify that the target service will accept messages from any ESB system. ■ Same System Select this option to specify that the target service will accept only messages that originate from the same ESB system as that to which the target service belongs. ■ Other Systems Select this option to specify that the target service will accept only messages that originate from the a different ESB system than that to which the target service belongs.
Rule Execution	Specifies whether execution of the routing rule will be synchronous or asynchronous. See "Specifying Synchronous or Asynchronous Execution" on page 5-27 for more information.

Deleting Routing Rules

To delete a routing rule:

1. In the **Services** panel, click the routing service from which you want to delete a routing rule.
2. Click the Routing Rules tab.
3. Click the plus sign (+) to the left of the routing rule to select it.
4. Click the minus (-) button on the far right of the routing rule page to delete the selected routing rule.
5. Click **Apply**.

Defining and Managing Tracking Fields

The Oracle ESB Control enables you to track the path particular message instances take across the Oracle Enterprise Service Bus and the current routing status of that message.

The mechanism by which you track messages is using trackable fields. A **trackable field** is a name-value pair, where the name is any meaningful string that you specify (such as CustomerName or POId) and the value is an XPath expression defined for the input and output message of an entire service or an operation performed by the service.

Using the Trackable Fields tab in the Services view of the Oracle ESB Control, you can define one or more trackable fields for each message associated with a service and operation to keep track of the messages that the service operation processes at run time. You can then search the message instances that the service or operation has processed using the Instances view of the Oracle ESB Control.

The following sections provide step-by-step instructions on the tasks associated with trackable fields:

- Understanding the Trackable Fields Tab on page 3-17

- Defining and Updating Trackable Fields on page 3-17
- Using the Expression Builder to Specify a Trackable Field Expression on page 3-18
- Enabling and Disabling Trackable Fields on page 3-19
- Deleting Trackable Fields on page 3-20

Understanding the Trackable Fields Tab

The Trackable Fields tab contains the following fields:

- Name

  A unique name that you provide for the trackable field.

- Operation

  The operation associated with the service that you select for the trackable field.

- Message

  The message type of the service that you select for the trackable field.

- Expression

  The expression that you want to track for the service. This field includes a variable of the message instance that is tracked.

- Enabled

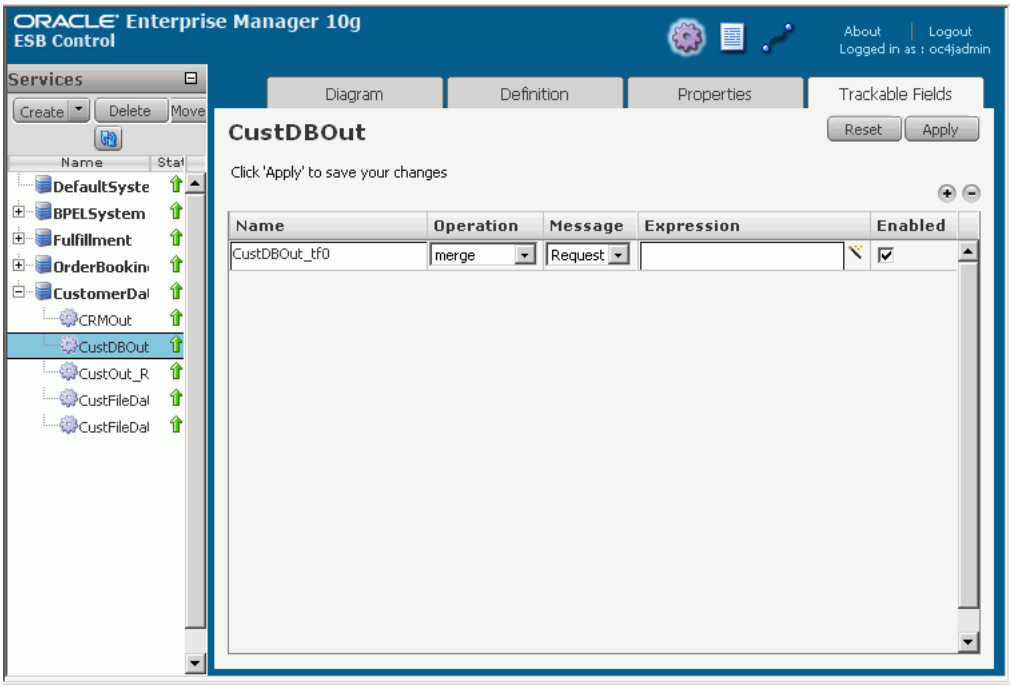
  The status of the trackable field you can set to enable or disable.

Defining and Updating Trackable Fields

To define or update trackable fields, follow these steps:

1. At the top of the Oracle ESB Control, click the **Services** button if the Services view is not currently displaying.
2. In the **Services** panel, select the service for which you want to set trackable fields.
3. Click the **Trackable Fields** tab.
4. Click the plus (+) button to add a row to the table and enable editing for that row; double-click an existing row to update that row.

Figure 3–7 Services View – Trackable Fields Tab



5. Enter values in the row, as follows:

   a. In the **Name** field, enter a name for the trackable field that is unique across the ESB system in which the service exists.

   b. In the Operation field, select from the list the operation on which you want to define the trackable field.

   c. In the Message field, select from the list the type of message you want to track for this service. Possible values are: Request, Response, Fault.

   d. In the Expression field, enter the XPath expression that identifies the field that you want to track.

      Click the wand icon to the right of the Expression field to open the Expression Builder, which assists you in building the XPath expression. For information about the Expression Builder, see "Filter Expression Overview" on page 5-10.

   e. In the Enabled field, click the check box to enabled the specified trackable field. Deselect the check box to disable it.

      If you disable the trackable field, the trackable field information is not logged and is not available in the Instances view of the Oracle ESB Control.

   f. Click **OK** when finished.

6. Click **Apply** to save the changes you have made; click **Reset** to return to the values that were presented before you began editing fields.

Using the Expression Builder to Specify a Trackable Field Expression

For information about the Expression Builder, see "Filter Expression Overview" on page 5-10.

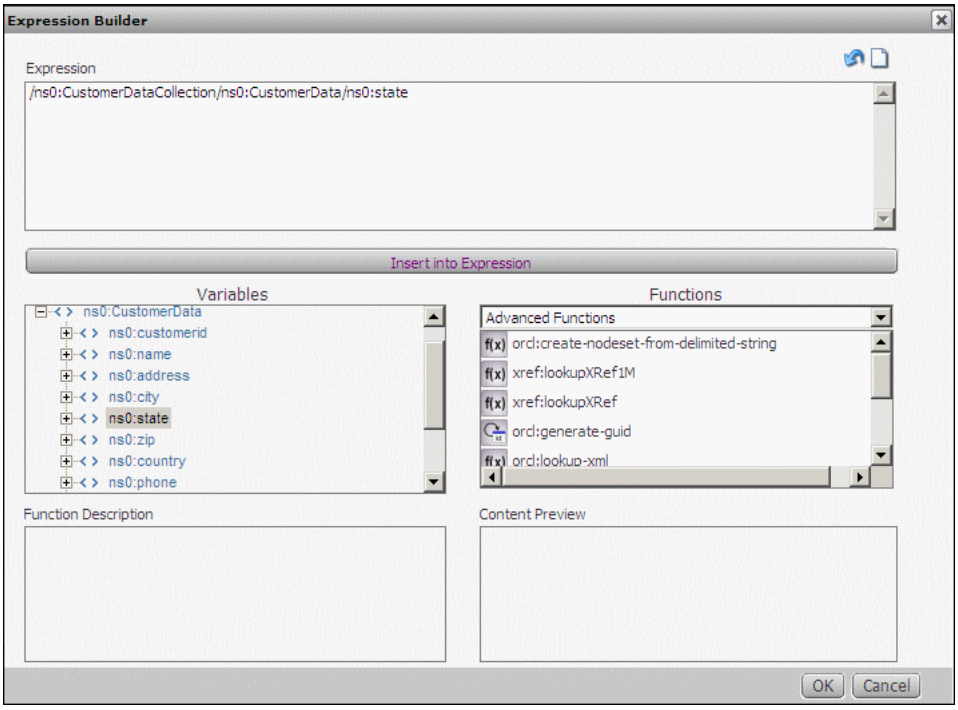
1. At the top of the Oracle ESB Control, click the **Services** button if the Services view is not currently displaying.

2. In the **Services** panel, select the service for which you want to set trackable fields.
3. Click the **Trackable Fields** tab.
4. Click the plus (+) button to add a row to the table and enable editing for that row; double-click an existing row to update that row.
5. Click the plus \ icon to launch the Expression Builder.
6. Select and expand the Variables tree and select a variable.

> **Note:** If you are not able to expand the Variables tree, then you can manually enter the expression in the Expression field.

7. Click **Insert into Expression**.

Figure 3–8 Trackable Fields - Expression Builder



8. Click **OK**.
9. Click **Apply** to save the changes you have made; click **Reset** to return to the values that were presented before you began editing fields.

Enabling and Disabling Trackable Fields

To enable or disable trackable fields, follow these steps:

1. At the top of the Oracle ESB Control, click the **Services** button if the Services view is not currently displaying.
2. Click the **Trackable Fields** tab.
3. Double-click the row that contains the trackable field that you want to enable or disable.

4. In the **Enabled** column, select the check box to enable the trackable field, deselect the check box to disable the trackable field.
5. Click **Apply.**

Deleting Trackable Fields

To delete a trackable field:

1. At the top of the Oracle ESB Control, click the **Services** button if the Services view is not currently displaying.
2. Click the **Trackable Fields** tab.
3. Click the row that contains the trackable field to select it.

   A selected row is highlighted in blue.
4. Click the minus (-) button.
5. Click **Apply.**

4

Creating Inbound and Outbound Services

This chapter discusses creating inbound and outbound services.

This chapter contains the following topics:

- Configuring Adapter Services with Oracle Enterprise Service Bus on page 4-1
- Creating a BPEL Partner Link to an ESB Service on page 4-12
- Calling an ESB Service From an External Service on page 4-13

Configuring Adapter Services with Oracle Enterprise Service Bus

The services you create with Oracle JDeveloper enable you to integrate the Oracle Enterprise Service Bus with file systems, database tables, database queues, Java Message Services (JMS), MQ services, and Oracle E-Business Suite and any SOAP service. In addition to these services, which get messages into the enterprise service bus, Oracle Enterprise Service Bus provides a service known as a routing service, which determines the path that messages take across the enterprise service bus to get from a source endpoint to a target endpoint.

Services are the core of the enterprise service bus. You design an Oracle Enterprise Service Bus by creating a variety of services to move messages onto, across, and off of the service bus.

To move data on to the service bus, you use inbound adapter services or have an external application call an ESB service; to move data off of the service bus, you use an outbound adapter service or invoke an external SOAP service. To move data across the service bus and transform the data structure from the structure presented by the source application to the structure required by the target application you use routing services.

The following sections describe adapter services and routing services in more detail.

- Using Adapter Services on page 4-2
- Creating Adapter Services on page 4-3
- Using SOAP Invocation Services on page 4-7
- Browsing for Deployed Services on page 4-10
- Using Endpoint Properties on page 4-11

See Chapter 5, "Creating Routing Services and Routing Rules" for information about routing services.

Using Adapter Services

Oracle Enterprise Service Bus provides support for creating services for the Oracle Technology adapters. The Oracle Technology adapters enable you to integrate mainframe and legacy applications with enterprise resource planning (ERP), customer relationship management (CRM), database, and messaging systems.

Table 4–1 provides a summary of the Oracle Technology adapter services you can create. Adapters services can be configured as inbound or outbound adapters services. Inbound adapter services send messages to the enterprise service bus, while outbound adapter services send messages to an application or system external to the enterprise service bus.

Table 4–1 Summary of Oracle Technology Adapters

Adapter Service	Description
AQ Adapter Service	Sends or receives messages from Oracle Advanced Queuing single or multiconsumer queues
Database Adapter Service	Sends or receives messages extracted from an Oracle Database table or created by executing a stored procedure
File Adapter Service	Sends or receives messages from a file in the local file system
FTP Adapter Service	Sends or receives messages from a file at a remote FTP server
JMS Adapter Service	Sends or receives messages from a JMS queue or topic
MQ Adapter	Sends or receives messages from IBM's MQ Series
Oracle Applications Adapter Service	Sends or receives messages from an Oracle E-Business Suite interface

This section includes the following topics:

- Creating Adapter Services on page 4-3
- Modifying Adapter Services on page 4-4
- Deleting Adapter Services on page 4-4
- Example: Creating an Inbound File Adapter on page 4-4

See Also:

- *Oracle Application Server Adapter Concepts* for an overview of adapters
- *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide* for specific details about configuring adapters for Oracle Enterprise Service Bus
- *Oracle Application Server Adapter for Oracle Applications User's Guide* for information about using the Oracle Applications adapter for Oracle E-Business Suite
- *Oracle Enterprise Service Bus Quick Start Guide* for a tutorial that describes how to design an enterprise service bus that uses a file adapter service, a database adapter services and a JMS adapter service.

Creating Adapter Services

Oracle JDeveloper provides wizards that assist you in creating inbound and outbound adapter services. The wizard collects the necessary information to generate the WSDL file that defines the service.

To add an adapter service, perform the following steps:

1. In the Application navigator, navigate to the ESB project for which you want to create an adapter service, expand the **Resources** folder and double click *project-name*.esb, where *project-name* is the name of the project to which you want to add the SOAP service.

   The Design tab for the project is displayed.

2. In the **Component Palette**, click the down arrow and select **Adapter Services** if not already selected.

3. Drag and drop an adapter into the **Design** tab.

   The create adapter service dialog box for the specific adapter type opens.

4. Enter the **Name**, **System/Group**, and an optional **Description** for the adapter.

   Name: Enter a unique name across the ESB system in which you are creating the service; spaces are not allowed. For example: `CustomerDataFileIn`.

   System/Group: Click the flashlight (torch) to open the ESB Service Group Browser dialog and select the system/group for this service. For example: **CustomerData** under **Systems/Groups in project**.

   Description: Enter an optional description. For example: This adapter service reads records from a local data file.

5. To complete WSDL File field, you can click the following icons to the right of the field:

   - Configure adapter service wsdl icon

     Click this icon to launch the adapter configuration wizard for the specific type of adapter you have selected from the Component Palette. The wizard guides you through the setups.

     For details about configuring the AQ, database, file, FTP, JMS, and MQ adapters in Oracle JDeveloper, see *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide*. For details about configuring the Oracle application adapter for Oracle E-Business Suite, see *Oracle Application Server Adapter for Oracle Applications User's Guide*.

   - Service Explorer icon

     Click this icon to launch the Service Explorer and search for deployed services.

     See "Browsing for Deployed Services" on page 4-10.

6. In the **Port Type** field, click the **down arrow**, and then select the port type for the routing service.

7. Click **OK**.

After you complete the process for creating an adapter service, an icon for the service appears in the Design tab. In the Applications Navigator, files with `esbsvc` and `wsdl` extensions are created in the Resources folder of the project. The `esbsvc` file provides the definition of the ESB service. The wsdl file defines the input and output messages for this instance flow, the supported client interface and operations, and other features.

In addition to the wsdl files that are named with the adapter service name, there are standard JCA header files that are created with a new adapter. The header file name is of the form `adapter_type`AdapterOutboundHeader.wsdl or `adapter_type`AdapterInboundHeader.wsdl, depending on whether the service is inbound or outbound. The `adapter_type` specifies the type of adapter, such as `DB`, `file`, or `jms`. For information about the adapter inbound and outbound header WSDL files, see *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide*.

When you create an inbound adapter service, Oracle JDeveloper automatically creates a routing service for the newly created adapter and sets up the link between the two. The name of this routing service is the name of the inbound adapter appended with the string `_RS`, as an abbreviation for routing service. See "Specifying Routing Rules" on page 5-18.

Modifying Adapter Services

To modify an existing adapter service, perform the following steps:

1. Double click the upper section of the adapter service icon in the **Design** tab. The cursor is shaped like a hand on that region of the icon.
2. Make your changes to the adapter service information that can be modified. Note that the name of the service cannot be changed.
3. In the Endpoint Properties panel, you can add, delete, update, or view endpoint properties for the service. Click the **+** next to **Endpoint Properties** to open the panel. See "Using Endpoint Properties" on page 4-11.
4. Save your changes.

Deleting Adapter Services

To delete an adapter service, perform the following steps:

1. Select the adapter service icon in the **Design** tab.
2. Click the large red X at the top of Design tab to delete the selected adapter service.
3. Confirm that you want to delete the selected service.
4. Save your changes.

> **Note:** Do not delete adapter services in the Application Navigator.

Example: Creating an Inbound File Adapter

For an example of the information that is collected by Oracle JDeveloper wizards, assume you want to create inbound file adapter service to read in local XML files from the `c:\customer\in` directory. To create this adapter, perform these steps:

1. In the Applications navigator, navigate to the ESB project for which you want to create a SOAP service, expand the **Resources** folder and double click *project-name*.esb, where *project-name* is the name of the project to which you want to add the SOAP service.

   The Design tab for the project is displayed.
2. In the **Component Palette**, click the down arrow and select **Adapter Services** if not already selected.
3. Drag and drop **File Adapter** into the **Design** tab.

The Create File Adapter Service dialog box opens.

4. In the Create File Adapter Service dialog box, enter the name.

   Enter a unique name across the ESB system in which you are creating the service; spaces are not allowed.

   For example: `CustFileDataIn`

5. In the Create File Adapter Service dialog box, enter the system/group.

   Click the browse (flashlight) icon to display the ESB Service Group Browser dialog. Locate the system/group where you want to place this adapter. For example: `CustomerData` under ESB > Systems/Groups in project

6. In the Create File Adapter Service dialog box, enter the optional description.

   For example: File adapter service for inbound data files

7. In the Create File Adapter Service dialog box, enter the WSDL file.

   a. Click the Configure adapter service wsdl icon to launch the Adapter Configuration Wizard.

   b. In the Welcome page, click Next.

   c. In the Service page, accept the Service Name and click Next.

      You can add an optional description.

   d. In the Operation page, enter the Operation type and name, then click Next.

      For example:

      Operation type: Read file

      Operation Name: ReadCustFileData

   e. In the File Directories page, update the directory information and click Next.

      For example:

      Directory Names are Specified as: Physical Path

      Directory for Incoming Files (physical path): `c:\customer\in`

      Delete files after successful retrieval: check box to enable

      Accept the other default setups on the page.

   f. In the File Filtering page, update the filtering setups and click Next.

      For example:

      Name Patterns are specified with: File Wildcards (po*.txt)

      Include Files with Name Pattern: *.xml

      Accept the other default setups on the page.

   g. In the File Polling page, update the polling setups and click Next.

      Polling Frequency: 10 seconds

      Minimum File Age: 0 seconds

      Accept the other default setups on the page.

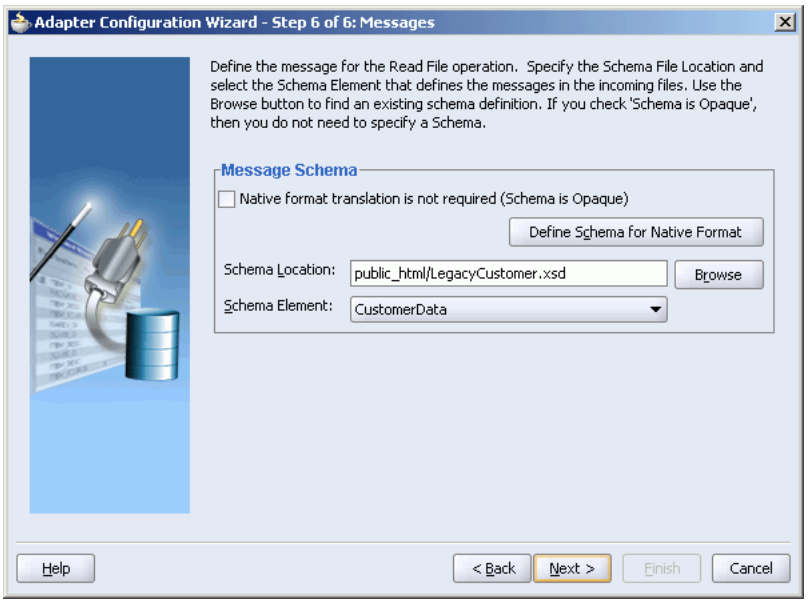
   h. In the Messages page, specify the message schema and click Next.

      For example:

To locate an existing CustomerData schema, click the Browse button for Schema Location to display the Type Chooser dialog box. Expand Project Schema Files /Legacy Customer.xsd and select CustomerData. Click OK.

Accept the other default setups on the page.

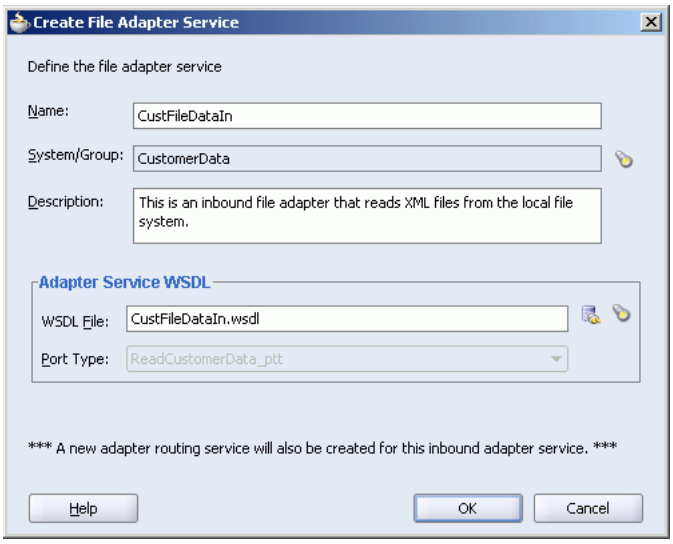
Figure 4–1 Adapter Configuration Wizard - Messages Page



i. In the Finish page, click Finish to complete the Adapter Configuration Wizard.

8. In the Create File Adapter Service dialog box, click OK to create the service.

Figure 4–2 Create File Adapter Service Dialog

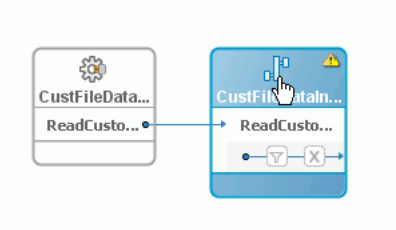


The new adapter service appears in the Design tab with a new routing service. The routing service has the `_RS` suffix attached to the name.

For example, when use the Oracle JDeveloper Adapter Configuration wizard to create an inbound file adapter service named `CustFileDataIn`, Oracle JDeveloper

automatically creates a routing service named `CustDataFileIn_RS` and creates a connection from the `CustDataFileIn` adapter service to the `CustDataFileIn_RS` routing service, as illustrated in Figure 4–3.

Figure 4–3 Automatic Routing Service Creation



You now need to add the routing rules for this new routing service. See "Specifying Routing Rules" on page 5-18.

Using SOAP Invocation Services

A SOAP invocation service enables you to integrate external Web services, such as Apache AXIS, JDeveloper Web Services and .NET, into Oracle Enterprise Service Bus.

This section includes the following topics:

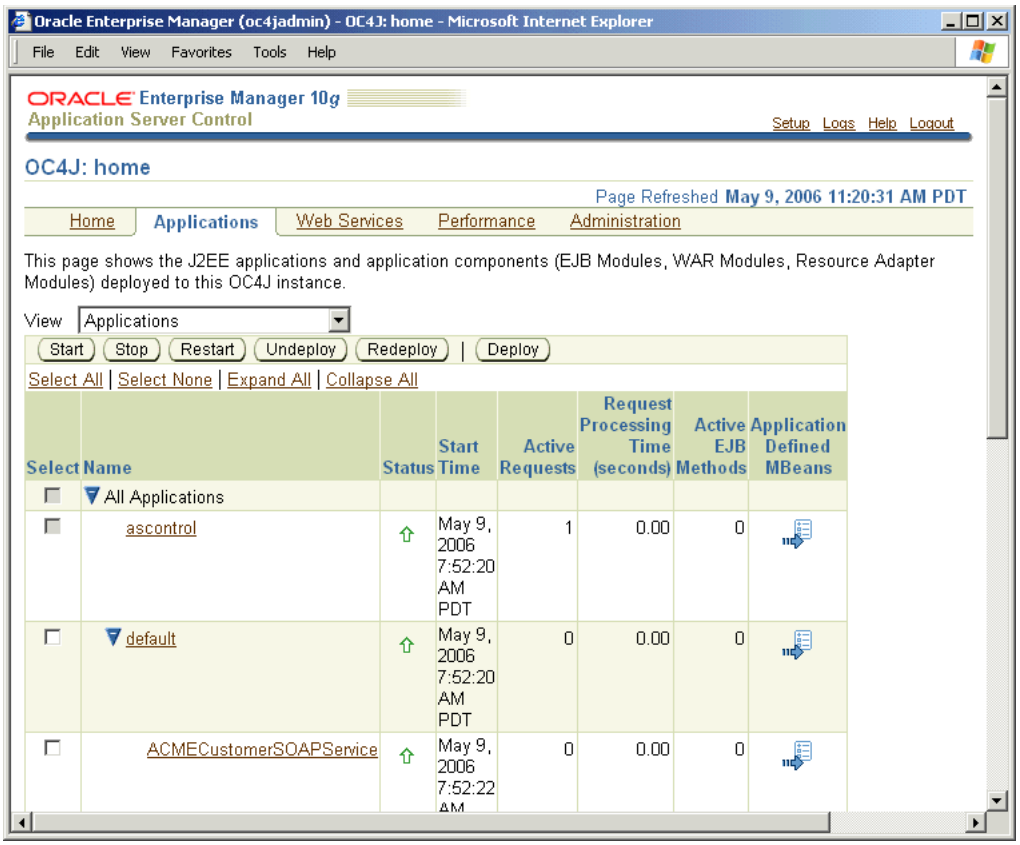
- Creating a SOAP Service on page 4-7
- Modifying SOAP Services on page 4-9
- Deleting SOAP Services on page 4-9

Creating a SOAP Service

This section describes how to use Oracle JDeveloper to create an outbound SOAP service that integrates with an external service.

The external service could be an application deployed in an OC4J instance, such as `ACMECustomerSOAPService` shown in Figure 4–4.

Figure 4–4 Oracle Application Server Control – Applications



To create an outbound SOAP service with Oracle JDeveloper, follow these steps:

1. In the Applications navigator, navigate to the ESB project for which you want to create a SOAP service, expand the **Resources** folder and double click *project-name*.esb, where *project-name* is the name of the project to which you want to add the SOAP service.

   The Design tab for the project is displayed.

2. In the **Component Palette**, click the down arrow and select ESB services if not already selected.

3. Drag and drop **SOAP Service** into the **Design** tab.

   The Create SOAP Invocation Service dialog box opens.

4. In the Create SOAP Invocation Service dialog box, enter the following information:

   a. Name

   Enter a unique name across the ESB system in which you are creating the service; spaces are not allowed.

   For example: `ACMEService`

   b. System/Group

   For example: `CustomerData`

   c. Description

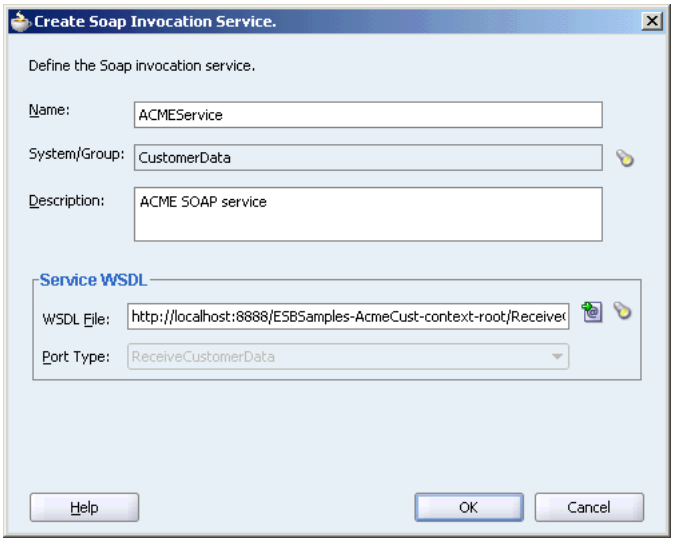
   For example: A SOAP Service

d. WSDL File

For example:
`http://localhost:8888/ESBSamples-AcmeCust-context-root/ReceiveCustomerDataSoapHttpPort?WSDL`

e. Port Type

For example: `ReceiveCustomerData`



5. Click **OK** when finished.

   The new SOAP service appears in the Design tab.

Now you need to add a routing rule that has the new SOAP service as a target. For information about defining routing rules, see "Specifying Routing Rules" on page 5-18.

Modifying SOAP Services

You can modify SOAP services in Oracle JDeveloper by opening the SOAP service property page.

To modify a SOAP service:

1. In the Oracle JDeveloper **Design** tab, double click the upper section of the SOAP service icon. The cursor is shaped like a hand on that region of the icon.
2. Make your changes to the service information that can be modified. Note that the name of the service cannot be changed.
3. In the Endpoint Properties panel, you can add, delete, update, or view endpoint properties for the service. Click the **+** next to **Endpoint Properties** to open the panel. See "Using Endpoint Properties" on page 4-11.
4. Save your changes.

Deleting SOAP Services

You can delete a SOAP service in the Oracle JDeveloper Design tab.

To delete a SOAP service:

1. In the Oracle JDeveloper **Design** tab, select the SOAP service icon.
2. Click the large red **X** at the top of Design tab to delete the selected service.

3. Confirm that you want to delete the selected service.
4. Save your changes.

Note: Do not delete SOAP services in the Application Navigator.

Browsing for Deployed Services

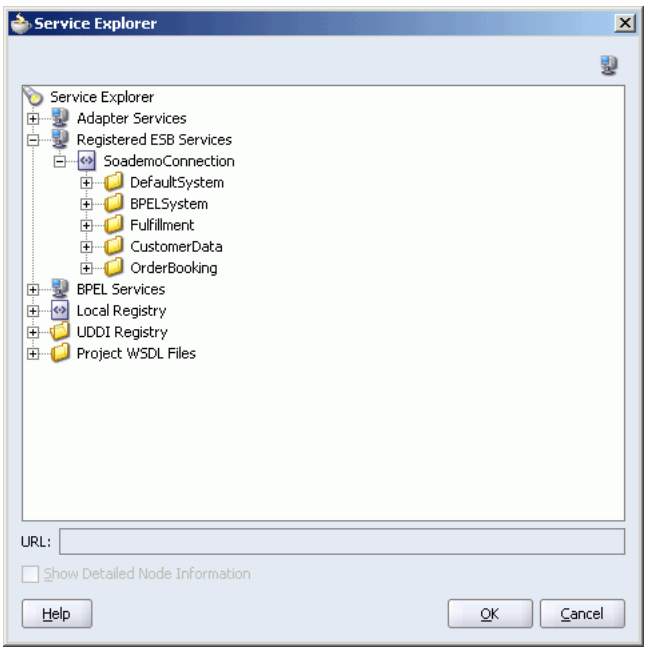
You can browse for deployed services with the Service Explorer. The Service Explorer is available when creating a adapter, routing, or SOAP service.

To browse for deployed services, perform the following steps.

1. If you have not already done so, open the Create Adapter, Routing, or SOAP Service dialog.
2. Enter information for other fields on the page, such as Name, System/Group, and Description.
3. If you are creating a routing service, choose **Select Existing WSDL** in the Create Routing Service dialog.
4. To the right of the **WSDL Location** field, click the flashlight (torch) icon to open the Service Explorer dialog.
5. Select a branch of the Service Explorer tree, such as **BPEL Server Connection**.
6. Expand the server connection folder to view the existing projects.

   For example, expand the integration server connection that is set up.
7. Expand the project to view the deployed services.
8. Select a deployed service.

Figure 4–5 Service Explorer Dialog



9. Click **OK** when done to close the dialog box.

The **Service Explorer** dialog box closes and you are returned to the Create Service dialog box.

Using Endpoint Properties

Each adapter type and SOAP service has a specific set of endpoint properties that are provided when the service is created. For example, you can specify a new URL, or URLs, for the `Location` endpoint property of a SOAP service. When you specify multiple URLs for `Location` endpoint property, the SOAP service can be exposed externally to multiple endpoints. If one endpoint is down, Oracle Enterprise Service Bus redirects the request to alternative endpoint.

You can also change the `RetryCount` and `RetryInterval` endpoint properties to override the default values for adapter error handling. For information about adapter error handling, see "Inbound Adapter Error Handling" on page 12-2.

There are two types of endpoint properties: predefined and logical.

- Predefined (override) properties

  These properties are defined in the adapter wizard and the value can be overridden or a value can be provided at runtime.

- Logical properties

  For the file and FTP adapters you can specify that the directories are logical rather than physical. These logical names appear as endpoint properties in the Endpoint Property Chooser dialog.

  In addition, any adapter property can be made into a logical property by inserting a `$` into the wizard value (or manually edit the WSDL file). For example, if the user specifies `$myName` as a value in the adapter wizard, then `myName` appear in the Endpoint Property Chooser dialog.

You can add, delete, update, or view endpoint properties for the service in the Endpoint Properties panel when modifying an adapter or SOAP service in Oracle JDeveloper. See "Modifying Adapter Services" on page 4-4 and "Creating a SOAP Service" on page 4-7. You can also access endpoints in the Property tab of the Oracle ESB Control Services view. See "Oracle ESB Control Services View" on page 3-3.

Specifying Endpoint Properties

You can add and modify endpoint properties in Oracle JDeveloper by opening the adapter or SOAP service property page.

To add or modify endpoint properties:

1. In the Oracle JDeveloper **Design** tab, double click the upper section of the adapter or SOAP service icon to open the properties page. The cursor is shaped like a hand on that region of the icon.

2. In the Endpoint Properties panel, you can add, delete, update, or view endpoint properties for the service.

   Click the **+** next to **Endpoint Properties** to open the panel. The Endpoint Properties panel includes the following options:

   - Large Green Plus Sign (+) in upper right

     Opens the Endpoint Properties Chooser dialog to enable you to add an endpoint property to the service. Select a property from the list in the Endpoint Properties Chooser dialog and click the OK button to add the

endpoint property. Place the cursor over the property name to view the tooltip description for the property.

- Large Red X in upper right

  Deletes the selected endpoint property.

- Name

  Name of the endpoint property. Place the cursor over the property name to view the tooltip description for the property.

- Value

  Click in this field to enter or update the value of the endpoint property.

3. Select **File** > **Save** to save your changes to the property page.

Creating a BPEL Partner Link to an ESB Service

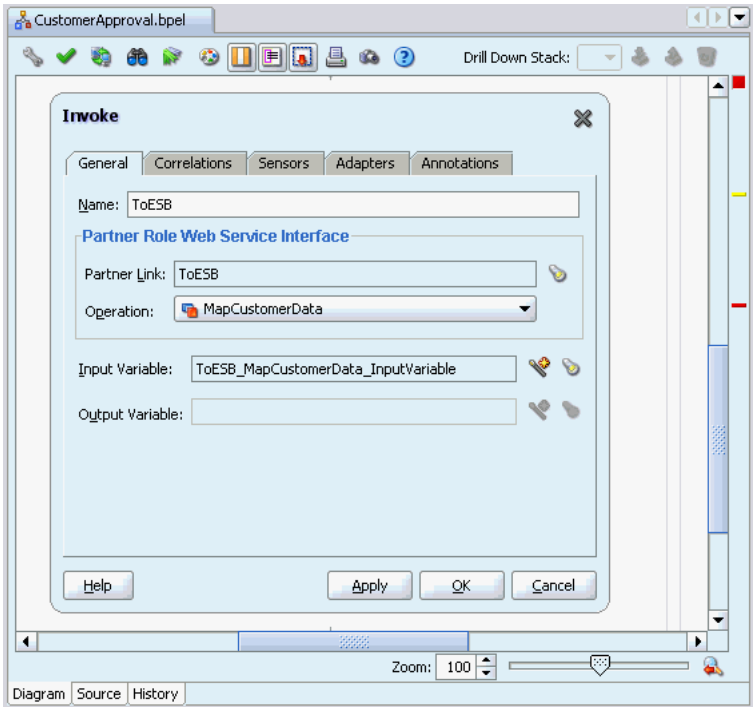
You can create a partner link from BPEL to an ESB service, as shown in Figure 4–6, if SOA Suite is installed with both BPEL and ESB. For example, you can create a partner link for an Invoke activity that invokes an ESB routing service.

To create a partner link from BPEL to invoke an ESB service:

1. In an Oracle JDeveloper BPEL Process project, drag and a drop an Invoke activity from the Process Activities Component palette to the Design tab.
2. Enter a name in the Name field in the Invoke dialog.
3. In the Invoke dialog, click the torch icon next to Partner Link field to display the Partner Link Chooser dialog.
4. In the Partner Link Chooser, select **Partner Links** and click the magic wand to display the Create Partner Link dialog.
5. In the Create Partner Link dialog, enter a name in the Name field.
6. In the Create Partner Link dialog, click the torch icon to display the Service Explorer dialog.
7. In the Service Explorer dialog, select the desired ESB routing service in the Registered ESB Services tree and click OK.
8. In the Create Partner Link dialog, select the Partner Role from the list and click OK.
9. In the Create Partner Link dialog, click the magic wand next to the Input Variable field to display the Create Variable dialog.
10. In the Create Variable dialog, click OK.
11. In the Invoke dialog, click OK.

After you have successfully deployed the BPEL process to the integration server, the ESB service that you specified in the BPEL partner link will be invoked by the BPEL process.

Figure 4–6 BPEL Partner Link



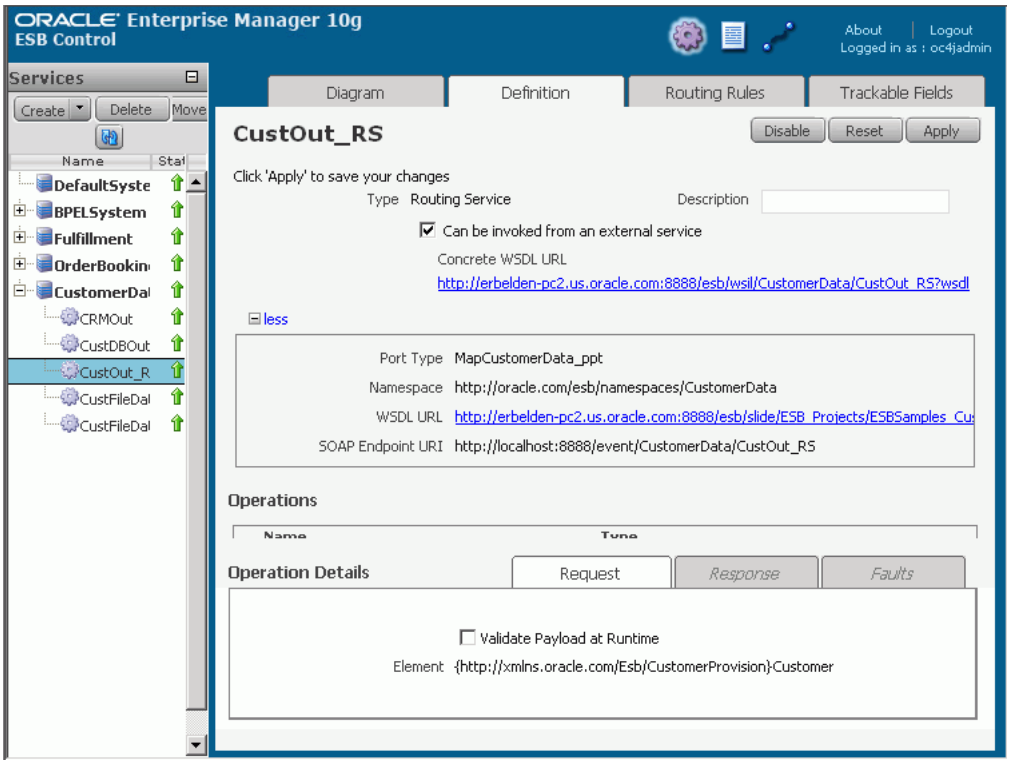
For information about creating BPEL projects, see *Oracle BPEL Process Manager Developer's Guide*.

Calling an ESB Service From an External Service

You can call an ESB service from an external service with the Concrete WSDL URL of the service. You can find the Concrete WSDL URL in the Definition tab of the Oracle ESB Control Services view, shown in Figure 4–7. Copy and paste the Concrete WSDL URL into the external tool that you are using to call the ESB service.

Note that you need to specify that the ESB service can be invoked from an external service in the service property page in Oracle JDeveloper or in the Definition tab of Oracle ESB Control. For information about the service property page in Oracle JDeveloper, see "Overview of Service Definitions and Routing Rules" on page 2-4. For information about the service Definition tab in Oracle ESB Control, see "Viewing Service Definitions" on page 3-12.

Figure 4–7 Oracle ESB Control – Definition Tab of Services View



For information testing an ESB service in Oracle Enterprise Manager, see "Testing the ESB Services" on page 9-9.

5

Creating Routing Services and Routing Rules

This chapter introduces routing services and routing rules.

This chapter contains the following topics:

- Introduction to Routing Services and Routing Rules on page 5-1
- Creating and Modifying Routing Services on page 5-13

Introduction to Routing Services and Routing Rules

A routing service is the key component for moving a message across the enterprise service bus – from its entry point to its exit point. Oracle JDeveloper provides tools that assist you in creating a routing service

The following are the key components that define a routing service:

- WSDL file
- Target services and operations
- Transformation definition
- Filter Expression
- Execution type (synchronous or asynchronous)
- The ESB systems from which the routing service accepts messages

The WSDL specifies how other services (either within or outside of the enterprise service bus) call the routing service. The remaining items, referred to as routing rules, determine where the routing service sends each message it receives, how it sends it, and what, if any, changes it makes to the message structure prior to sending it to the target service.

Oracle JDeveloper provides tools that assist you in creating the routing service WSDL and defining the routing rules.

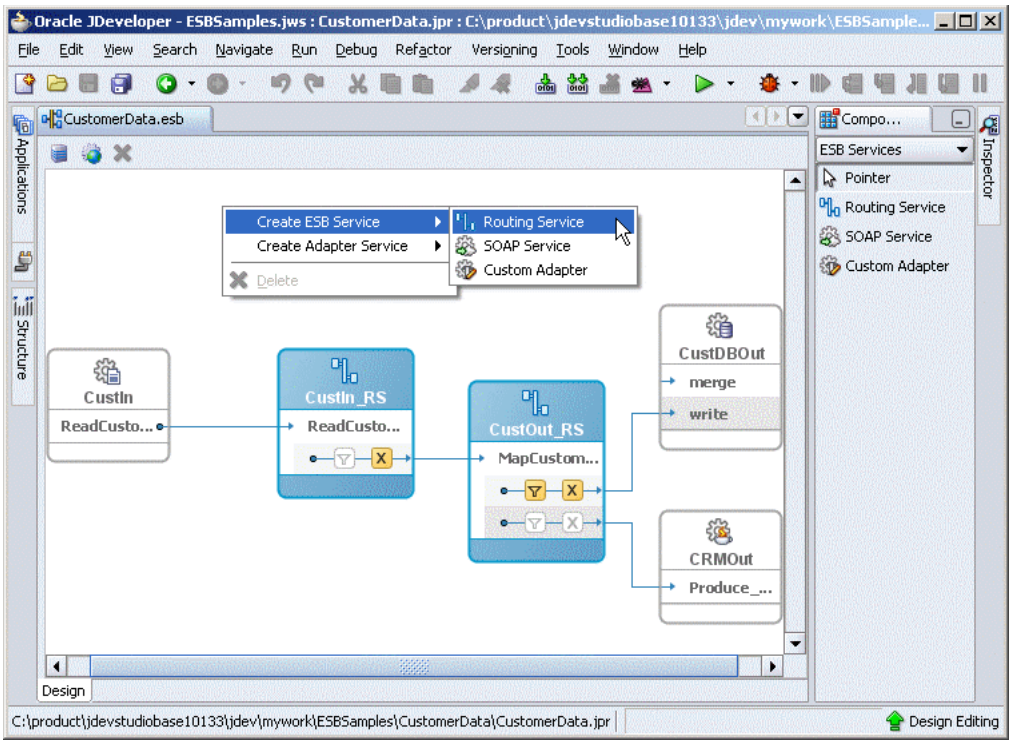
The following sections provide an overview of concepts and how the routing service and routing rules are specified. "Creating and Modifying Routing Services" on page 5-13 provides step-by-step instructions on performing these tasks.

- Overview of Specifying the Routing Service WSDL on page 5-2
- Overview of Specifying Routing Rules on page 5-7

Overview of Specifying the Routing Service WSDL

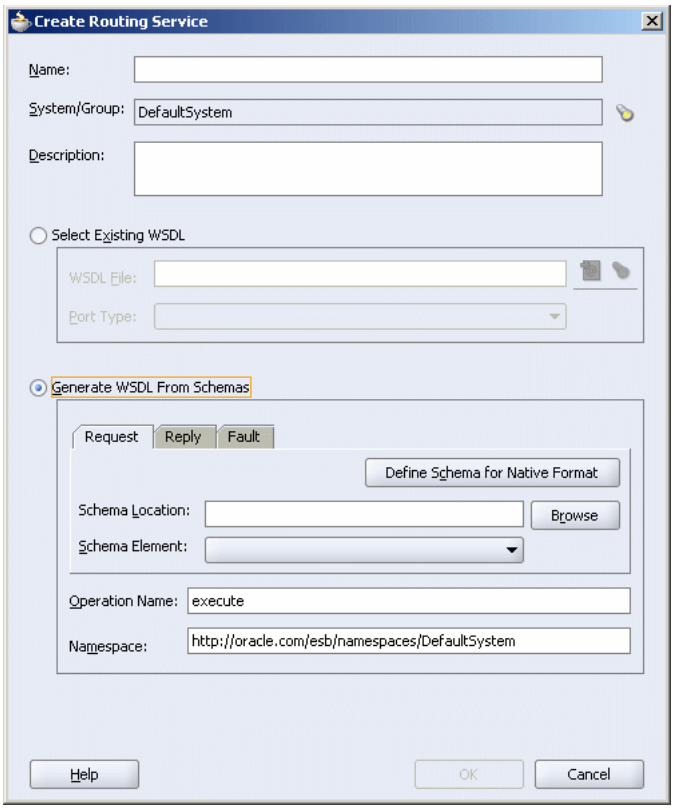
To create a routing service, you can drag and drop an ESB routing service from the Component Palette into the Design tab. You can also right-click in the Design tab, then select **Create ESB Service** > **Routing Service** as illustrated in Figure 5–1.

Figure 5–1 Creating a Routing Service



When you create a routing service, Oracle JDeveloper opens a dialog box, as shown in Figure 5–2.

Figure 5–2 Create Routing Service Dialog Box



As Figure 5–2 indicates, there are two main ways you can create the routing service WSDL:

- By specifying an existing WSDL file and one of the port types defined within that WSDL

  This option enables you to use an existing WSDL file (on the local file system or at a Oracle JDeveloper connection) to define the routing service. Oracle JDeveloper parses the WSDL you specify to present the list of port types from which you make a selection.

  If you want to edit an existing WSDL for the routing service, edit it using a WSDL editor (such as Oracle JDeveloper) prior to specifying it in this dialog box.

- By generating the WSDL from a schema file (XSD)

  This option enables you to use an existing XSD file or a file in a native file format (such as a comma-delimited value (CSV) file, a fixed-length file, a document type definition (DTD) file or a COBOL copybook file) to define the routing service.

  You can specify the same or different schema files for the request, response, and fault message schemas, which Oracle JDeveloper converts into WSDL input, output, and fault elements in the WSDL. Minimally, you must specify the schema for the request message. You cannot specify a fault message schema, unless you also specify a response.

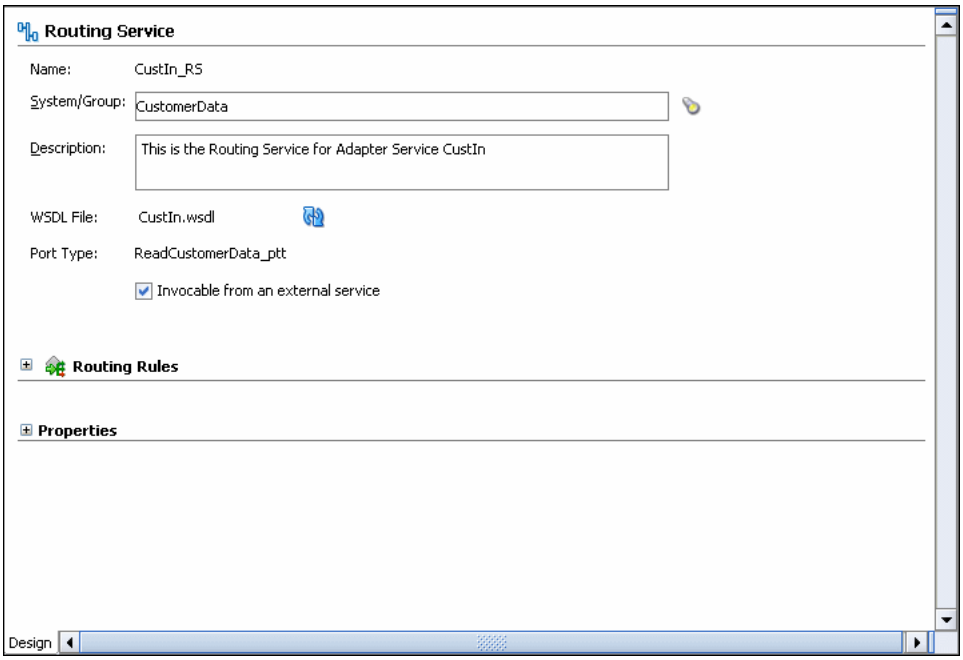
  In addition, you specify the operation and namespace. Oracle JDeveloper converts the operation into a operation element in the WSDL file and the namespace you specify is defined as the tns namespace in the WSDL file.

  The operation element describes how input to the operation is defined.

> **Note:** WSDLs with multiple faults are not supported.

When you complete the Create Routing Service dialog box, an icon is added to the Design tab, as shown in Figure 5–1 on page 5-2. The name of the routing service appears at the top of the icon, the name of the routing service operation appears below that. A yellow triangle with the question mark indicates that the routing rules for this routing service have not been specified yet. When you double-click the routing service icon, the routing service Property Sheet is displayed as shown in Figure 5–3.

Figure 5–3 Routing Service



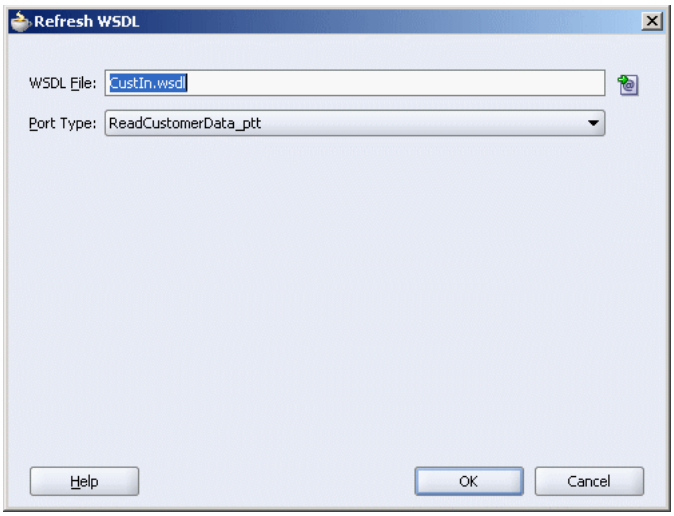
Modifying the Service WSDL File

You can modify the WSDL file for a service by adding or removing operations. Oracle JDeveloper provides a WSDL editor that can be used to modify the WSDL file. You can open the WSDL editor by double-clicking the WSDL file in Application Navigator. After modifying the file, you can use the Refresh Operation From WSDL icon shown in Figure 5–3 to synchronize the routing service with changes made to the WSDL.

> **Note:** it is not recommended that you make modifications to a WSDL file other than adding or deleting operations. For example, if you change the message schema type, then a transformation in a routing rule that is expecting the old schema will fail.

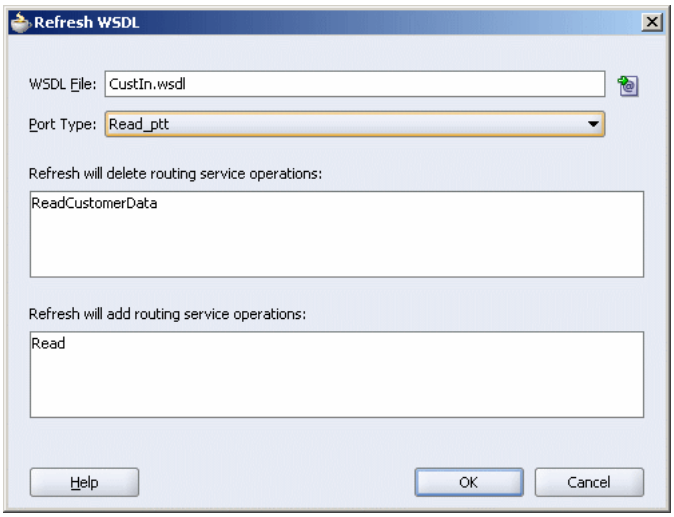
When you click the Refresh Operation From WSDL icon, the Refresh WSDL dialog is displayed as shown in Figure 5–4.

Figure 5–4 Refresh WSDL Dialog



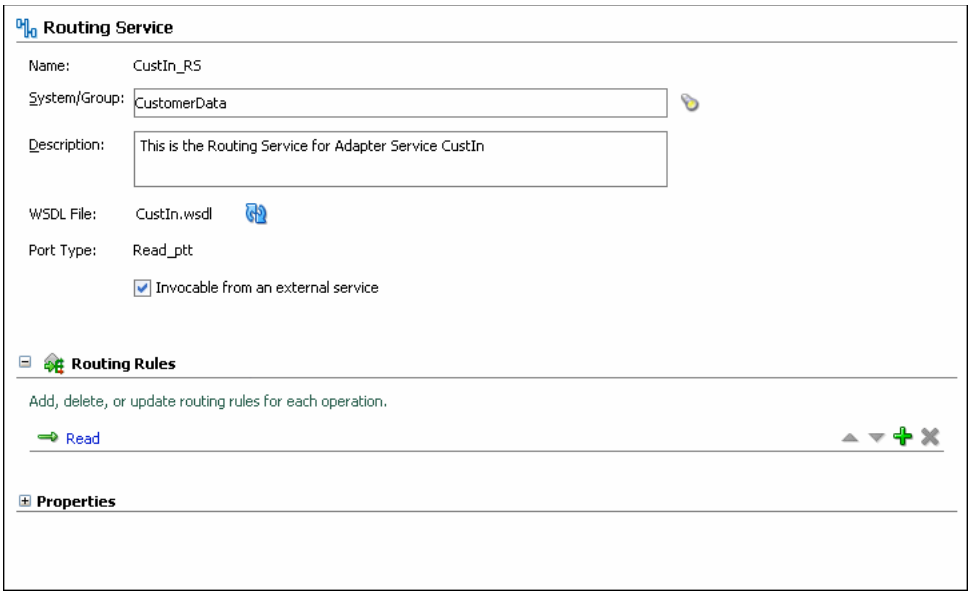
Click the **Find existing WSDLs** icon to the right of WSDL File field to select the WSDL file. The Refresh WSDL dialog is updated as shown in Figure 5–5.

Figure 5–5 Populated Refresh WSDL Dialog



The CustIn_RS routing service shown in Figure 5–3 is updated as shown in Figure 5–6.

Figure 5–6 Modified Routing Service



Specifying Routing Service Properties

You can add and modify routing service properties by using the Properties panel of the routing service property sheet shown in Figure 5–6. For example, you can specify priority of a routing rule. The priority determines the order in which the message for a routing rule gets processed. You can access the `priority` property of a routing rule in following way:

```
OperationName.priority
```

For example: `ReadCustomerData.priority`

The priority value can range from 0 to 9. The message with priority 9 has a highest priority during dequeue. The default priority is set to 4.

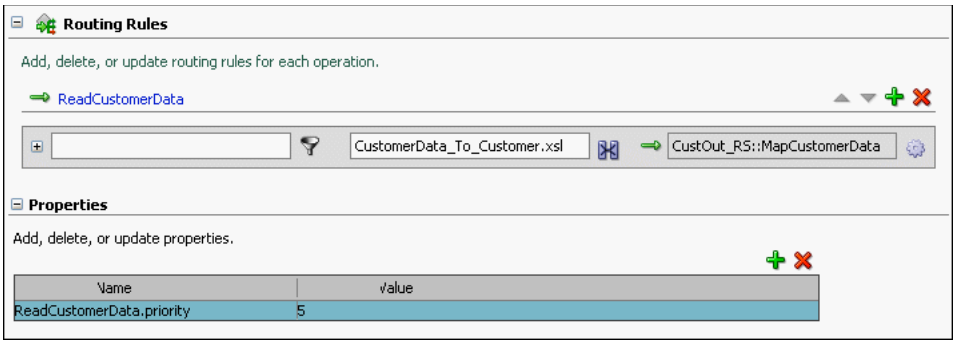
Perform the following steps to specify properties of a routing service:

1. Click the + next to **Properties** to open the panel.
2. Click the large Green Plus Sign (+) in upper right.

   A new line is added where you can specify property name and value.
3. Specify a property name in the Name column.
4. Specify value for the property in the Value field.
5. Save the changes.

Figure 5–7 shows how you can specify priority for a routing rules.

Figure 5–7 Routing Rules Priority

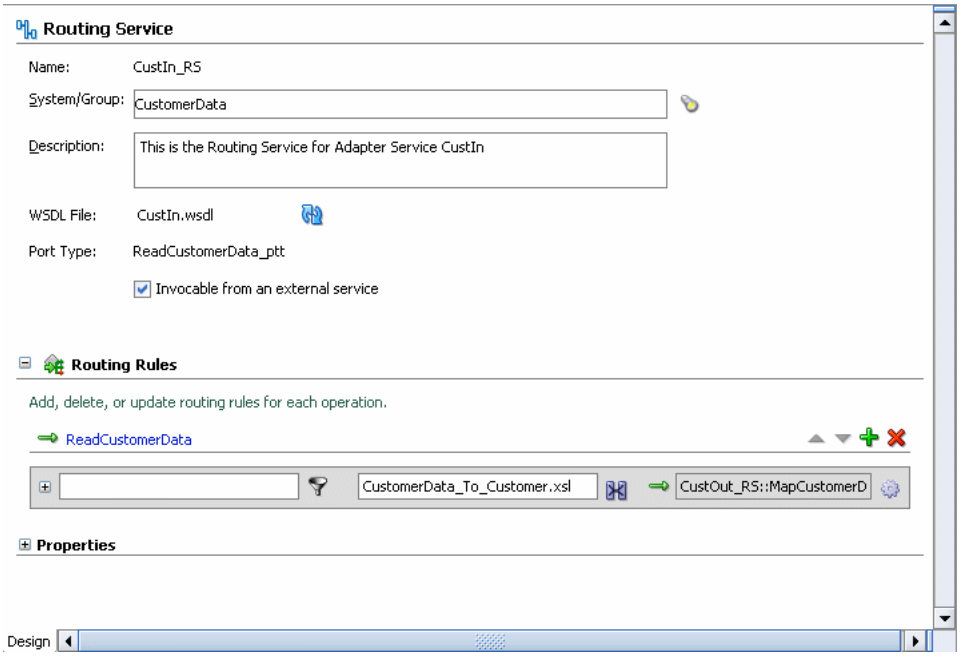


Overview of Specifying Routing Rules

After you have created a routing service and at least one of the targets to which that routing service will send messages, you can specify the routing rules to that target or targets. You can specify routing rules using Oracle JDeveloper at design time or using the Oracle ESB Control at run time. The interface for specifying routing rules is similar in both interfaces; this section presents screen captures from Oracle JDeveloper.

You can access the page for specifying routing rules using Oracle JDeveloper by double-clicking the icon on the Design tab that corresponds to the routing service for which you want to specify routing rules. The property page has several sections, including one labeled Routing Rules, as shown in Figure 5–8.

Figure 5–8 Routing Rules Properties Page



Clicking the large green + icon to the far right of the routing service operation, shown in Figure 5–8, creates a routing rule and launches the service browser, allowing you to choose the target service and operation for this new rule. When you click the small + icon to the left of Routing Rules, fields such as shown in Figure 5–9 are presented.

Note that the operation specified when the routing service was defined appears with a green arrow next to it. The green arrow indicates that only a request message schema

was specified for the routing service. If request, reply, and fault message schemas were specified, three arrows would appear, such as shown in Figure 5–9. If a fault message schema is not specified, then the red arrow is not presented.

Figure 5–9 Routing Rules – Request/Reply/Response Schema

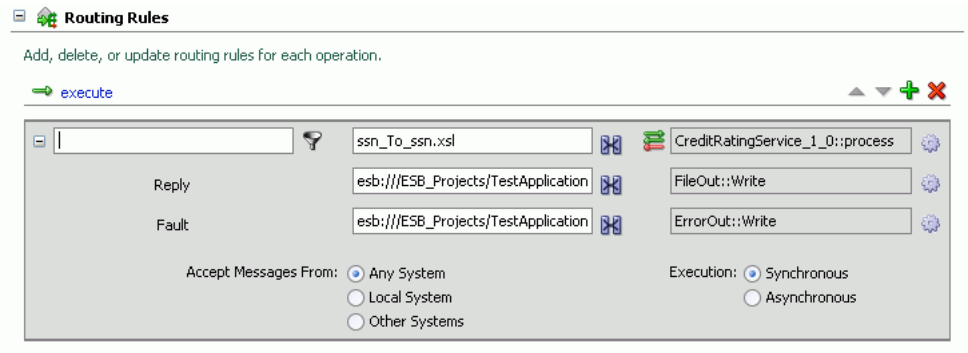
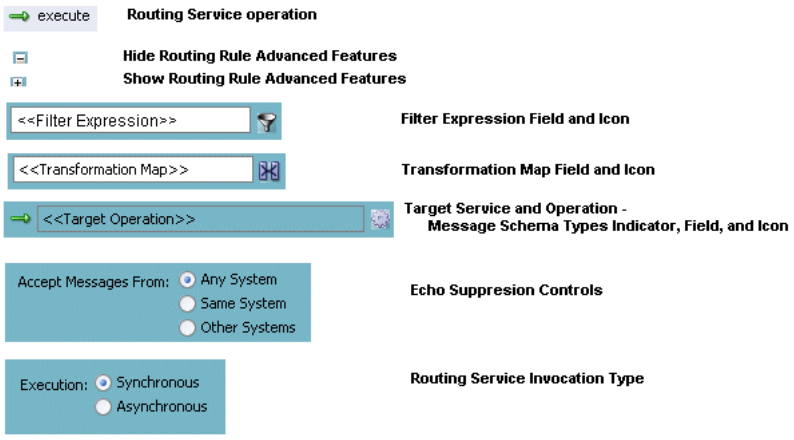


Figure 5–10 describes the icons on the Routing Rules panel.

Figure 5–10 Routing Rules Icons, Fields, and Options



The following sections provide an overview of the routing rules options.

- Target Service and Operation Overview on page 5-9
- Filter Expression Overview on page 5-10
- Transformation Overview on page 5-12
- Accept Messages From Overview on page 5-13
- Routing Invocation Type Overview on page 5-13
- Routing Rule Priority Overview on page 5-13

Note: You must specify the target service and operation for each routing rule before you can specify a transformation. Because the service browser is launched when a new rule is created, the only time you will see a rule without a target is when the target service is deleted.

Target Service and Operation Overview

This routing rule specifies:

- The service where message instances that the routing service receives will be sent. This is referred to as a target service
- The operation that will be executed on the message instances received by the target service.

Later, if you specify a filter expression, you can prevent particular message instances from being sent to a given target service on the basis of each message instance's payload.

For each routing rule, you can specify only one target service and operation. Therefore, if you want to specify multiple target operations for a given target service, you must specify one routing rule for each target service operation.

For example, suppose you specify a database adapter service as a target service and you want the operation applied to a message instance to be one of the following, depending on the message payload:

- Insert
- Update
- Delete

To do so, you create three routing rules, one for each operation. Later, when you specify a filter expression, you can specify which target service and operation is applied to each message instance on the basis of the message payload.

When you click the gear icon to the right of the target operation field (shown in Figure 5–9), the Browse Target Service Operation dialog box opens, such as shown in Figure 5–11.

After you have registered an ESB project, the services are listed under both **Services in project** and **Services at an ESB Connection**. When selecting a target service, you can navigate to the target service and operation at either of the locations as they are the same.

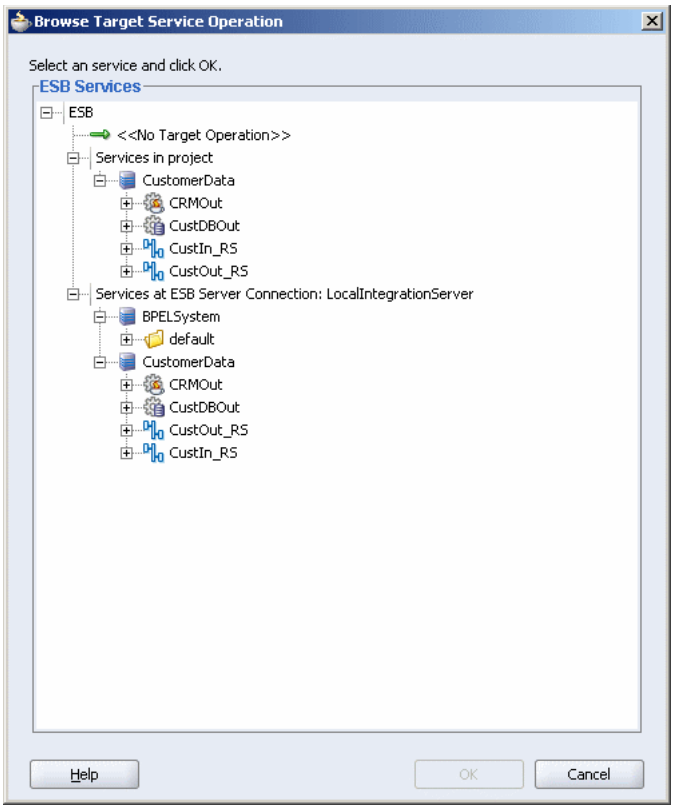
- Services in project

  This branch lists all the ESB systems and the target services and operations within each ESB system in the local Oracle JDeveloper project. In Figure 5–11, for example, you could select target services and operations from the ESB system named DefaultSystem.

- Services at an ESB Server Connection

  This branch lists all the ESB systems and the target services and operations within each ESB system defined at an ESB Server connection. See "Registering ESB Projects and Services with the ESB Server" on page 2-15.

Figure 5–11 Browse Target Service Operation



If, for example, you want to select a target service and operation in the local Oracle JDeveloper project, you expand the desired ESB system under Services in Project, expand the desired service, and then select the desired operation.

For step-by-step instructions on specifying the target service in the routing rules, see "Specifying Routing Rules" on page 5-18.

Filter Expression Overview

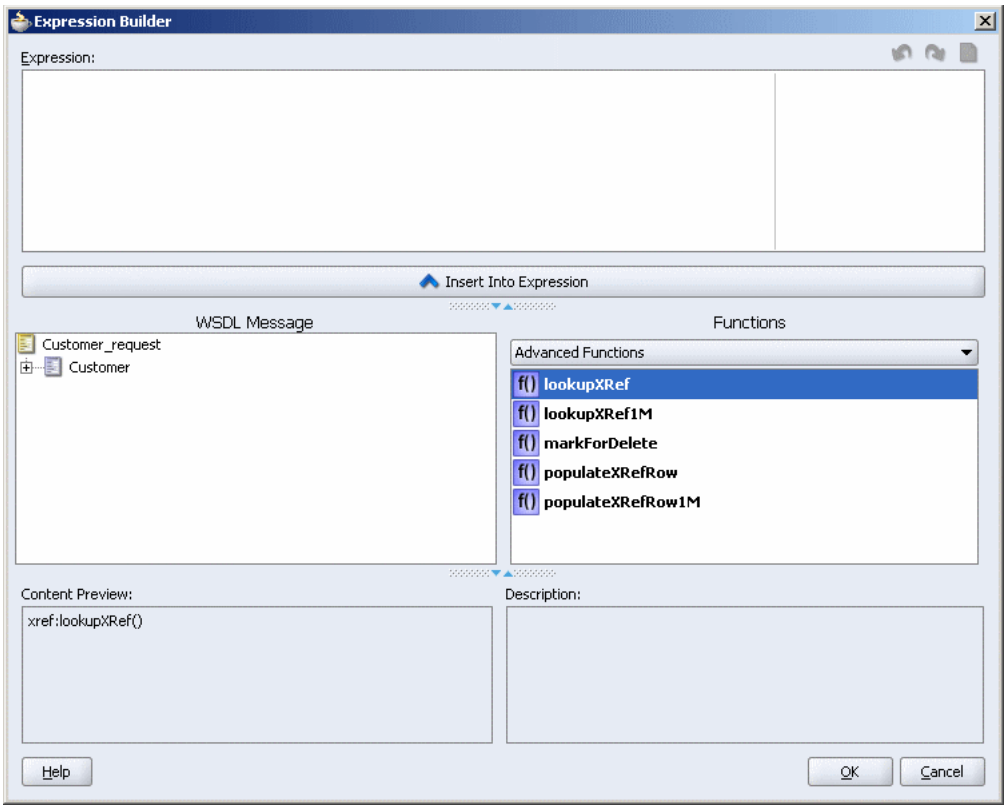
The filter expression routing rule enables you to have the ESB Server filter messages based on their payload. If the expression filter for a given message instance evaluates to true, then the message is not delivered to the target service /operation pair specified within the routing rule.

Suppose, for example, that you want notices of new product launches from headquarters to be routed to three different stores: one in New York, one in Houston, and one in San Francisco. However, you only want notices regarding the product line of type MOBILE to be sent to the New York store.

To do so, you define a routing rule for each service/operation pair that sends messages to the target stores. In addition, for the routing rule (or rules) that send messages to the New York store, you specify a filter expression.

When you click the icon to the right of the filter expression field, shown in Figure 5–9, the Expression Builder window opens, as shown in Figure 5–12.

Figure 5–12 Expression Builder Window – Initial View



The Expression Builder Window contains the components and controls that assist you in designing a filter expression. Briefly, you double-click a value in the WSDL Message navigation tree or the Functions palette, to add the value to the Expression field. Using a combination of WSDL Message elements, functions, and manually entered text, you use this window to build an expression by which you want message payloads to be filtered for a given routing rule.

The following list describes each of the fields in the Expression Builder window:

- Expression field

  This field is where you enter the filter expression – either manually, or by using the WSDL Message navigation tree and the Functions palette.

  The icons that appear to the upper right of this field enable you to validate the expression, undo the last edit made, redo the last edit made, or clear the entire Expression field, respectively.

- WSDL Message Navigation Tree

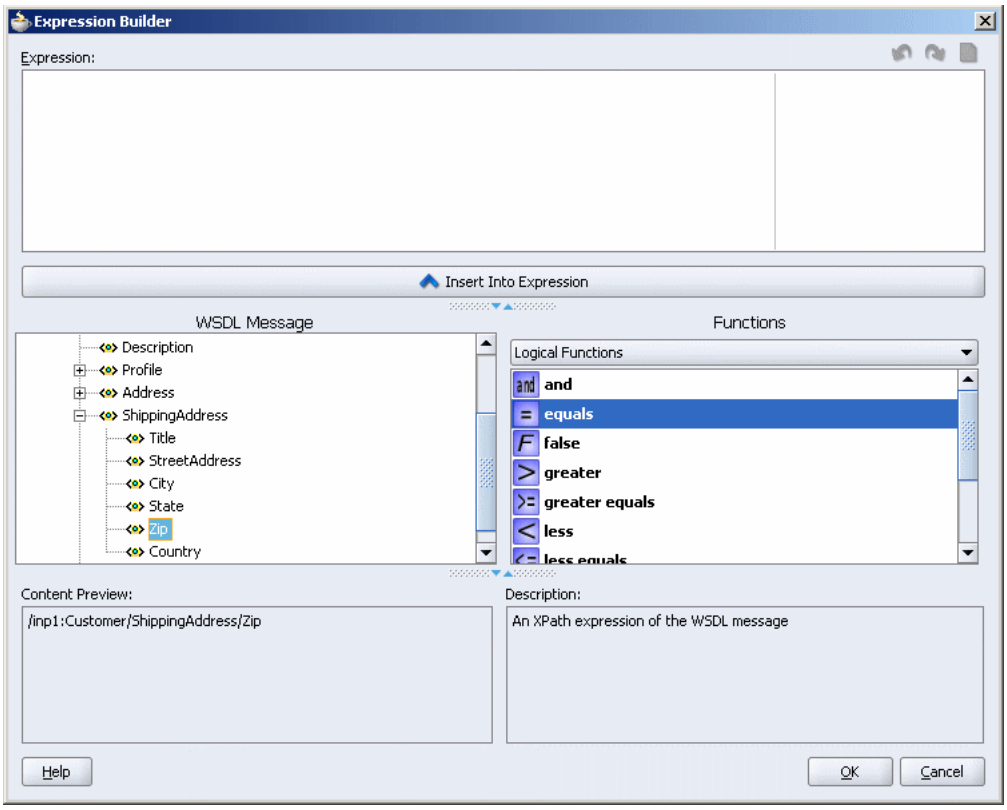
  This field contains the WSDL for the message defined for the routing service. When you select an element from the navigation tree, it is presented in the Content Preview field and described in the Description field. For example, Figure 5–18 on page 5-22 shows how the Expression Builder window appears when you select an element and click Insert into Expression.

- Functions Palette

  This list enables you to select different functions to include in an expression. When you select a function, a preview of how that function will appear when added to

the Expression field is presented in the Content Preview field, and a description of the function is presented in the Description field, as shown in Figure .

Expression Builder Window – Function Preview and Description



- Content Preview

  This field indicates how a value selected from the WSDL Message or Functions palette will appear when it is inserted into the Expression field.

- Description

  This field provides a description of a value selected from the WSDL Message or Functions palette.

See "Using An Expression for Filtering Messages Based on Payload" on page 5-20 for step-by-step instructions on building a filter expression.

Transformation Overview

A transformation map enables you to transform data from one XML schema to another, enabling data interchange among applications using different schemas.

For example, suppose an inbound adapter service picks up a comma-delimited file from a file system. A routing service receives that message and, using a transformation, changes the file structure to the table structure as required by an outbound database adapter service.

When you click the transformation map icon to the right of the Transformation Map field in the routing rules panel of the routing service properties page, the Request Transformation Map dialog displays. In that dialog, you can select an existing mapper file (xsl) or create a new xsl file with the Data Mapper tool to perform the required

transformation. For information about the Data Mapper tool, see Chapter 6, "XSLT Data Mapper and Transformations".

Accept Messages From Overview

You can specify the ESB systems from which the target service will accept messages. The options are any system, the local system only, or systems other than the one in which the target service is defined. See "Specifying the ESB Systems From which Messages are Accepted" on page 5-26.

Routing Invocation Type Overview

You can choose to execute the message instance synchronously or asynchronously in the routing rules. Synchronous messaging provides an immediate response. Asynchronous messaging is useful for environments in which a service can take a long time to process a client request. Asynchronous services also provide a more reliable fault-tolerant and scalable architecture than synchronous services. See "Specifying Synchronous or Asynchronous Execution" on page 5-27.

Routing Rule Priority Overview

You can choose the priority of the routing rules of a routing service by ordering the rule in the ascending and descending order. The priority determines the order in which the rules are applied during message processing. See "Specifying Routing Rules Priority" on page 5-27.

Creating and Modifying Routing Services

You create routing services using Oracle JDeveloper, but you can create and modify routing rules using Oracle JDeveloper or the Oracle ESB Control.

This section contains the following topics:

- Opening the Create Routing Service Dialog on page 5-13
- Specifying the WSDL File for a Routing Service on page 5-14
- Specifying Routing Rules on page 5-18
- Header Transformation and Filtering on page 5-27
- Modifying a Routing Service on page 5-33
- Deleting a Routing Service on page 5-33

Opening the Create Routing Service Dialog

To create a routing service, you complete the Create Routing Service dialog box available in Oracle JDeveloper. You can access this dialog box using either of the following two methods:

- Schematically, using the Design tab, by following these steps

  1. In the Applications navigator, navigate to the ESB project for which you want to create a routing service, expand the **Resources** folder and double click *project-name*.esb, where *project-name* is the name of the project to which you want to add the routing service.

     The Design tab for the project is displayed.

  2. In the **Component Palette**, click the down arrow and select ESB services if they are not already selected.

3. Drag and drop **Routing Service** into the **Design** tab.

   The Create Routing Services dialog box opens.

- Using dialog boxes only, by following these steps:

  To create the routing service using dialog boxes only, follow these steps:

  a. In the Applications navigator, right-click the ESB project for which you want to create a routing service, and then click **New.**

     The New Gallery dialog box opens.

  b. In the **Categories** panel, expand **Business Tier,** and then click **Web Services.**

  c. In the **Items** panel, click **ESB Routing Service,** and then click **OK.**

     The Create Routing Service dialog box opens.

Specifying the WSDL File for a Routing Service

The Create Routing Service dialog box provides three methods for specifying the WSDL for the routing service, as described in the following sections. Each of these sections provides step-by-step instructions on completing the Create Routing Service dialog box for the method you want to use:

- Generating the WSDL for a Routing Service from an Existing XSD File on page 5-14
- Generating the WSDL to Create a Routing Service Based on a Sample File on page 5-16
- Selecting an Existing WSDL to Create a Routing Service on page 5-17

Generating the WSDL for a Routing Service from an Existing XSD File

Use this method to provide an existing WSDL for the routing service. After you specify the file, Oracle JDeveloper parses it to determine the defined schema elements and presents them in a drop-down list from which you can make a selection.

1. If you have not already done so, open the Routing Service dialog box, as described in "Creating and Modifying Routing Services" on page 5-13.
2. In the **Name** field, enter a name for the routing service.

   The name must be unique within the scope of the project in which the routing service is being created. Spaces are not allowed.
3. For the **System/Group** field, click **Browse** to open the ESB Service Group Browser dialog box, select the system (and service group, if desired) to which you want to add the routing service, and then click **OK.**

   To create a new system or service group to contain the routing service you are creating, click **Create New** at the top of the ESB Service Group Browser dialog box. See "Creating ESB Systems and Service Groups" on page 2-12 for information about creating a new ESB system or service group.
4. In the **Description** field, enter a description for the routing service, if desired. This field is optional.
5. Choose **Generate WSDL from Schemas.**

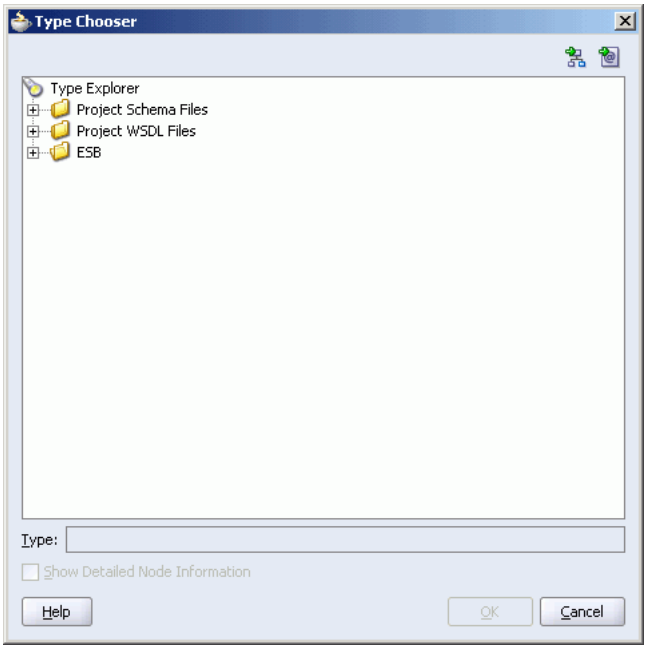
   This option includes the **Request**, **Reply**, and **Fault** tabs.
6. On the **Request** tab, click **Browse** to access the **Schema Location**.

The Type Chooser dialog box, shown in Figure 5–13, opens and presents the schema files (XSD files) from which you can choose to generate the WSDL. Expand the trees under **Project Schema Files**, **Project WSDL Files**, and **ESB** to locate the schema. Navigate to the root element of the XSD file for the message instance that you want this routing service to process. Select the element and click OK.

Note: If you want to use a schema XSD file that resides on your local file system, ensure that the XSD file, and any XSD files that it imports, all reside in the JDeveloper project directory.

You can import a schema XSD file or WSDL file into a project by clicking on the Import Schema File or Import WSDL icon that appears at the top right of the Type Chooser dialog box. For information about importing schemas into an ESB project, see "Importing Files into a Project" on page 2-14.

Figure 5–13 Type Chooser Dialog

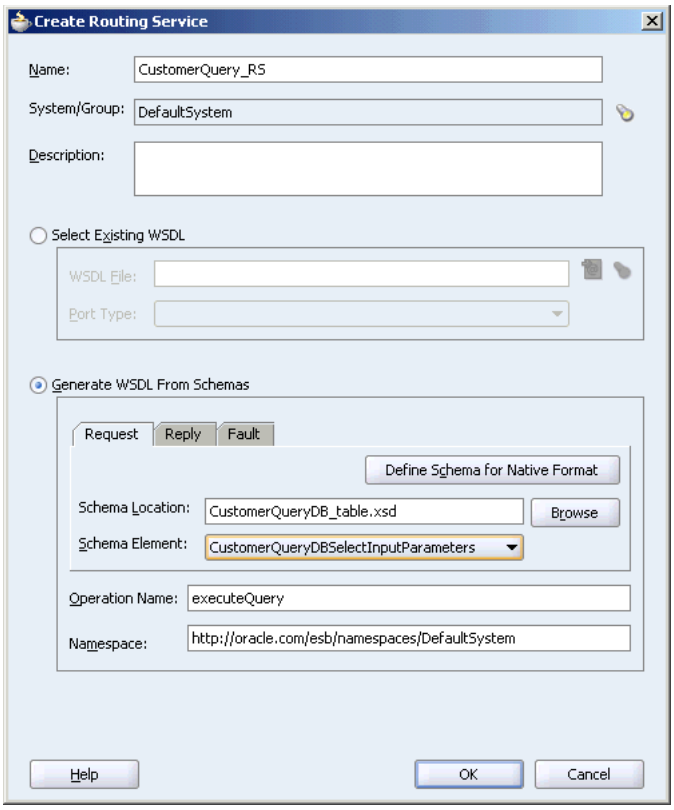


7. In the **Request** tab **Schema Element** field, select the root element for the message that you want this routing service to process if not already selected.
8. On the **Reply** tab, repeat the steps for the **Request** tab if entering any information.
9. On the **Fault** tab, repeat the steps for the **Request** tab if entering any information.
10. In the **Operation Name** field, enter the operation name. Spaces are not allowed.

    For example: **executeQuery**
11. In the **Namespace** field, enter a namespace or accept the current value.

    For example: **http://oracle.com/esb/namespaces/DefaultSystem**
12. Click **OK.**

Figure 5–14 Create Routing Service - Request Tab



The routing service is created and an icon is added to the Design tab of the ESB project.

Generating the WSDL to Create a Routing Service Based on a Sample File

Oracle JDeveloper provides a wizard that assists you in creating the XSD representation of various file formats (such as CSV file, fixed-length file, DTD, and Cobol copybooks) based on a sample file and details that you provide about the file's structure. You can then direct Oracle JDeveloper to generate the WSDL for the routing service from that XSD file.

To use this method, follow these steps in the Create Routing Service dialog box:

1. If you have not already done so, open the Routing Service dialog box, as described in "Creating and Modifying Routing Services" on page 5-13.
2. In the **Service Name** field, enter a name for the routing service.

   The name must be unique within the scope of the project in which the routing service is being created. Spaces are not allowed.
3. For the **System/Group** field, click **Browse** to open the ESB Service Group Browser dialog box, select the system (and service group, if desired) to which you want to add the routing service, and then click **OK.**

   Click **Help** for assistance in using the ESB Service Group Browser dialog box.
4. In the **Description** field, enter a description for the routing service, if desired. This field is optional.
5. Choose **Generate WSDL from Schema**.

This option includes the **Request**, **Reply**, and **Fault** tabs.

6. On the **Request** tab, click **Define Schema for Native Format.**

   The Native Format File Builder wizard opens.

7. Follow the steps through the wizard.

   If you need assistance on a wizard page, click **Help.**

8. In the **Schema Element** field on the **Request** tab, select the root element for the message that you want this routing service to process.

9. **Reply** tab

10. **Fault** tab

11. **Operation Name**

    Spaces are not allowed in the operation name.

12. **Namespace**

13. Click **OK.**

The routing service is created and an icon is added to the Design tab of the ESB project.

Selecting an Existing WSDL to Create a Routing Service

If you use this method to provide the WSDL for the routing service, the existing WSDL must exist on the local file system. After you specify the file, Oracle JDeveloper parses it to determine the defined port types and presents them in a drop-down list from which you can make a selection.

To use this method, follow these steps in the Create Routing Service dialog box:

1. If you have not already done so, open the Routing Service dialog box, as described in "Creating and Modifying Routing Services" on page 5-13.

   In the **Service Name** field, enter a name for the routing service.

   The name must be unique within the scope of the project in which the routing service is being created. Spaces are not allowed.

2. For the **System/Group** field, click **Browse** to open the ESB Service Group Browser dialog box, select the system (and service group, if desired) to which you want to add the routing service, and then click **OK.**

   Click **Help** for assistance in using the ESB Service Group Browser dialog box.

3. In the **Description** field, enter a description for the routing service, if desired. This field is optional.

4. Choose **Select Existing WSDL.**

5. To the right of the **WSDL Location** field, click the icon.

   The **Open** dialog box opens.

6. In the Open dialog box, navigate to the existing WSDL file, and then click **OK**. (If you need assistance with this dialog box, click **Help.**)

7. In the **Port Type** field, click the **down arrow**, and then select the port type for the routing service.

8. Click **OK.**

After you complete the process for creating a routing service, an icon for the service appears in the Design tab of the project. For an example, see Figure 5–1. In the Applications Navigator, files with `esbsvc` and `wsdl` extensions are created in the Resources folder of the project. The `esbsvc` file provides the definition of the ESB service. The WSDL file defines the input and output messages for this instance flow, the supported client interface and operations, and other features.

Specifying Routing Rules

After you define a routing service, by specifying its WSDL, you can specify the rules that determine how a message processed by the routing service gets to its next destination. Routing rules can be defined using a property sheet in Oracle JDeveloper or a property sheet in the Oracle ESB Control.

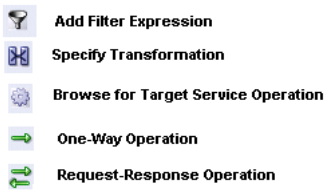
To access the Routing Rules property sheet inOracle JDeveloper, use either of the following two methods. For information about accessing the Routing Rules property sheet in the Oracle ESB Control, see "Viewing Routing Rules" on page 3-15.

- From the Applications Navigator:
  - **a.** In the **Applications Navigator,** expand the ESB project, followed by the **Resources** folder.
  - **b.** In the **Resources** folder, double click the name of the routing service for which you want to specify routing rules.
  - **c.** Click the plus symbol (+) to expand the **Routing Rules** information.
- From the Design tab:
  - **a.** Double-click the icon that represents the routing service for which you want to specify routing rules.
  - **b.** Click the plus symbol (+) to expand the **Routing Rules** information.

The Routing Rules property sheet is displayed, as shown in Figure 5–8 on page 5-7.

The icons on the Routing Rules property sheet are summarized in Figure 5–15.

Figure 5–15 Routing Rule Property Sheet Icons



The following sections describe each of the available routing rules and how to specify them:

- Specifying the Target Operations on page 5-19
- Creating an XSL Map File for Data Structure Transformation on page 5-20
- Using An Expression for Filtering Messages Based on Payload on page 5-20
- Specifying the ESB Systems From which Messages are Accepted on page 5-26
- Specifying Synchronous or Asynchronous Execution on page 5-27
- Specifying Routing Rules Priority on page 5-27

Specifying the Target Operations

The target operation is the only routing rule you must specify to make use of a routing service. This routing rule tells the routing service the next service, known as the target service, to which the message should be sent and the operation to perform on that message when it reaches the target service.

You can specify multiple target service and target operation pairs for each routing service. In addition, for request/reply message flows, you can forward the reply message to another target, and specify a target service in the event that a message fault occurs. Note that a fault is simply a special reply that can be directed to the requester.

You can specify the following target operations for routing rules:

- Specifying the Target Operation for a One-Way Configuration
- Specifying the Target Operation to Return a Reply to the Source Service
- Specifying the Target Operation to Forward a Reply to a Non-Source Service
- Specifying the Target Operation for a Faulted Message

> **Note:** WSDLs with multiple faults are not supported.

Follow these steps to specify a target service and operation:

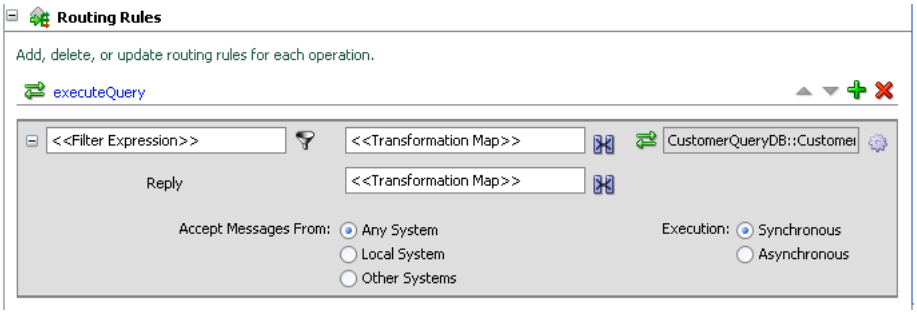
1. If you have not already done so, expand the Routing Rules information, as described in "Specifying Routing Rules" on page 5-18.
2. Click the large green **plus** (+) button.

   The Browse Target Service Operation dialog box opens. For information about the target service and operation, see "Target Service and Operation Overview" on page 5-9.
3. In the Browse Target Service Operation dialog, follow these steps:
   a. Navigate to, and then expand the desired target service.
   b. Select the target service operation in the Browse Target Service Operation dialog. See Figure 5–11 on page 5-10.
   c. Click **OK.**

      The Routing Rules panel is updated to reflect the newly added target service and operation, with the value expressed as *target_service.target_operation*, as shown in Figure 5–16. Depending on the target selected, the rule includes a combination of Request, Reply, or Fault fields.
4. Repeat steps two and three if you want to add an additional target service and target operation pair. You can specify the same target service and a different operation, if desired.

> **Note:** To modify the target service and target service operation, click the Browse for Target Service Operation icon, as shown in Figure 5–15.

Figure 5–16 Routing Rules for Request/Reply with Target Service Specified



Creating an XSL Map File for Data Structure Transformation

Oracle JDeveloper provides an XSLT Data Mapper tool that enables you to specify an `xsl` file to transform data from one XML schema to another. This enables data interchange among applications using different schemas. For example, you can map incoming source purchase order schema to an outgoing invoice schema. After you define an xsl file, you can reuse it in multiple routing rule specifications.

See "Creating an XSL Map with Data Mapper" on page 6-3.

Using An Expression for Filtering Messages Based on Payload

You can specify an expression to filter messages based on their payload. You can, for example, route messages for a customer record to different offices, based on that customer's postal code.

To specify a filtering expression, follow these steps:

1. If you have not already done so, expand the Routing Rules information, as described in "Specifying Routing Rules" on page 5-18.
2. Click the **Add Filter Expression** icon, as shown in Figure 5–15.

   The Expression Builder opens.
3. Specify the filter expression, and then click **OK.**

   Information about using the Expression Builder is described in the text following these steps.

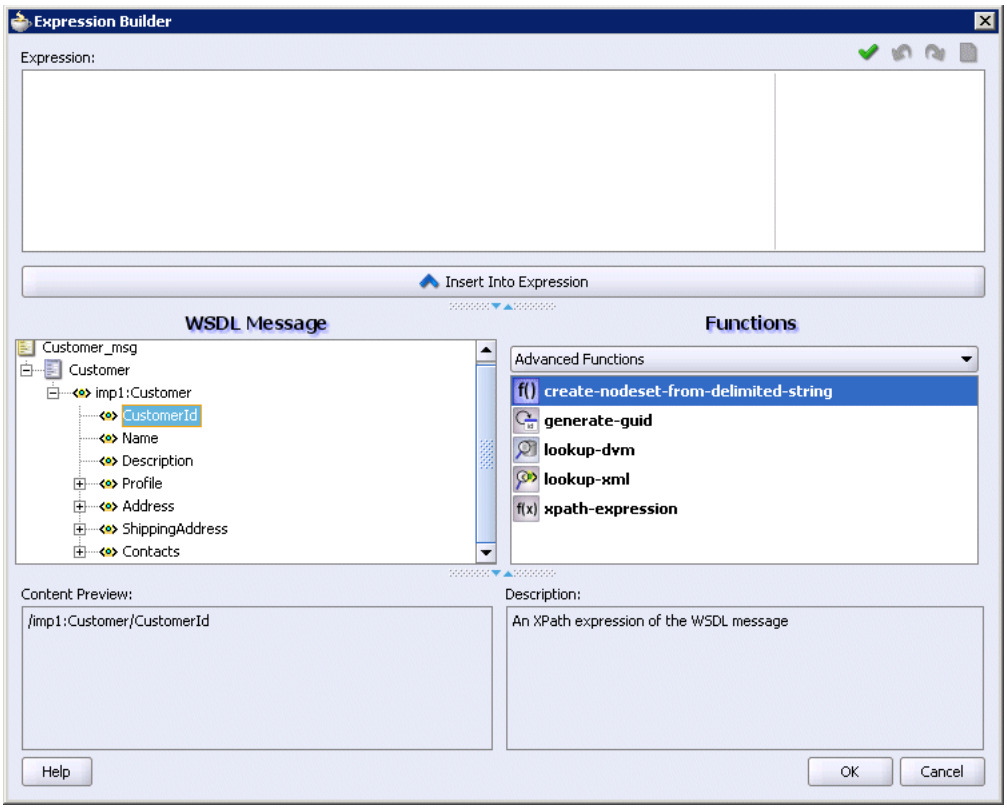
When the Expression Builder opens, it appears similar to Figure on page 5-12. Notice that Oracle JDeveloper parses the routing service WSDL and presents the message definition in the WSDL Message box.

You build the expression for filtering as follows:

1. In the WSDL Message box, expand the message definition and select the message element on which you want to base the expression.

   Notice that the Content Preview box indicates the XPath expression for the selected WSDL message element, an example of which is shown in Figure 5–17.

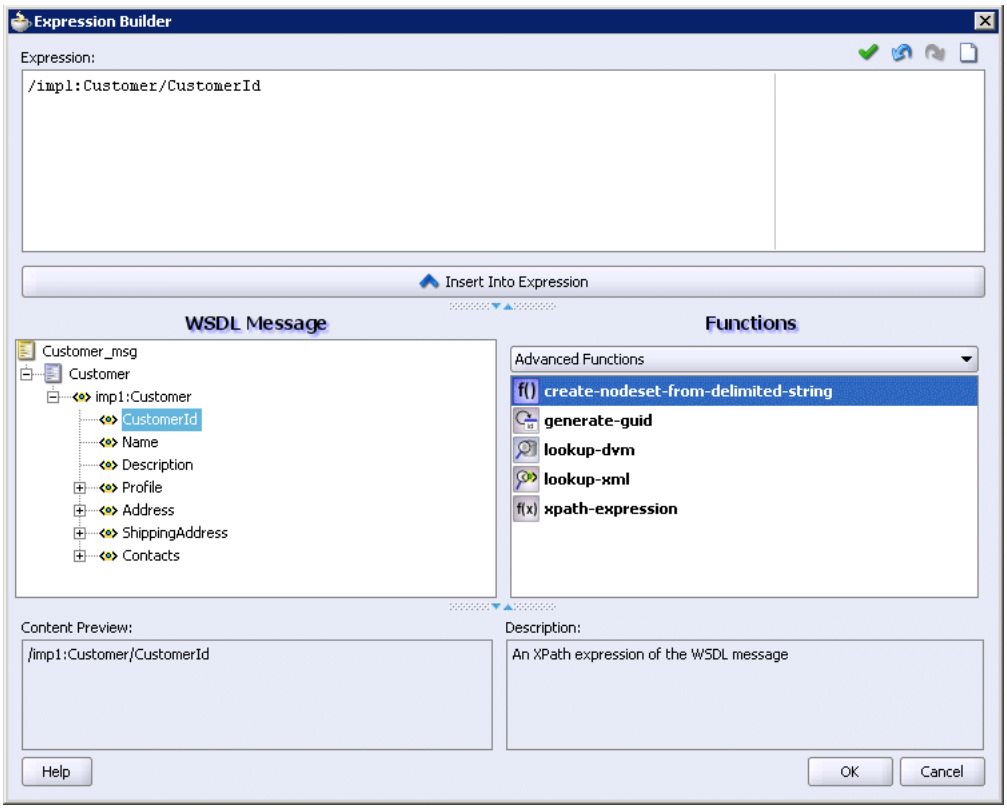
Figure 5–17 Sample Expression Builder Tool – WSDL Message Element Selected



2. Click **Insert Into Expression.**

   The expression is presented in the Expression box, as shown in Figure 5–18.

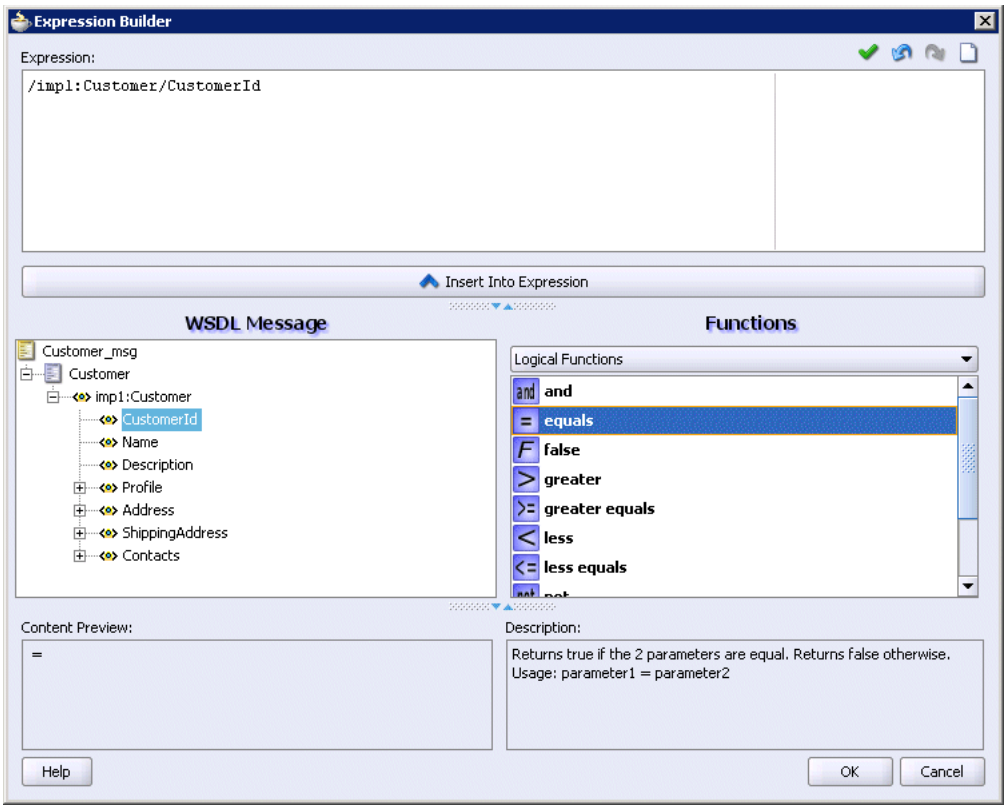
Figure 5–18 Sample Expression Builder Tool – WSDL Message Element Inserted



3. From the Function box, select the function that you want to apply to the WSDL Message payload.

   Functions are listed within categories that are listed when you click the down arrow within the Functions box. For example, if you click the down arrow and select Logical Functions, the list appears as shown in Figure 5–19. Notice that if you select a function within the Logical Functions list, a description of that function is presented in the Explanation box.

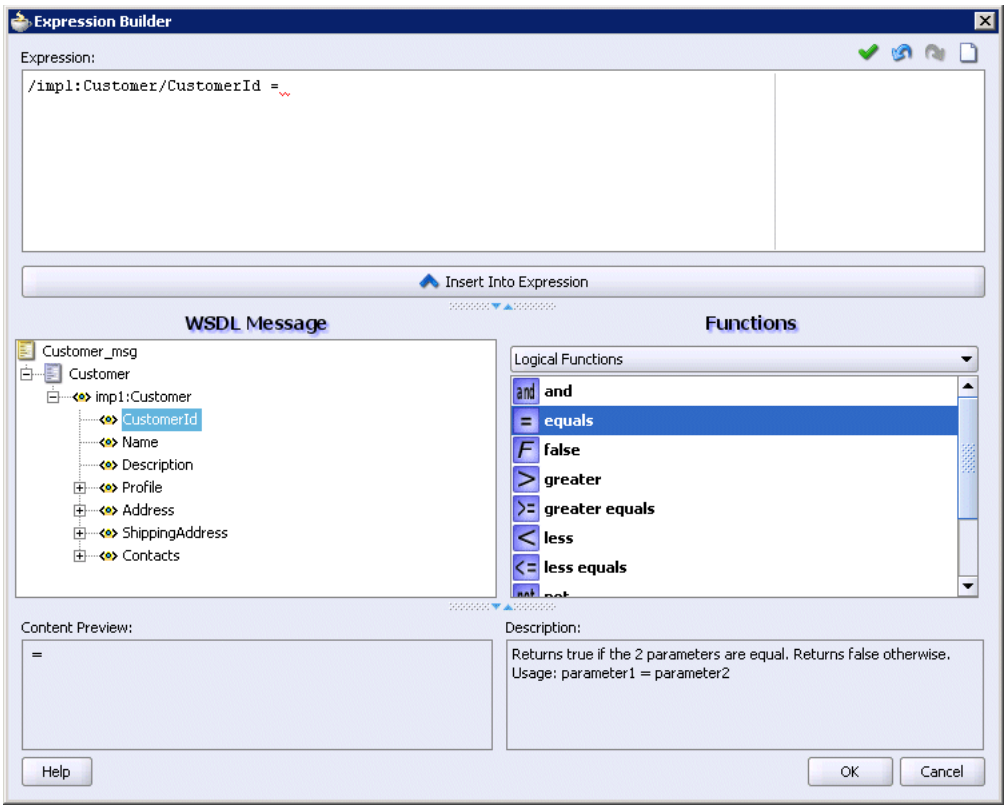
Figure 5–19 Sample Expression Builder Tool – Function Selected



4. Click **Insert Into Expression.**

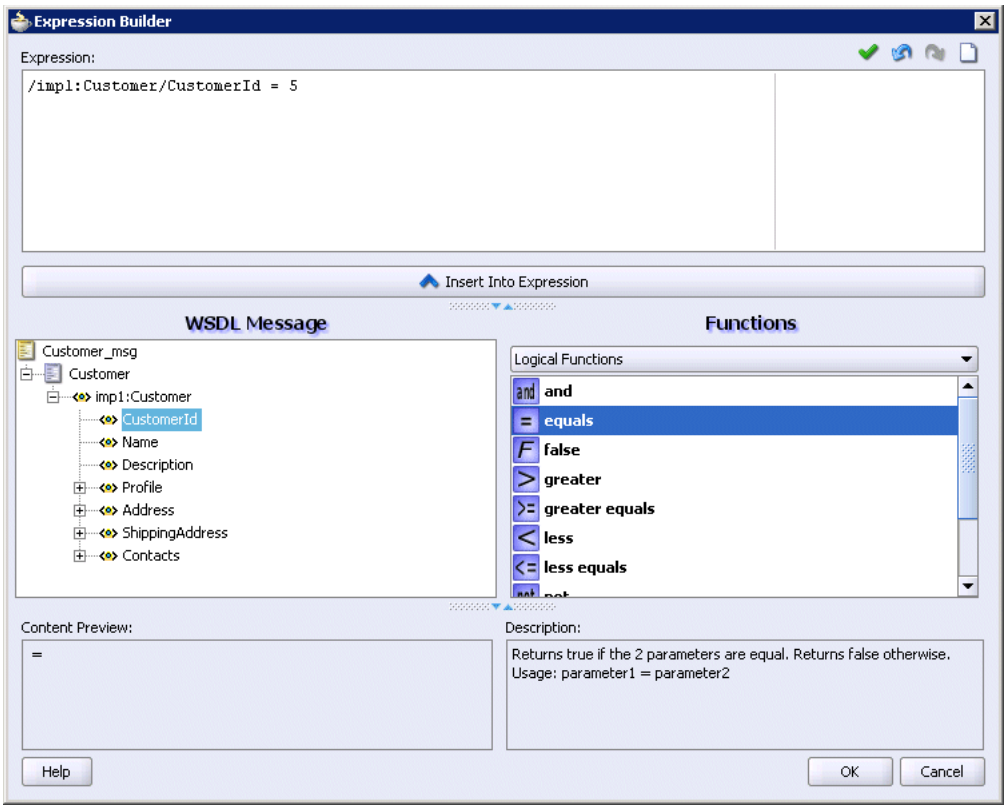
   The XPath expression for the selected function is inserted in to the Expression box. Notice that because the expression requires editing by hand to complete the expression, a red squiggle appears at the end of the line, as shown in Figure 5–20.

Figure 5–20 Sample Expression Builder Tool – Function Inserted



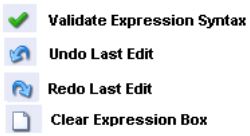
5. Complete the expression. In this example, a value of 5 is entered, as shown in Figure 5–21.

Figure 5–21 Sample Expression Builder Tool – Value Entered



6. Validate the expression by clicking the green check mark.
7. If the expression is invalid or you need to make a change, you can edit the expression manually, or use the expression editing icons, which are summarized in Figure 5–22.

Figure 5–22 Expression Editing Icons



8. Click **OK.**

   The expression is added to the Routing Rule property sheet.

Note that in addition to the XPath expression generated by the Expression Builder, namespace definitions are automatically provided by Oracle Enterprise Service Bus to create an extended syntax. If you place the cursor on a non-empty filter icon of a routing service in the Design tab, the extended syntax displays.

To modify or delete a filter expression, double-click the Add Filter Expression icon, and then modify or delete the expression in the Expression panel of the Expression Builder.

You can also use user-defined extension functions in the Expression builder dialog box. Perform the following steps to use a user-defined extension function:

1. Write a function that implements the `javax.xml.xpath.XPathFunction` interface. A sample function is shown in the following example:

```
public class ExtnFunc implements XPathFunction
{
           public Object evaluate(List args) throws XPathFunctionException
          {
              return "helloworld";
          }
}
```

2. Modify the `extn_xpath_functions_config.xml` file in the *ORACLE_HOME*`/integration/esb/config` folder to add the user-defined extension function details.

   A sample `extn_xpath_functions_config.xml` is shown in the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ==================================================================== -->
<!--  XPath extension functions available to ESB 10.1.3 -->
<!-- ==================================================================== -->
<esb-xpath-functions xmlns="http://xmlns.oracle.com/soa/config/esb"
version="10.1.3.1">
  <!-- function id="function id which will be used in xpath" arity="1">
        <classname>your class name including package</classname>
          <namespace-uri>
             your namespace
          </namespace-uri>
  </function -->
</esb-xpath-functions>
```

3. You can use the user-defined extension function to build an expression in the Expression builder dialog box in following way:

   NamespaceName:*ExtensionFunctionName*

   The *NameSpace* refers to the namespace defined in the `extn_xpath_functions_config.xml` file. The *ExtensionFunctionName* refers to the extension function. For example,

```
{ns1:MyExtensionFunction= 'US'};
```

Specifying the ESB Systems From which Messages are Accepted

To indicate the ESB systems from which the routing service will accept messages, follow these steps:

1. If you have not already done so, open the Routing Rules properties, as described in "Specifying Routing Rules" on page 5-18.
2. Choose one of the following options:
   - Any System

     Choose this option to specify that the routing service accept messages that originate from any ESB system, including the one in which the routing service was created.
   - Same System

Choose this option to specify that the routing service accept messages that originate only from the same ESB system as the one in which the routing service was created.

- Other Systems

  Choose this option to specify that the routing service accept messages that originate only from ESB systems other than the one in which the routing service was created.

Specifying Synchronous or Asynchronous Execution

Synchronous execution provides an immediate response to a request; asynchronous does not. Asynchronous execution is useful for environments in which a service can take a long time to process a request. Asynchronous services also provide a more reliable fault-tolerant and scalable architecture than synchronous services.

To indicate whether the routing service invokes the target service as synchronous or asynchronous execution, follow these steps:

1. If you have not already done so, open the Routing Rules properties, as described in "Specifying Routing Rules" on page 5-18.
2. Click the **plus** (+) button to expand the routing rules information.

   The **Execution** options display on the right side of the property sheet.
3. Specify **Synchronous** or **Asynchronous** execution with the appropriate button. For an illustration of the property sheet, see Figure 5–16 on page 5-20.

Specifying Routing Rules Priority

You can choose the priority of multiple routing rules of a routing service by placing the rules in ascending order, with the top rules having the highest priority.

After you have selected a rule by clicking on the + on the left of the rule, you can click the up or down triangles in the upper right of the routing rules panel to move the selected rule to order of the correct priority. For an illustration of the property sheet, see Figure 5–16 on page 5-20.

Header Transformation and Filtering

Oracle Enterprise Service Bus provides limited support for header-based transformation and filtering (routing).

This section includes the following topics:

- Header Support Terminology on page 5-27
- Header-based Transformation on page 5-28
- Header-Based Filtering on page 5-31
- Limitations of ESB Header Support on page 5-32

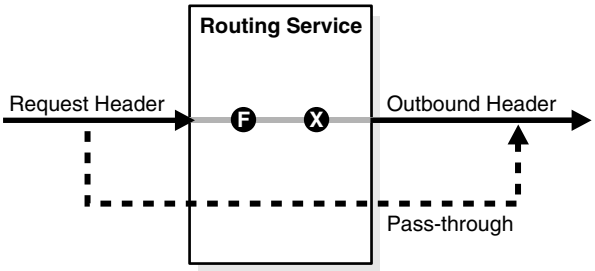
Header Support Terminology

In ESB, the headers are referred to as follows:

- A request header goes to a service and leaves as an outbound header.

  Filtering and transformation can be used with the request header. If a transformation is not specified, the header is copied as-is (pass-through).

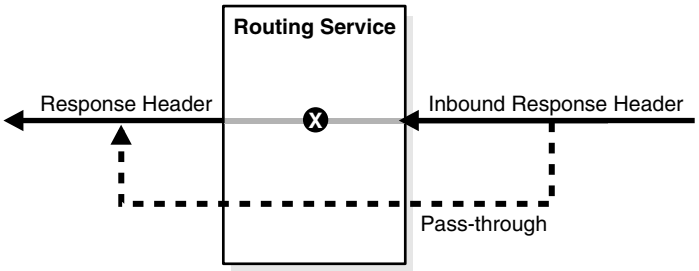
Figure 5–23 Request and Outbound Headers



- An inbound response header goes to an outbound routing service and leaves as a response header.

  Transformation can be used with the response header. If a transformation is not specified, the header is copied as-is (pass-through).

Figure 5–24 Response and Inbound Response Headers



For information about the adapter inbound and outbound header WSDL files, see *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide*.

Header-based Transformation

ESB header-based transformation is supported with XSLT extension functions for:

- Reading request headers:

```
String getRequestHeader(String xpathExpression,
                        String namespaceDecl)
```

- Writing outbound headers:

```
void setOutboundHeader(String xpathExpression,
                       String value,
                       String namespaceDecl)
```

- Reading inbound response headers to a routing service:

```
String getInboundResponseHeader(String xpathExpression,
                                String namespaceDecl)
```

- Writing response headers:

```
void setResponseHeader(String xpathExpression,
                       String value,
                       String namespaceDecl)
```

The following applies to the syntax of the XSLT function declarations:

- `xpathExpression` - the XPath expression to get or set in the header
- `value` - the value to be set for the `xpathExpression`
- `namespaceDecl` - namespace declarations in the form `'prefix=namespace;'`

  Note the semi-colon (`;`) at the end of the namespace declaration.

The namespace for the XSLT extension functions is:
`http://www.oracle.com/XSL/Transform/java/oracle.tip.esb.server.headers.ESBHeaderFunctions`

To set up a header-based transformation, follow these steps:

1. Create a data transformation with the XSLT Data Mapper tool if one does not already exist.
2. Right click **<target>** in the data map, then select **Add Variable** from the menu to display the Create Variable dialog, shown in Figure 5–25.
3. In the Create Variable dialog, complete the fields for each variable as follows:
   - **Local Name**:

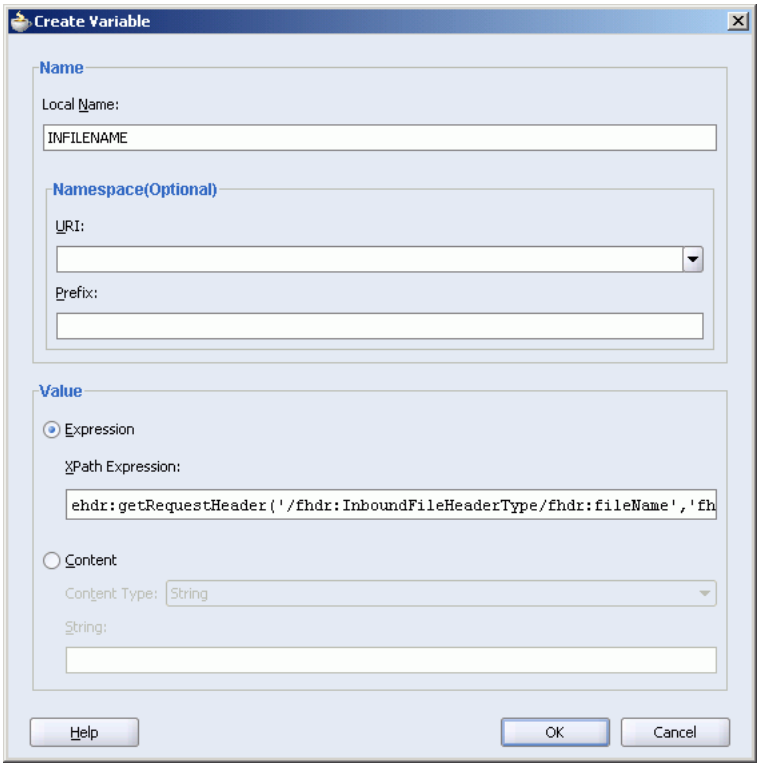
     Enter a name for the variable
   - **XPath Expression**:

     Enter the expression using one of the header transformation functions.

     To display the header-based transformation functions, you can enter `ehdr:` and then **Ctrl+Spacebar** in the field. Double-click the function that you want to add to the expression, then add the required parameters to complete the function.

     When you are entering the XPath expression for an adapter, you need to check the source of the adapter header schema definition for values of the parameters, shown in Example 5–1. Oracle JCA adapter headers are defined in a separate WSDL file. For information about the adapter inbound and outbound header WSDL files, see *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide*.

Figure 5–25 Create Variable Dialog With a Header Transformation Function



Example 5–1 is an example of a an adapter schema file.

Example 5–1 Sample fileAdapterInboundHeader.wsdl

```
<definitions
     name="fileAdapter"
     targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/"
     xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/file/"
     xmlns="http://schemas.xmlsoap.org/wsdl/" >

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

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

```
</definitions>
```

For Example 5–1, the corresponding `getRequestHeader` call in the Xpath expression would be in the following form:

```
ehdr:getRequestHeader('/fhdr:InboundFileHeaderType/fhdr:fileName',
                      'fhdr=http://xmlns.oracle.com/pcbpel/adapter/file/;')
```

If you are writing the `fileName` to an outbound file adapter, you could use the following in the XPath expression when creating a variable, where `INFILENAME` is the variable name corresponding to `getRequestHeader` call:

```
ehdr:setOutboundHeader('/fhdr:OutboundFileHeaderType/fhdr:fileName', $INFILENAME,
'fhdr=http://xmlns.oracle.com/pcbpel/adapter/file/;')
```

Note: When using the set header functions, you must set values in the order that is expected by the target service. This means that in Example 5–1, you would need to first set the `fileName` before setting the `directory` element.

Header-Based Filtering

You can route a message based on the message headers using filtering. ESB header-based filtering is supported for **request** and **inbound response headers** with XPath extension functions:

- Reading request headers:

  ```
  String getRequestHeader(String xpathExpression)
  ```

- Reading inbound response headers to a routing service:

  ```
  String getInboundResponseHeader(String xpathExpression)
  ```

The following applies to the syntax of the XPath function declarations:

- `xpathExpression` - the XPath expression to get in the header

The syntax of the filter expression in Expression Builder is:

```
{filterExpression};{namespaceDeclaration}
```

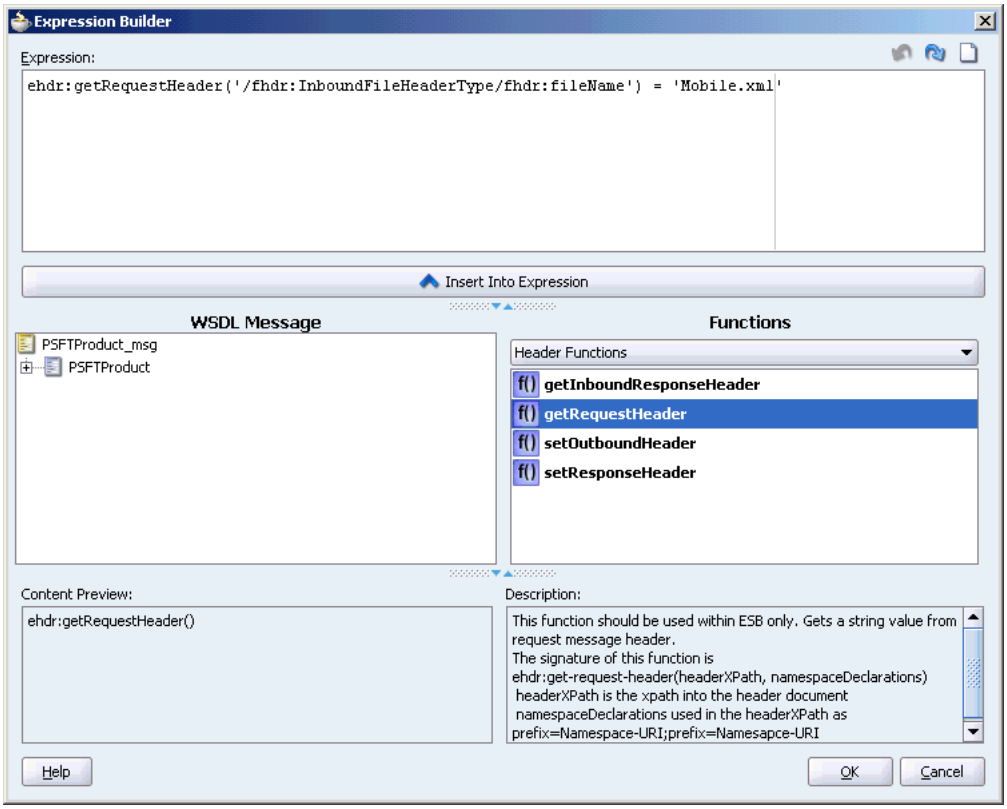
The namespace for the XSLT extension functions is:
`http://www.oracle.com/XSL/Transform/java/oracle.tip.esb.server.headers.ESBHeaderFunctions`

To set up a header-based filtering (routing), follow these steps:

1. Launch the Expression Builder tool from the routing service.
2. Select **Header Functions** under **Functions** to display the XPath extension functions.
3. Select the function you want and click **Insert Into Expression**.
4. Check the source of the adapter schema file for values of the parameters, then edit the function to the correct syntax. For information about the adapter inbound and outbound header WSDL files, see *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide*.

   When creating the expression, it might be easier to edit the expression in a text editor and paste the completed expression into Expression Builder.

Figure 5–26 Expression Builder With a Header Transformation Function



For example, if you want to check if the `fileName` is `payload.xml`, then you use the following in Expression Builder after checking the values in `fileAdapterInboundHeader.wsdl` file shown in Example 5–1.

```
{ehdr:getRequestHeader('/fhdr:InboundFileHeaderType/fhdr:fileName') =
'payload.xml'};
{ namespace fhdr=http://xmlns.oracle.com/pcbpel/adapter/file/ namespace

ehdr=http://www.oracle.com/XSL/Transform/java/oracle.tip.esb.server.headers.ESBHea
derFunctions}
```

> **See Also:** The samples for the Header-Based Filtering are available on the Oracle Enterprise Service Bus page on the Oracle Technology Network site (`http://www.oracle.com/technology/products/integration/esb/index.html`).

Limitations of ESB Header Support

These are the limitations when using ESB header transformation and routing:

- The `setXHeader` functions only support the following types of Xpath expressions:

  Absolute path, such as `/Customer/Address/Zip`.

  Indexed, such as `/Customer/Address[2]/Zip`. If `Address[1]` is not already created in the target document, it is created.

- Multiple headers are not supported.

- When using SOAP with headers, `SOAP Header` elements are set as `header`. Xpath expressions must start with the `Header` element, such as `/**env:Header**/Message/Priority`.
- There is limited GUI tools support.
- Does not support setting attributes.

Modifying a Routing Service

To edit an existing routing service, perform the following steps:

1. Double click the routing service in the **Application Navigator** or in the **Design** tab.
2. Make your changes to the routing service information. Note that the name of the service cannot be changed.
3. Save your changes.

Deleting a Routing Service

To delete a routing service, perform the following steps:

1. Select the routing service icon in the **Design** tab.
2. Click the large red X at the top of Design tab to delete the selected routing service.
3. Confirm that you want to delete the selected service.
4. Save your changes.

Note: Do not delete routing services in the Application Navigator.

6

XSLT Data Mapper and Transformations

This chapter describes features of the XSLT Data Mapper and provides instructions for using a transformation mapping in routing rules.

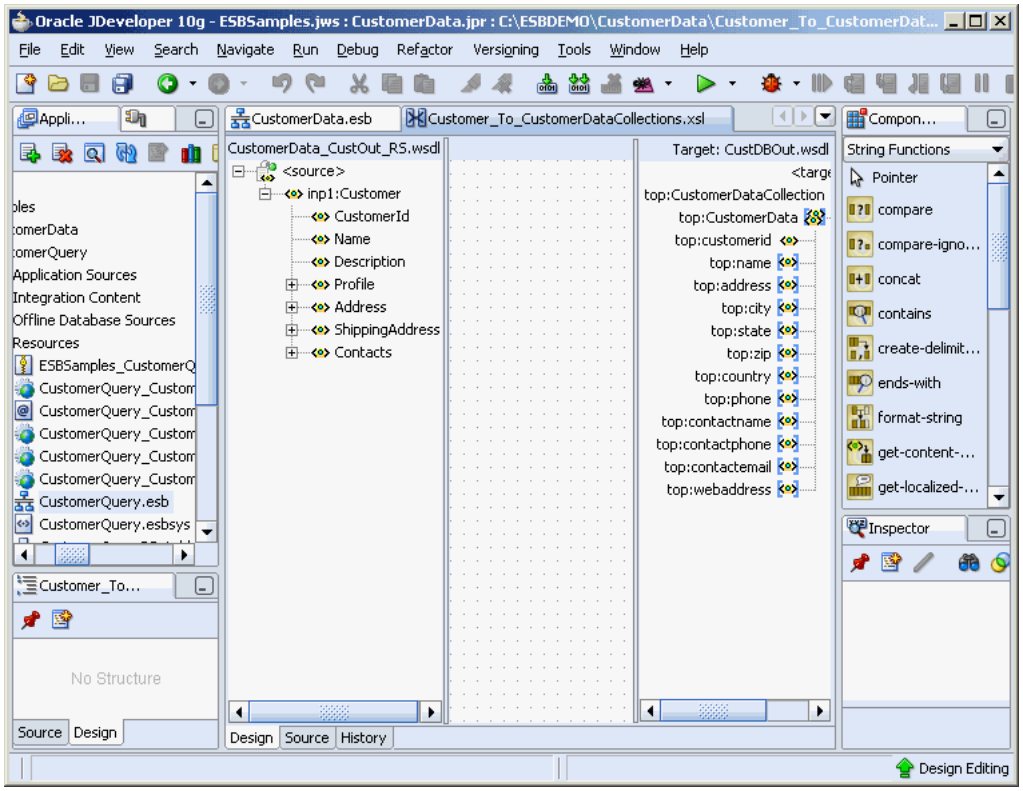
This chapter contains the following topics:

- XSLT Data Mapper on page 6-1
- Creating an XSL Map with Data Mapper on page 6-3
- Using the XSLT Mapper on page 6-6
- Using the Mapper Test Utility on page 6-24

XSLT Data Mapper

The Data Mapper tool provides data transformation in the routing rules of the routing service. You use the XSLT Mapper transformation tool to create the contents of a map file. Figure 6–1 shows the layout of the XSLT Mapper.

Figure 6–1 Sample XSLT Data Mapper Tool



The **Source** and the **Target** schemas are represented as trees and the nodes in the trees are represented using a variety of icons. The displayed icon reflects the schema or property of the node. For example:

- An XSD attribute is denoted with an icon that is different from an XSD element
- An optional element is represented with an icon that is different from a mandatory element
- A repeating element is represented with an icon that is different from a nonrepeating element, and so on

The various properties of the element and attribute are displayed in the **Property Inspector** in the lower right of Figure 6–1 (for example, type, cardinality, and so on). The **Functions Palette** in the upper right of Figure 6–1 is the container for all functions provided by the XSLT Mapper. The mapper pane or canvas is the actual drawing area for dropping functions and connecting them to source and target nodes.

The XSLT Mapper provides three separate context sensitive menus:

- One in the source panel
- One in the target panel
- One in the mapper pane or canvas in the middle

Right-click in each of the three separate panels to see what the context menus look like. A full set of **Undo Auto Map**, **Redo**, **Delete**, and **Delete All** functions are also available.

Notes on the Mapper

- A node in the target tree can be linked only once (that is, you cannot have two links connecting a node in the target tree).
- An incomplete function and expression does not result in an XPath expression in the source view. If you switch from the design view to the source view with one or more incomplete expressions, the Mapper Messages window displays warning messages.
- When you map duplicate elements in the XSLT Mapper, the style sheet becomes invalid and you cannot work in the **Design** view. The **Log Window** shows the following error messages when you map an element with a duplicate name:

  ```
  Error: This Node is Already Mapped  :
  "/ns0:rulebase/for-each/ns0:if/ns0:atom/ns0:rel"
    Error: This Node is Already Mapped  :
  "/ns0:rulebase/for-each/ns0:if/ns0:atom/choice_1/ns0:ind"
    Error: This Node is Already Mapped  :
  "/ns0:rulebase/for-each/ns0:if/ns0:atom/choice_1/ns0:var"
  ```

  The workaround is to give each element a unique name.

Creating an XSL Map with Data Mapper

The XSLT Data Mapper tool enables you to create an `xsl` file to transform data from one XML schema to another. After you define an xsl file, you can reuse it in multiple routing rule specifications. This section provides an overview of creating a transformation map xsl file with the XSLT Data Mapper tool.

The Data Mapper tool is available through the Routing Service property page or from the routing service icon in the Design tab of Oracle JDeveloper. You can either create a new transformation map or update an existing one.

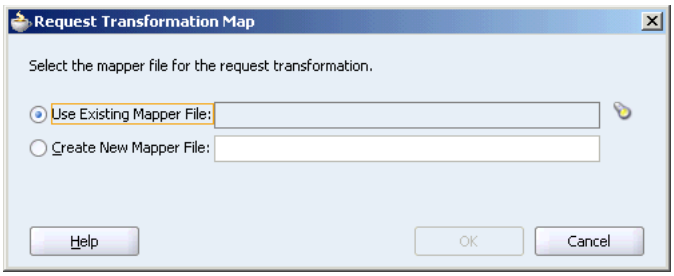
To launch the Data Mapper tool from Routing Service property page and create or update a data transformation xsl file, follow these steps:

1. Open the properties page of a Routing Service.
2. Open the Routing Rules panel by clicking on the **+** to the left of Routing Rules as described in "Specifying Routing Rules" on page 5-18.

   The transformation map icon is visible in the routing rules panel. For an example of the routing service property page with the transformation map icon in the routing rules panel, see Figure 5–8 on page 5-7.
3. Click the transformation map icon to the right of the Transformation Map field in the routing rules panel to open the Request Transformation Map dialog, shown in Figure 5–16 on page 5-20.

   The Request Transformation Map dialog displays with options for selecting an existing transformation map (XSL) file or creating a new map file.

Figure 6–2 Request Transformation Map Dialog

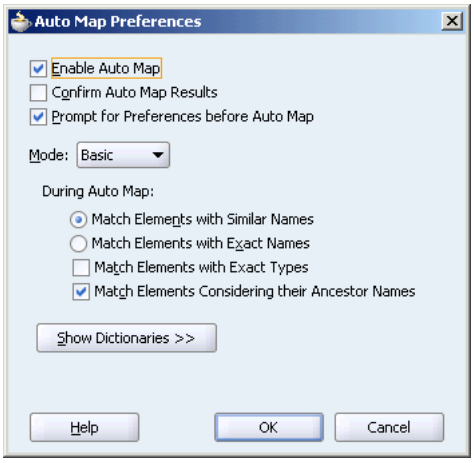


If the routing rule includes a reply or fault, the Request Transformation Map dialog contains the **Include Request in the Reply Payload** option for reply or fault transformations. When you enable this option, you can obtain information from the request message. When you create a new map for reply or fault transformations, the `ESBREQUEST` variable is added on the source side of the Data Mapper tool. When you view the properties in the Edit Parameter dialog for `ESBREQUEST`, you can add XPath Expression functions.

4. Choose one of the following options:
   - **Create New Mapper File** and then enter a name for the file (or accept the default value).
   - **Use Existing Mapper File** and then click the flashlight icon to browse for an existing mapper file (or accept the default value). When the data mapper tool opens, you can update the existing file.
5. Click **OK.**

   If you chose Create New Mapper File, the XSLT Data Mapper tool opens to enable you to correlate source schema elements to target schema elements, as shown in Figure 6–1 on page 6-2.
6. You can select and drag a component on either side of the tool to the component you want to correlate on the other side of the mapper tool. When you initially select and drag, the Auto Map Preferences dialog displays so you can set preferences for the mapping.

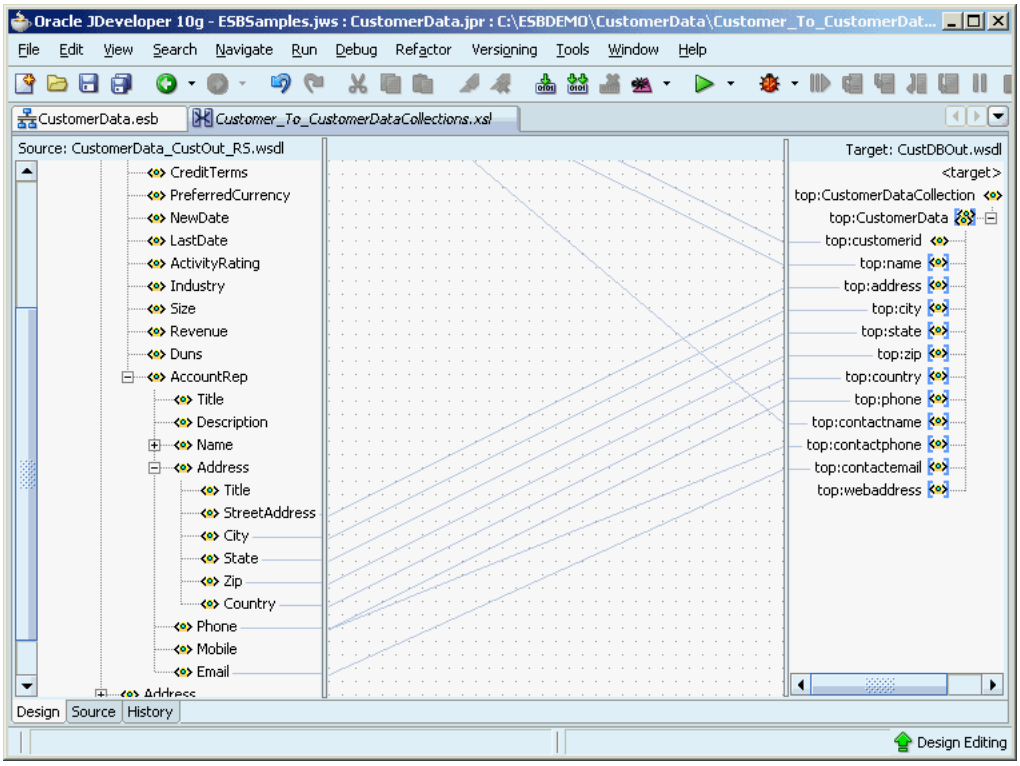
Figure 6–3 Auto Map Preferences Dialog



7. You can choose to drag and drop source elements to target elements individually or you can use the AutoMap feature in the Auto Map Preferences dialog. If you

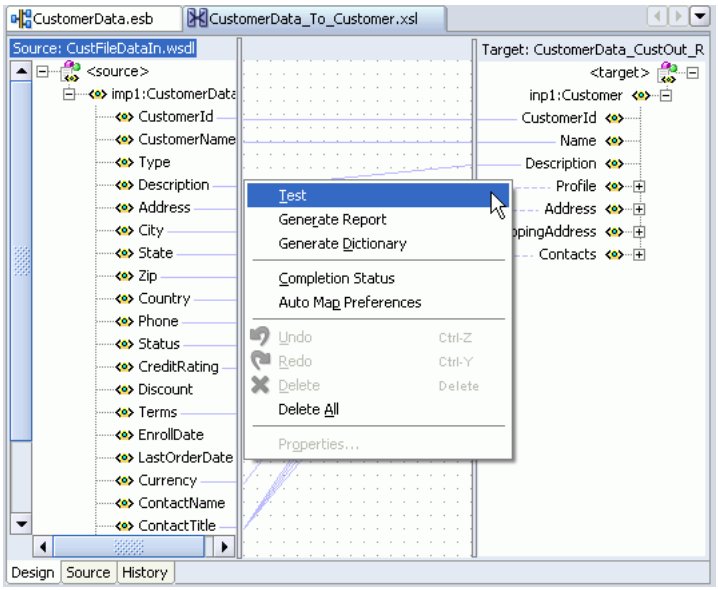
enable the AutoMap and Match Elements with Similar Names options, the Data Mapper tool automatically maps source elements to target elements after you click **OK** in the Auto Map Preferences dialog, as shown in Figure 6–4.

Figure 6–4 XSLT Data Mapper AutoMap



8. You can edit a new or existing transformation map.
   - Add new links by dragging and dropping source elements to target elements.
   - Remove (undo) links by selecting the link and right-clicking to bring up menu options. Select Undo link from the context menu.

Figure 6–5 XSLT Data Mapper Context Menu



9. After you have completed the transformation map, use **File** > **Save** to save the xsl file.

You can directly launch the Data Mapper tool by double-clicking on a data transformation icon in a routing service icon in the Design tab. If the transformation exists, the Data Mapper tool opens for you to update the transformation file. If the transformation file has not been specified yet, the Request Transformation Map dialog displays and enables you to create a new transformation file or select an existing transformation map file for update.

Note: You can also create an XSL transformation map file from an XSL stylesheet. Click **New**, then **XML**, then **XSL Map From XSL Stylesheet** from the **File** main menu in Oracle JDeveloper.

Note: If you select a file with an `xslt` extension such as `xform.xslt`, it opens the mapper pane to create a new XSL file named `xform.xslt.xsl`, even though your intension was to use the existing `xform.xslt` file. An `xsl` extension is appended to *any* file that does not already have an `xsl` extension, and you must create the mappings in the new file. As a workaround, ensure that your files first have an extension of `xsl`. If the XSL file has an extension of `xslt`, then rename it to `xsl`.

Using the XSLT Mapper

The following sections describe how to use the XSLT Mapper:

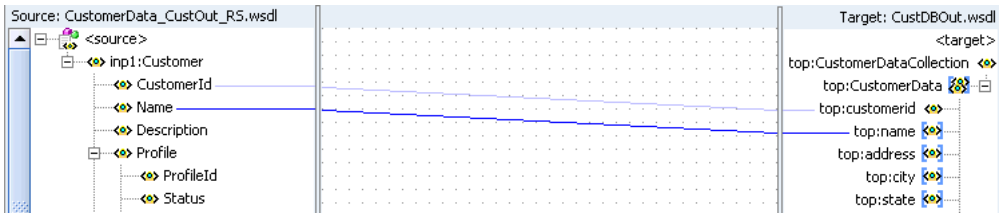
- Simple Copy by Linking Nodes on page 6-7
- Setting Constant Values on page 6-7
- Adding Functions on page 6-8
- Editing XPath Expressions on page 6-13

- Adding XSLT Constructs on page 6-14
- Automatically Mapping Nodes on page 6-17
- Viewing Unmapped Target Nodes on page 6-19
- Generating Dictionaries on page 6-20
- Creating Map Parameters and Variables on page 6-20
- Searching Source and Target Nodes on page 6-22
- Ignoring Elements in the XSLT Document on page 6-23
- Replacing a Schema in the XSLT Mapper on page 6-24

Simple Copy by Linking Nodes

To copy an attribute or leaf-element in the source to an attribute or leaf-element in the target, drag and drop the source to the target, as shown in Figure 6–6.

Figure 6–6 Linking Nodes



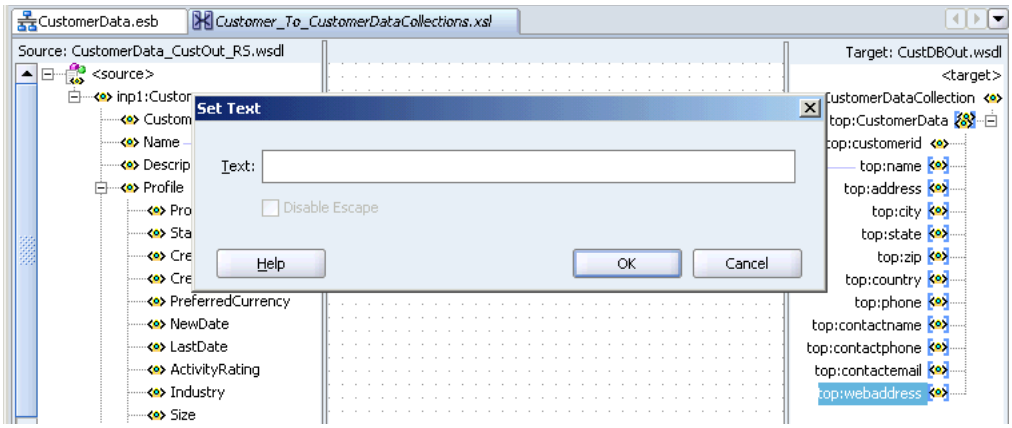
Setting Constant Values

Perform the following steps to set a constant value.

1. Select a node in the target tree.
2. Invoke the context menu by right-clicking the mouse.
3. Select the **Set Text** menu option.
4. Enter text in the Set Text window.
5. Click **OK** to save the text.

   A **T** icon is displayed next to the node that has text associated with it.
6. If you want to remove the text associated with the node, right click the node to invoke the Set Text window again. Delete the contents and click **OK**.

Figure 6–7 Set Text Window



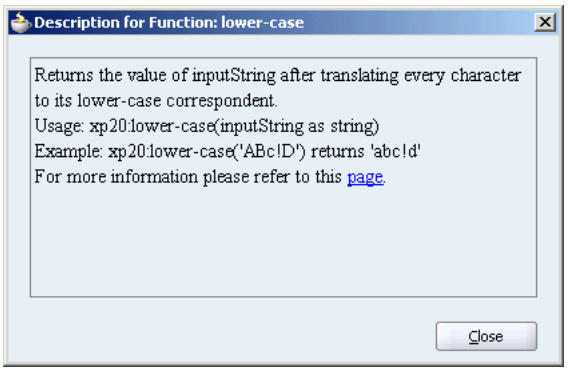
See Also: The online Help for the Set Text window for detailed information

Adding Functions

In addition to the standard XPath 1.0 functions, the Data Mapper provides a number of prebuilt extension functions and has the ability to support user-defined functions and named templates. The extension functions are prefixed with **xp20** or **orcl** and mimic XPath 2.0 functions.

To view function definitions and use a function in a data transformation map, Perform the following steps:

1. Select a category of functions, such as **String Functions**, from the **Component Palette**.
2. Right-click an individual function, such as **lower-case**.
3. Select **Help**. A window with a description of the function appears. You can also click a link at the bottom to access the function description at the World Wide Web Consortium at `www.w3.org`.



4. Drag a function from one of the functions categories in the Component Palette to the data mapper pane.
   - You can drag a function from the Component Palette on an existing link in the mapper pane.

     For example, drag the **lower-case** function under String Functions on an existing link into the mapper pane, such as **Email** in the source list to

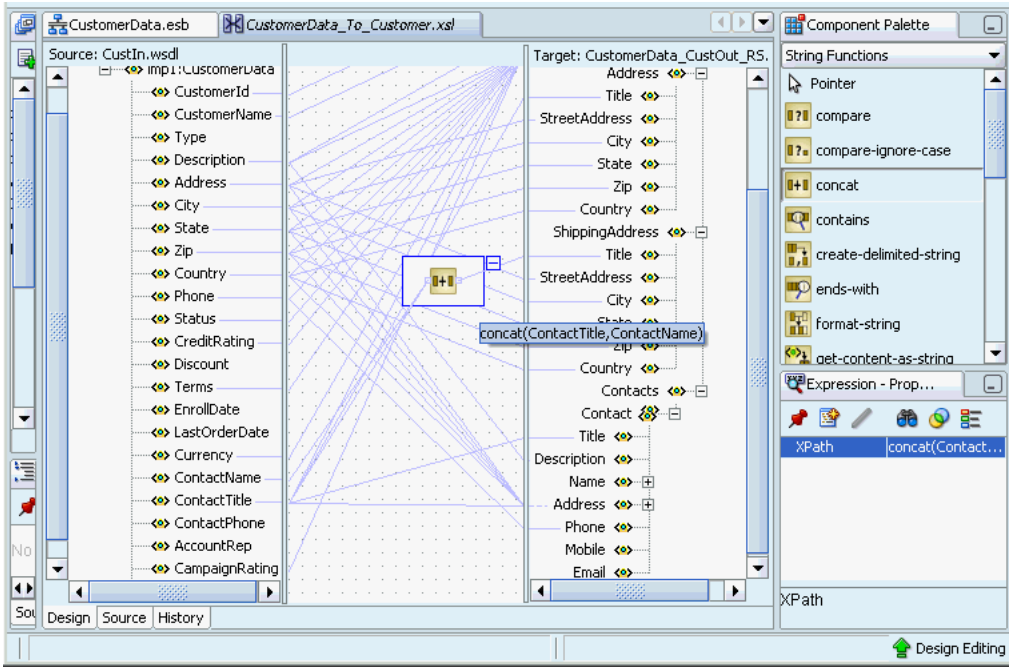
contactemail in the target list. This function converts the email value in the source element to lower case when it is output to the target element.

- You can drag a function into an empty area of the mapper pane. Then drag a source elements on the left handle of the function and a target element on the right handle of the function.

  For example, drag the **concat** function under String Functions to the mapper pane. Drag threads from multiple source elements, such as **CustomerData/ContactTitle** and **CustomerData/ContactName**, to the left handle of the **concat** function. Then drag a thread from the **Customer/ShippingAddress/Title** element in the target list. This function combines the values in the source elements when it is output to the target element.

5. When you have finished adding functions, use **File** > **Save** to save the changes to the data transformation xsl file.

Figure 6–8 Using the Concat Function



See Also:

- "Using a Domain-Value Map in a Transformation" on page 7-12
- Appendix A, "XPath Extension Functions" for reference information about the XPath extension functions.

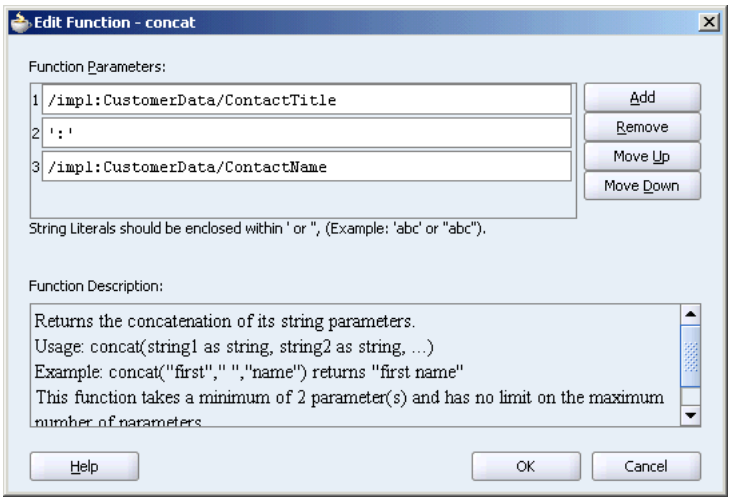
Editing Function Parameters

To edit the parameters of a function in the data mapper pane, double-click the function icon to launch the Edit Function dialog. This window enables you to add, remove, and reorder parameters.

For example, to edit the parameters of the **concat** function, double-click the function icon to launch the Edit Function - concat window. If you want to add a new parameter so that the output of the **concat** function is **ContactTitle: Contact Name**, then click

Add to add a new parameter **':'** (with single quotes) and reorder the parameters to get this output.

Figure 6–9 Editing Function Parameters



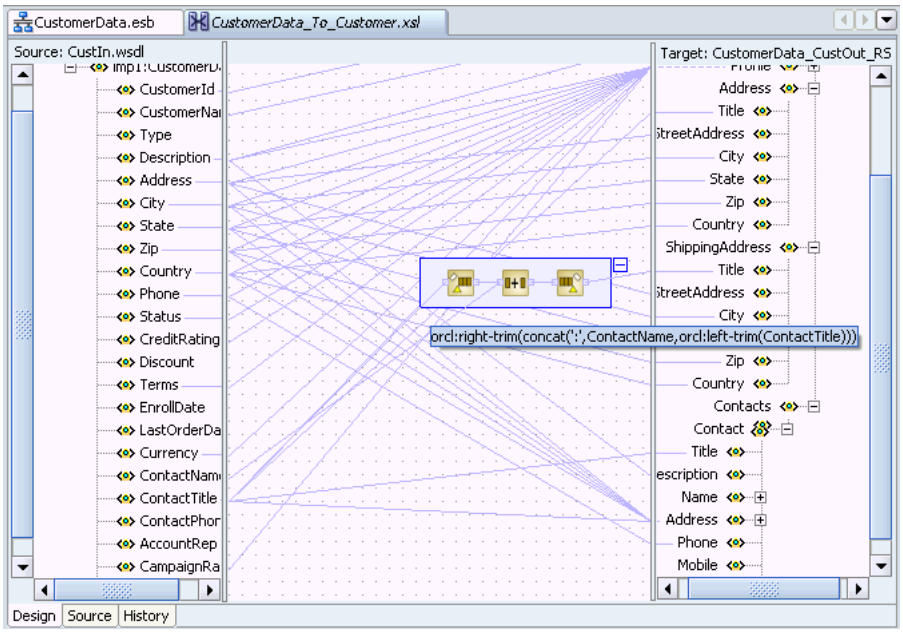
See Also: The online Help for the Edit Function window by clicking the **Help** button to see how to add, remove, and reorder function parameters

Chaining Functions

Complex expressions can be built by chaining functions. The chaining function can also be defined by dragging and dropping the function to a connecting link.

For example, to remove all leading and trailing spaces from the output of the **concat** function discussed in "Editing Function Parameters" on page 6-9, use the left-trim and right-trim functions and chain them as shown in the Figure 6–10.

Figure 6–10 Chaining Functions



Named Templates

Some complicated mapping logic cannot be represented or achieved by visual mappings. For these situations, named templates are useful. Named templates enable you to share common mapping logic. You can define the common mapping logic as a named template and then use it as often as you want.

You define named templates in the source view, and they appear in the **User Defined Named Templates** list of the **Component Palette**. You can use named templates in almost the same way as you use other functions. The only difference is that you cannot link the output of a named template to a function or another named template; you can only link its output to a target node in the target tree.

To write named templates, you must be familiar with the XSLT language. See any XSLT book or visit the following URL for details about writing named templates:

http://www.w3.org/TR/xslt

Importing User-Defined Extension Functions

You can import your own set of Java functions, which appear in the function palette under the **User defined Extension Functions** category. They can be used like any other function. To add functions, select **Preferences**, then **XSL Maps** from the **Tools** main menu.

In the XSLT mapper source view, you can use the user-defined functions in the following way:

```
NamespacePrefix:FunctionName([ARGUMENT1, ARGUMENT2, ARGUMENT3...])
```

The namespace prefix should be declared in the XSL file as follows:

```
"http://www.oracle.com/XSL/Transform/java/<ClassNameincludingthePackageName>"
```

Creating User-Defined Extension Functions

Oracle Application Server release 10.1.3.4 includes support for creating user-defined extension functions in ESB. Perform the following steps to create your own set of user-defined extension functions:

1. Create a sample Java source file with the `SampleExtensionFunctions` class file with a `ExtnFunc()` method.

```
package oracle.sample;
import java.util.List;
import javax.xml.xpath.XPathFunction;
import javax.xml.xpath.XPathFunctionException;
public class SampleExtensionFunctions implements XPathFunction{
                    public Object evaluate(List args) throws
XPathFunctionException {
                return ExtnFunc();
           }
        public static String extnFunc(){
                return "Hello From ExtnFunc";
        }
}
```

2. Compile this `SampleExntensionFunctions.java` file with `$ORACLE_HOME/lib/xmlparserv2.jar` in classpath.

3. Copy the `SampleExntensionFunctions.class` file to the `$ORACLE_HOME/integration/esb/system/classes/oracle/sample` directory.

Note: You need to create this directory structure manually, if it does not exist already.

4. Change `$ORACLE_HOME/integration/esb/config/extn-xpath-functions-config.xml` as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ==================================================================== -->
<!--  XPath extension functions available to ESB 10.1.3
 -->
<!-- ==================================================================== -->
<esb-xpath-functions  xmlns="http://xmlns.oracle.com/soa/config/esb"
 version="10.1.3">
   <function id="ExtnFunc">
       <classname>oracle.sample.SampleExtensionFunctions</classname>
       <property id="namespace-uri">
<value>http://www.oracle.com/XSL/Transform/java/oracle.sample.SampleExtensionFu
nctions</value>
          <comment>Namespace URI for this function</comment>
       </property>
       <property id="namespace-prefix">
           <value>udf</value>
           <comment>Namespace prefix for this function</comment>
       </property>
   </function></esb-xpath-functions>
```

5. In the XSL file, you should add the proper namespace to the Stylesheet section and the user-defined function to the Template section, as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?oracle-xsl-mapper
  <!-- SPECIFICATION OF MAP SOURCES AND TARGETS, DO NOT MODIFY. -->
  <mapSources>
    <source type="WSDL">
      <schema location="readFile.wsdl"/>
      <rootElement name="input" namespace="http://www.oracle.com/input"/>
    </source>
  </mapSources>
  <mapTargets>
    <target type="WSDL">
      <schema location="writeOut.wsdl"/>
      <rootElement name="output" namespace="http://www.oracle.com/output"/>
    </target>
  </mapTargets>
  <!-- GENERATED BY ORACLE XSL MAPPER 10.1.3.4.0(build 080324.0200) AT [FRI APR
 11 11:43:56 IST 2008]. -->
?>
<xsl:stylesheet version="1.0"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
                xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
                xmlns:imp1="http://www.oracle.com/input"
                xmlns:pc="http://xmlns.oracle.com/pcbpel/"
xmlns:ehdr="http://www.oracle.com/XSL/Transform/java/oracle.tip.esb.server.head
ers.ESBHeaderFunctions"
                xmlns:ns0="http://www.w3.org/2001/XMLSchema"
                xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
                xmlns:hwf="http://xmlns.oracle.com/bpel/workflow/xpath"
xmlns:ns1="http://xmlns.oracle.com/pcbpel/adapter/file/writeOut/"
```

```
xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.fun
ctions.Xpath20"
xmlns:xref="http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRef
XPathFunctions"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:ns2="http://www.oracle.com/output"
                xmlns:ora="http://schemas.oracle.com/xpath/extension"
 xmlns:ids="http://xmlns.oracle.com/bpel/services/IdentityService/xpath"
xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.fun
ctions.ExtFunc"
xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/file/readFile/"
               xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/file/"
xmlns:udf="http://www.oracle.com/XSL/Transform/java/oracle.sample.SampleExtensi
onFunctions"
exclude-result-prefixes="xsl plt imp1 pc ns0 jca tns hdr ns1 ns2 bpws ehdr hwf
xp20 xref ora ids orcl">
  <xsl:template match="/">
    <ns2:output>
      <ns2:stringA>
        <xsl:value-of select="udf:ExtnFunc()"/>
      </ns2:stringA>
    </ns2:output>
  </xsl:template>
</xsl:stylesheet>
```

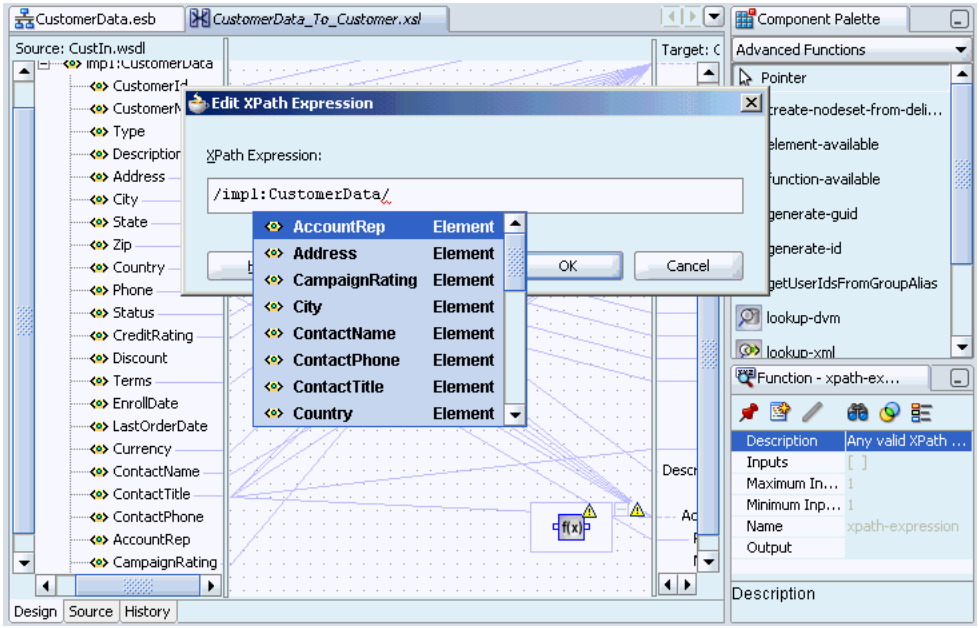
6. Restart the OC4J_SOA instance.
7. Deploy the project.

Editing XPath Expressions

To use an XPath expression in a transformation mapping, select **Advanced Functions** from the **Component Palette** and drag and drop **xpath-expression** from the list into the data mapper transformation pane.

When you double-click the icon, the Edit XPath Expression dialog appears. You can press the **Ctrl** + **spacebar** to invoke the XPath Building Assistant, as show in Figure 6–11.

Figure 6–11 The XPath Building Assistant



See Also: The online Help for the Edit XPath Expression window, which includes a link to instructions on using the XPath Building Assistant

Adding XSLT Constructs

While mapping complex schemas, it is sometimes essential to conditionally map a source node to a target or map an array of elements in the source to an array of elements in the target. The XSLT Mapper provides various XSLT constructs in the context sensitive menu of the target tree for the these conditional scenarios. These constructs include **for-each**, **if**, or **choose**.

To add an XSLT element such as **for-each**, **if**, or **choose** to a schema element:

1. Select the element in the target tree.
2. Right-click and select **Add XSL Node** to bring up the context menu.
3. Choose the required XSLT element in the menu.

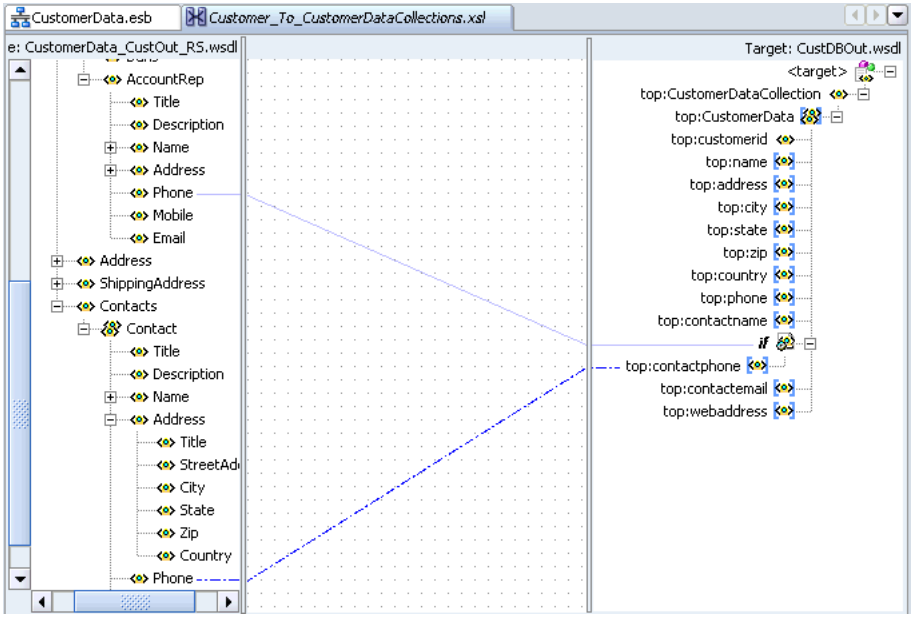
Conditional Processing with xsl:if

With the `if` construct, you can choose to map a source element if a the value exists and map a value from a different element when it does not exist. For example, if the source contains an `AccountRep` phone number use that value. If the `AccountRep` phone number does not exist, then use the Contact phone number.

1. Select the target element in the target tree and right-click to bring up the context sensitive menu.
2. Select **Add XSL Node**, and then **if**.
3. Connect a source element to the **if** element in the target tree.
4. Connect a source element to the target element.

Figure 6–12 shows the results.

Figure 6–12 Conditional Processing with xsl:if



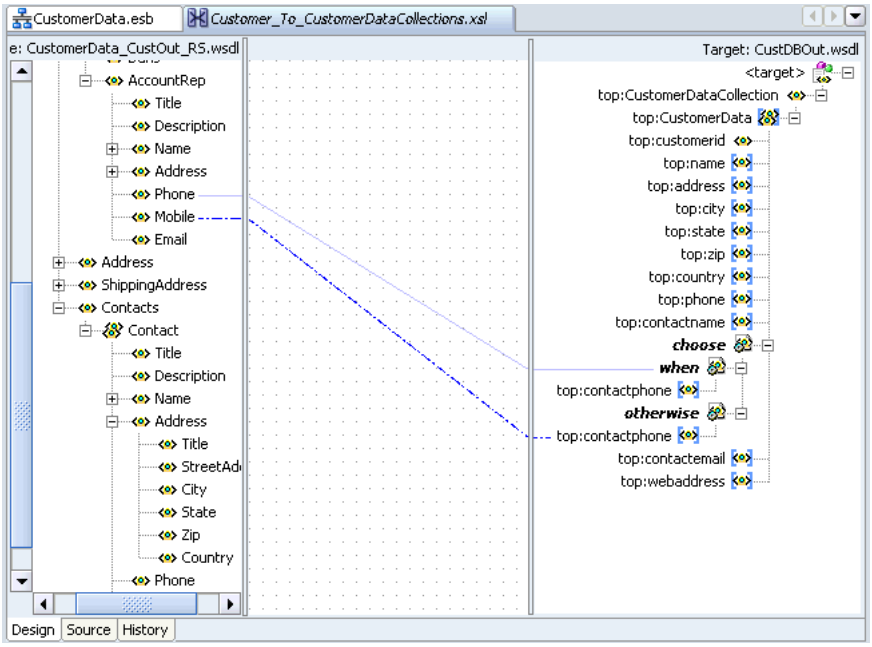
Conditional Processing with xsl:choose

With the `choose` construct, you can copy a source element to a specified target element, if the source element exists. Otherwise, copy a different source element to the target element.

1. Select the target element in the target tree and right-click to bring up the context sensitive menu.
2. Select **Add XSL Node**, and then **choose**.
3. Connect the source element to the target element to define the condition.
4. Select **choose** in the target tree and right-click to bring up the context sensitive menu.
5. Select **XSL Add Node** and then **otherwise**.
6. Connect the source element to the target element under **otherwise**.

Figure 6–13 shows the results.

Figure 6–13 Conditional Processing with xsl:choose



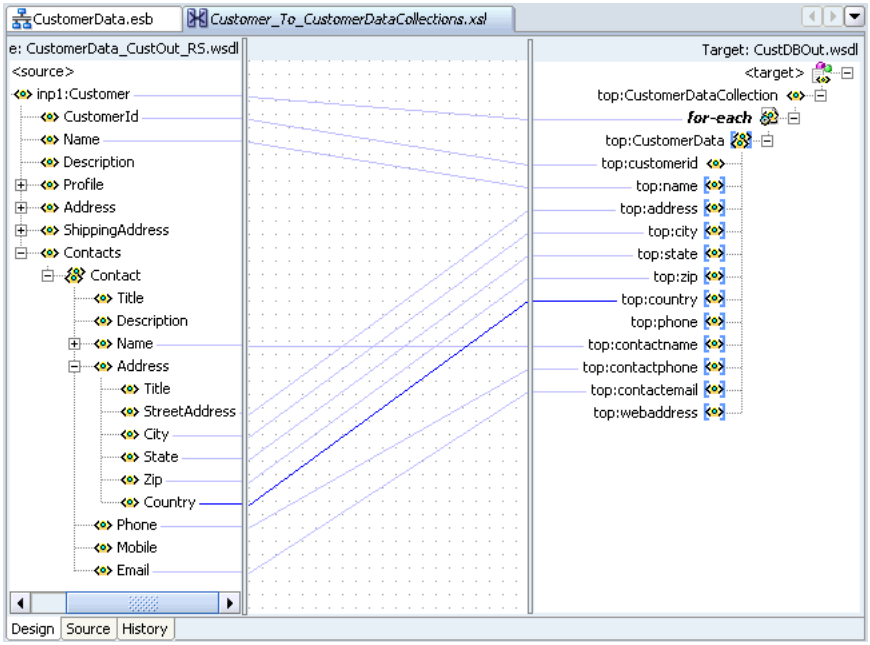
Handling Repetition or Arrays

The XSLT Mapper allows repeating elements on the source to be copied to repeating elements on the target.

1. Select a target element in the target tree and right-click to bring up the context sensitive menu.
2. Select **Add XSL Node**, and then **for-each**.
3. Connect the repeating source elements to the targets elements.

Figure 6–14 shows the results.

Figure 6–14 Handling Repetition or Arrays



Note: Executing an auto map automatically inserts **xsl:for-each**.

Automatically Mapping Nodes

Mapping nonleaf nodes starts the auto map feature. The system automatically tries to link all relevant nodes under the selected source and target, as shown in Figure 6–4 on page 6-5.

The behavior of the auto map can be tuned by altering the settings in Oracle JDeveloper preferences or by right-clicking the transformation window and selecting **Auto Map Preferences**. This displays the window shown in Figure 6–3 on page 6-4.

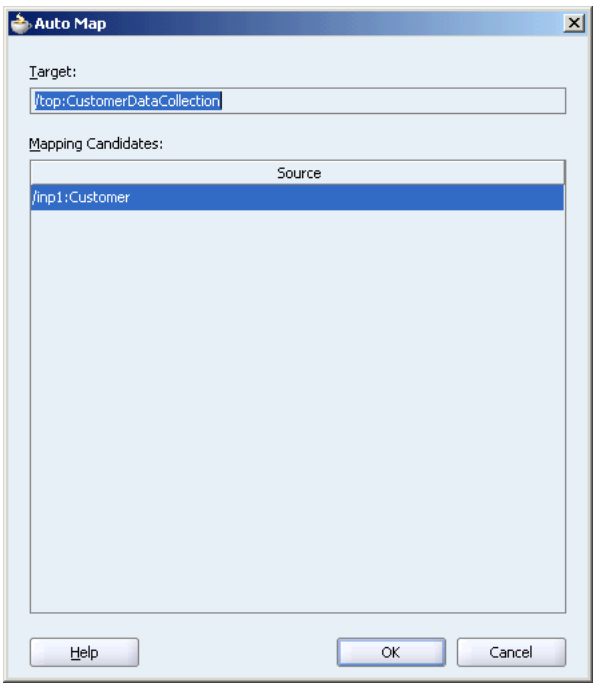
This window enables you to customize your auto mapping as follows:

- Invoke the automatic mapping feature, which attempts to automatically link all relevant nodes under the selected source and target. When disabled, you must individually map relevant nodes.
- Display and review all potential source-to-target mappings detected by the XSLT Mapper, and then confirm to create them.
- Be prompted to customize the auto map preferences before the auto map is invoked.
- Select the **Basic** or **Advanced** method for automatically mapping source and target nodes. This enables you to customize how the XSLT mapper attempts to automatically link all relevant nodes under the selected source and target.
- Manage your dictionaries. The XSLT Mapper uses the rules defined in a dictionary when attempting to automatically map source and target elements.

See Also: The online Help for the Auto Map Preferences window by clicking the **Help** button to see a description of the fields

To see potential source mapping candidates for a target node, right-click the target node, select **Show Matches**, and click **OK** in the Auto Map Preferences window. The Auto Map window appears, as shown in Figure 6–15.

Figure 6–15 Auto Mapping Candidates

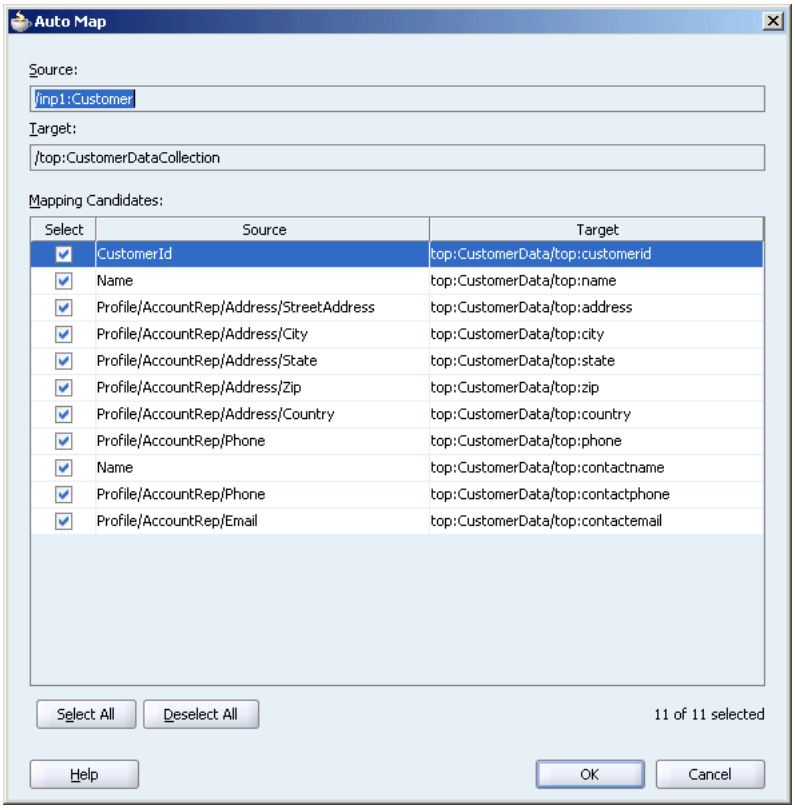


See Also: The online Help for the Auto Map window by clicking the **Help** button to see a description of the fields.

Auto Map with Confirmation

When the **Confirm Auto Map Results** check box shown in Figure 6–3 on page 6-4 is selected, a confirmation window appears. If matches are found, the potential source-to-target mappings detected by the XSLT Mapper are displayed, as shown in Figure 6–16. The window enables you to filter one or more mappings.

Figure 6–16 Auto Map with Confirmation



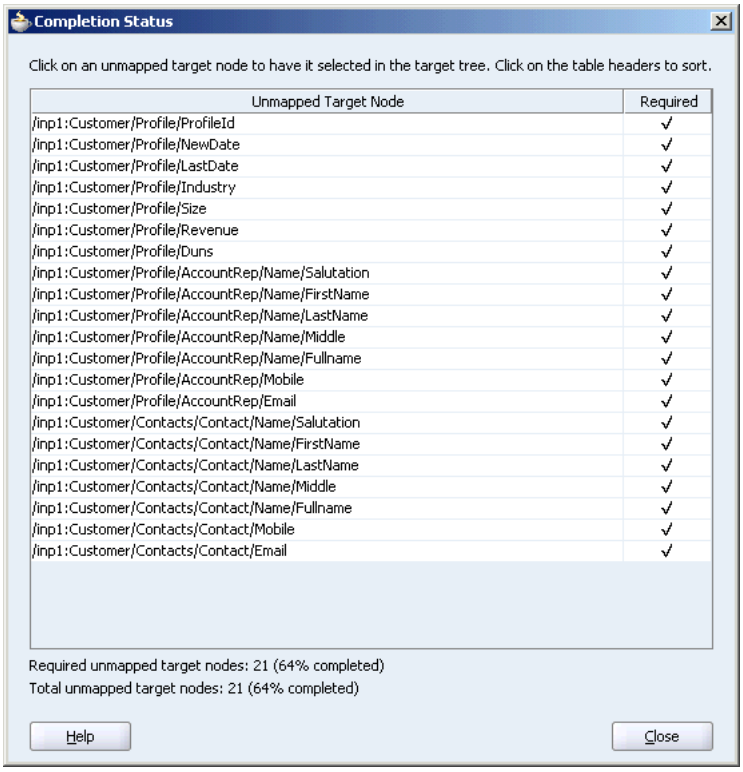
See Also: The online Help for the Auto Map window by clicking the **Help** button to see a description of the fields

Viewing Unmapped Target Nodes

You can view a list of target nodes that are currently unmapped to source nodes. Right click in the mapper pane as shown in Figure 6–5 on page 6-6 and select **Completion Status**. This window provides statistics at the bottom about the number of unmapped target nodes. This window enables you to identify and correct any unmapped nodes before you test your transformation mapping logic on the Test XSL Map window. Select a target node in the list. The node is highlighted. A check mark indicates that the target node is required to be mapped. If not required, the check box is empty.

Figure 6–17 provides an example of the Completion Status window.

Figure 6–17 Completion Status



Generating Dictionaries

A dictionary is an XML file that captures the synonyms for mappings. Right-click in the mapper pane as shown in Figure 6–5 on page 6-6 and select **Generate Dictionary**. This prompts you for the dictionary name and the directory in which to place the dictionary.

The XSLT Data Mapper uses the rules defined in the dictionary when attempting to automatically map source and target elements. For example, you may want to map a purchase order to a purchase order acknowledgment, then reuse most of the map definitions later.

1. Build all the mapping logic for the purchase order and purchase order acknowledgment.
2. Generate a dictionary for the created map.
3. Create a new map using a different purchase order and purchase order acknowledgment.
4. Load the previously created dictionary by selecting **Tools** > **Preferences** > **XSL Maps** > **Auto Map** in Oracle JDeveloper.
5. Perform an automatic mapping from the purchase order to the purchase order acknowledgment.

Creating Map Parameters and Variables

You can create map parameters and variables. You create map parameters in the source tree and map variables in the target tree.

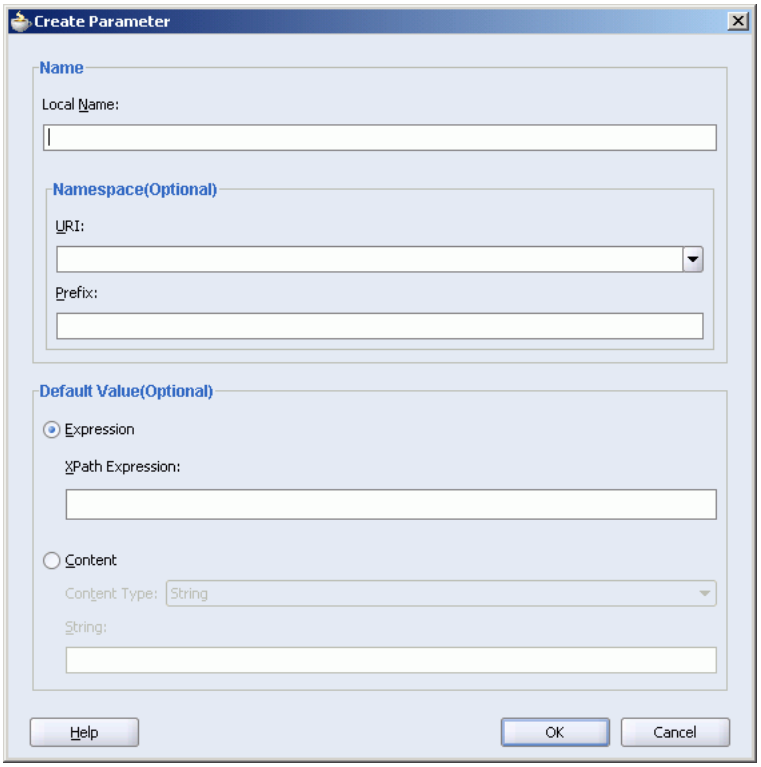
Note the following issues:

- Parameters are created in the source tree, are global, and can be used anywhere in the mappings.
- Variables are created in the target tree, and are either global or local. Where they are defined in the target tree determines if they are global or local.
  - Global variables are defined immediately below the **<target>** node and immediately above the actual target schema (for example, **POAcknowledge**). Right-click the **<target>** node to create a global variable.
  - Local variables are defined on a specific node below the actual target schema (for example, subnode **name** on schema **POAcknowledge**). Local variables can have the same name as long as they are in different scopes. Local variables can only be used in their scopes, while global variables can be used anywhere in the mappings.

Creating a Map Parameter

1. Right-click the source tree root and select **Add Parameter**.

   The Create Parameter window appears.

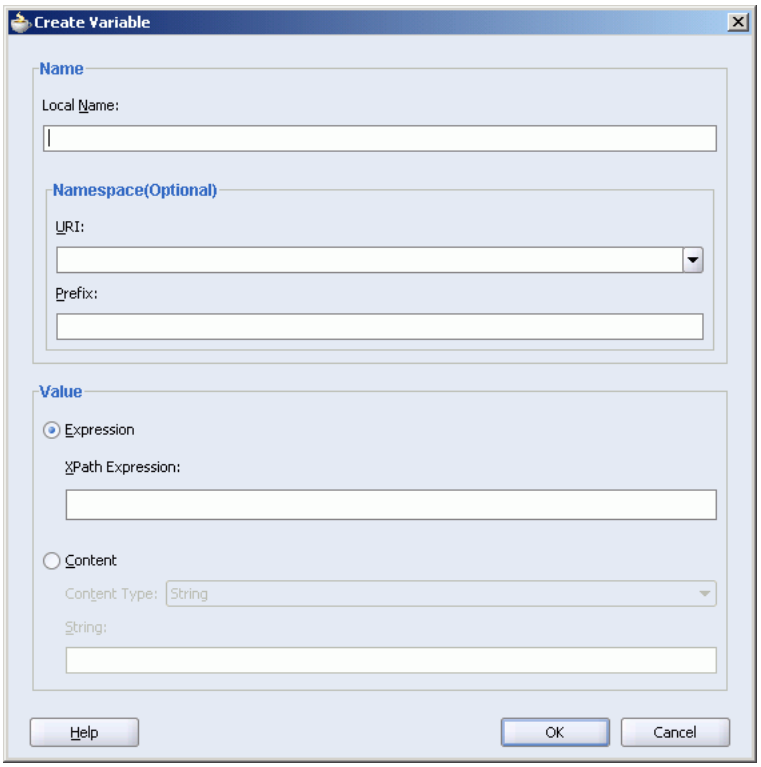


2. Specify the information for the parameter.
3. Click **OK**.

Creating a Map Variable

1. Right-click the target tree root and select **Add Variable**. If you right-click a node below the target tree root, select **Insert Variable**.

   The Create Variable window appears.



2. Specify the information for the variable.
3. Click **OK**.

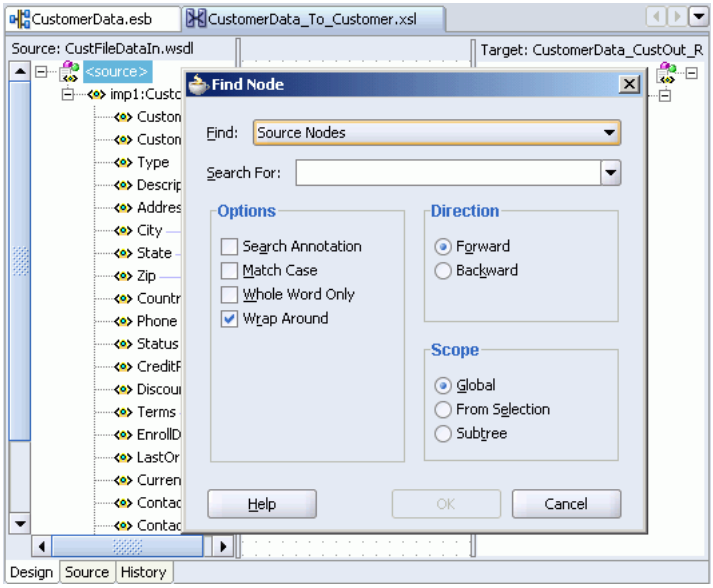
Searching Source and Target Nodes

You can search source and target nodes. For example, you can search in a source node named **Customer** for all occurrences of the subnode named **Name**.

To search for a source or target node:

1. Right-click a source or target node.
2. Select the **Find** menu item.

   The Find Node dialog displays.



3. In the **Search For** field, enter a keyword to search on.
4. Specify additional details, as necessary. For example:
   - Select **Search Annotations** if you want annotations text to also be searched.
   - Specify the scope of the search. You can search the entire source or target tree, search starting from a selected position, or search within a selected subtree.

   The first match found is highlighted, and the Find window closes. If no matches are found, a message displays on-screen.
5. Select the **F3** key to find the next match in the direction specified. To search in the opposite direction, select the **Shift** and **F3** keys.

Note: You cannot search on functions or text values set with the **Set Text** option.

Ignoring Elements in the XSLT Document

When the XSLT Mapper encounters any elements in the XSLT document that cannot be found in the source or target schema, it is unable to process them and displays an `Invalid Source Node Path` error. XSL map generation fails. You can create and import a file that directs the XSLT Mapper to ignore and preserve these specific elements during XSLT parsing by selecting **Preferences**, then **XSL Maps** in the **Tools** main menu of Oracle JDeveloper.

For example, preprocessing may create elements named `myElement` and `myOtherElementWithNS` that you want the XSLT Mapper to ignore when it creates the graphical representation of the XSLT document. You create and import a file with these elements to ignore that includes the following syntax:

```
<elements-to-ignore>
   <element name="myElement"/>
   <element name="myOtherElementWithNS" namespace="NS"/>
</elements-to-ignore>
```

You must restart Oracle JDeveloper after importing the file.

Replacing a Schema in the XSLT Mapper

You can replace the map source schema and map target schema that currently display in the XSLT Mapper. Right click in either the source or target panel and select **Replace Schema**. This opens the Select Source and Target Schema window, which enables you to select the new source or target schema to use.

Using Instance Id in the XSLT Mapper

You can use the ESB instance Id to correlate between messages in the XSLT mapper. In the source view of the XSLT mapper, you can use the `ehdr:getInstanceID()` function in the following way to access the id of an ESB instance:

```
<xsl:value-of select="ehdr:getInstanceID()"/>
```

A sample XSL file is shown in the following example:

```
<xsl:template match="/">
   <imp1:Root-Element>
     <imp1:info>
       <xsl:value-of
select="concat(/imp1:Root-Element/imp1:Root-Element/imp1:id,'--',
       /imp1:Root-Element/imp1:Root-Element/imp1:appname,'--',/imp1:
       Root-Element/imp1:Root-Element/imp1:operation)"/>
     </imp1:info>
     <imp1:more_info>
      <xsl:value-of select="ehdr:getInstanceID()"/>
     </imp1:more_info>
   </imp1:Root-Element>
</xsl:template>
```

Using the Mapper Test Utility

The XSLT Mapper provides a test utility to test the style sheet or map. The test tool can be invoked by selecting the **Test** menu item from the mapper pane context sensitive menu, as shown in Figure 6–5 on page 6-6. When you select **Test**, the Test XSL Map dialog displays, shown in Figure 6–18 on page 6-25.

The test settings you specify are stored and do not need to be entered again the next time you test. Test settings must be entered again if you close and reopen Oracle JDeveloper.

This section contains the following topics:

- Testing a Map on page 6-24
- Generating Reports on page 6-26
- Sample XML Generation on page 6-27

Testing a Map

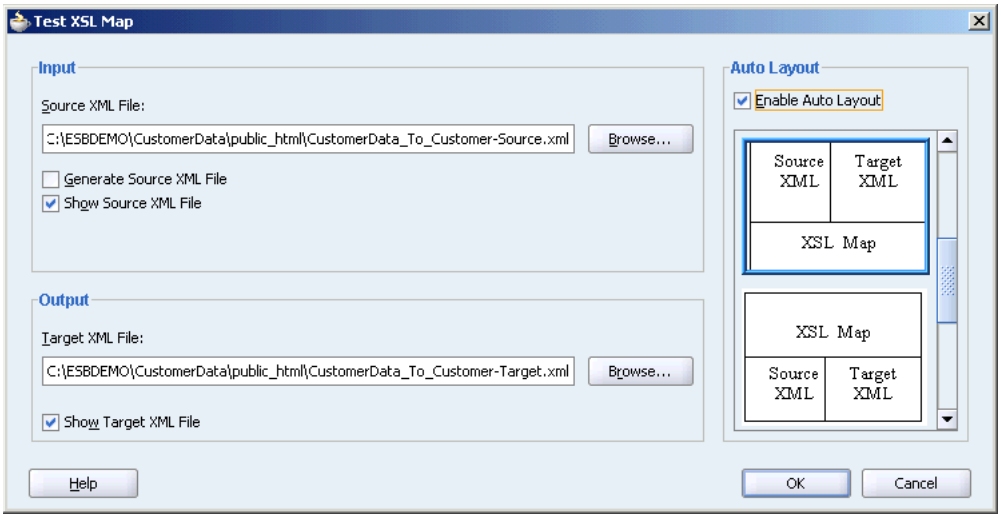
Use the Data Mapper test utility for testing from data transformation maps.

To test an XSL map:

1. Select the **Test** menu item from the mapper pane context sensitive menu.

   The Test XSL Map dialog displays, shown in Figure 6–18.

Figure 6–18 Test XSL Map Dialog



2. Choose to allow a sample source XML file to be generated for testing or click **Browse** to specify a different source XML file in the **Source XML File** field.

   When you click **OK**, the source XML file is validated. If validation passes, transformation occurs, and the target XML file is created.

   If validation fails, no transformation occurs and a message displays on-screen.

3. Select the **Generate Source XML File** check box to create a sample XML file based on the map source XSD schema.

4. Select the **Show Source XML File** check box to display the source XML file for the test. The source XML file displays in an Oracle JDeveloper XML editor.

   If the map has defined parameters, the **Parameters** table appears. If you want to specify a value, click **Specify Value** and make appropriate edits to the **Type** and **Value** columns.

5. Enter a file name in the **Target XML File** field or browse for a file name in which to store the resulting XML document from the transformation.

6. Select the **Show Target XML File** check box to display the target XML file for the test. The target XML file displays in an Oracle JDeveloper XML editor.

7. If you select to show both the source and target XML, you can customize the layout of your XML editors. Select **Enable Auto Layout** and click one of the patterns.

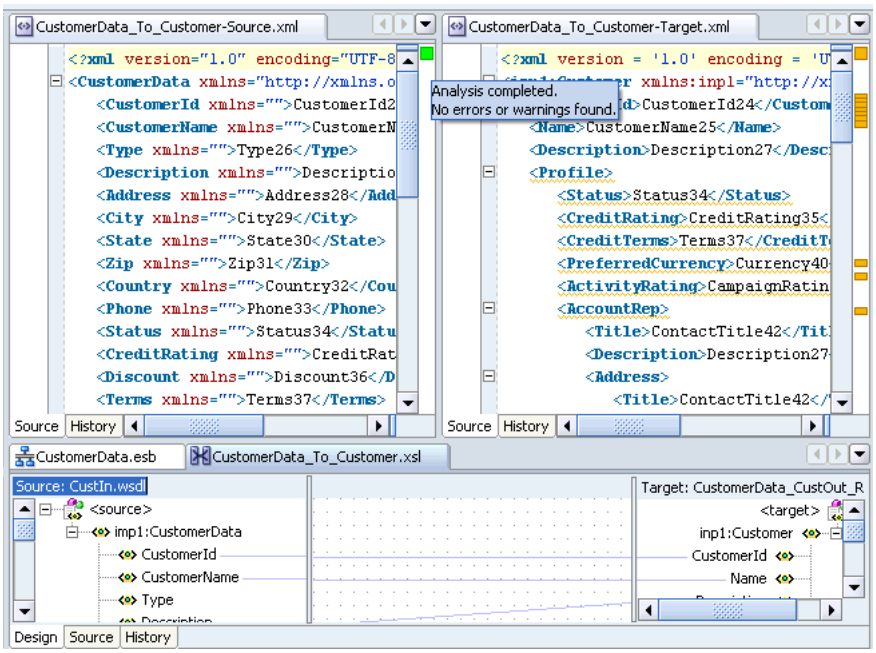
   For this example, the source XML and target XML display side-by-side, with the XSL map underneath (the default setting).

8. Click **OK**.

   The test results appear, as shown in Figure 6–19.

   The source and target XML each display in an Oracle JDeveloper XML editor. You can right-click an editor and select **Validate XML** to validate the source or target XML against the map source or target XSD schema.

Figure 6–19 Test Window



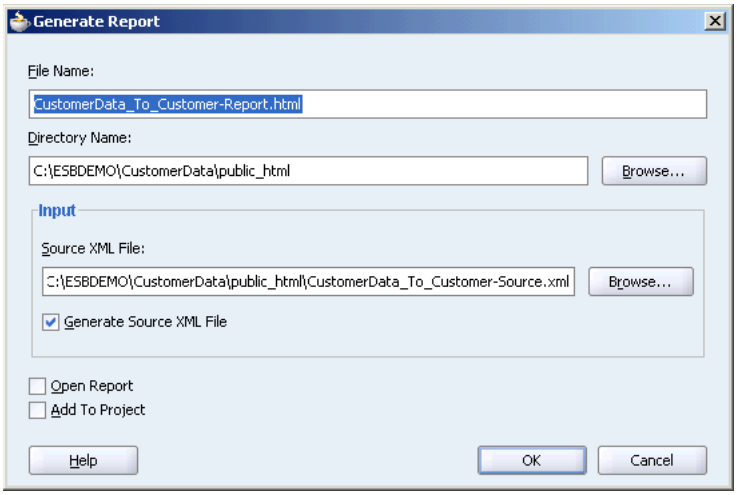
Generating Reports

You can generate an HTML report with the following information:

- XSL map file name, source and target schema file names, their root element names, and their root element namespaces
- Target document mappings
- Target fields not mapped (including mandatory fields)
- Sample transformation map execution

To generate a report, right-click the transformation window and select **Generate Report**. The Generate Report window appears in the transformation window, as shown in Figure 6–20. If the map has defined parameters, the **Parameters** table appears.

Figure 6–20 The Generate Report Dialog



See Also: The online Help for the Generate Report window by clicking the **Help** button to see detailed information

Correcting Memory Errors When Generating Reports

If you attempt to generate a report and receive an out-of-memory error, increase the heap size of the JVM as follows:

1. Open the *`Oracle_Home`*`\integration\jdev\jdev\bin\jdev.conf` file.
2. Go to the following section:

```
# Set the maximum heap to 512M
#
AddVMOption     -Xmx512M
```

3. Increase the size of the heap as follows (for example, to `1024`)

```
AddVMOption     -Xmx1024M
```

In addition, you can also uncheck the **Open Report** option on the **Generate Report** window before generating the report.

Sample XML Generation

You can customize sample XML generation by specifying the following parameters. Select **Preferences**, then **XSL Maps** in the **Tools** main menu of Oracle JDeveloper to display the Preferences window.

- Number of repeating elements

  Specifies how many occurrences of an element are created if the element has the attribute `maxOccurs` set to a value greater than `1`. If the specified value is greater than the value of the `maxOccurs` attribute for a particular element, the number of occurrences created for that particular element is the `maxOccurs` value, not the specified number.

- Generate optional elements

  If selected, any optional element (its attribute `minOccurs` set to a value of `0`) is generated the same way as any required element (its attribute `minOccurs` set to a value greater than `0`).

- Maximum depth

  To avoid the occurrence of recursion in sample XML generation caused by optional elements, specify a maximum depth in the XML document hierarchy tree beyond which no optional elements are generated.

7

Domain-Value Maps

This chapter introduces domain-value maps, presents the XML structure required by Oracle Enterprise Service Bus for a domain-value-map, describes how to create, populate, import and export domain-value maps using the Oracle ESB Control, and how to use domain-value maps when designing transformations within enterprise service bus routing rules.

This chapter contains the following topics:

- Understanding Domain-Value Maps on page 7-1
- Creating and Populating Domain-Value Maps on page 7-1
- Using a Domain-Value Map in a Transformation on page 7-12

Understanding Domain-Value Maps

Applications that you want to integrate using Oracle Enterprise Service Bus likely use different values to represent the same information. For example, one application might represent a state with the long name (Massachusetts) while another application may represent the state with an abbreviation (MA). A **domain-value map** enables you to associate values from one application with values from another.

Each domain-value map typically holds a specific category of value mappings among multiple applications. For example, one domain-value map might hold mappings for state codes and another might hold mappings for units of measurement.

After a domain-value map is created and populated using Oracle ESB Control, it can be used with the Oracle JDeveloper Mapper tool while developing XSLT data transformations during design time. Then, at runtime the lookups for application-specific values occur.

For example, suppose you want to use a a domain-value map to perform a runtime lookup to convert long state names input to the two-letter state code output. In this scenario, the state name is passed to an Oracle Enterprise Service Bus. Within the enterprise service bus, the data is transformed by a transformation specified in the routing rule of the routing service from the state name to the state code using a domain-value map look up.

Creating and Populating Domain-Value Maps

The overall process of creating and using a domain value map is summarized in the following list:

1. Using the Oracle ESB Control, you create a domain-value map and populate it with values that will need to be mapped across the applications integrated using the enterprise service bus. This process is described in the subsections of this topic.
2. Using Oracle JDeveloper, you create an XSL file to define the source to target transformation within a routing rule transformation. This process is described in "Creating an XSL Map File for Data Structure Transformation" on page 5-20.
3. At runtime, Oracle Enterprise Service Bus uses the domain-value map to look up appropriate values and populate the targets for the applications that it integrates with.

You create (and then save) a domain-value map using the Map view of the Oracle ESB Control. When you save the domain-value map, it is saved in the ESB repository, which makes it available for use with the data mapper in Oracle JDeveloper. The following sections describe the various methods available for creating (and saving) a domain-value map.

- Creating a New Domain-Value Map from Scratch on page 7-2
- Exporting a Domain-Value Map on page 7-4
- Domain-Value Map Template and XSD Files on page 7-5
- Importing an Existing Domain-Value Map File on page 7-7
- Importing Rows Into a Domain-Value Map on page 7-8
- Editing a Domain-Value Map on page 7-10
- Deleting a Domain-Value Map on page 7-12

Creating a New Domain-Value Map from Scratch

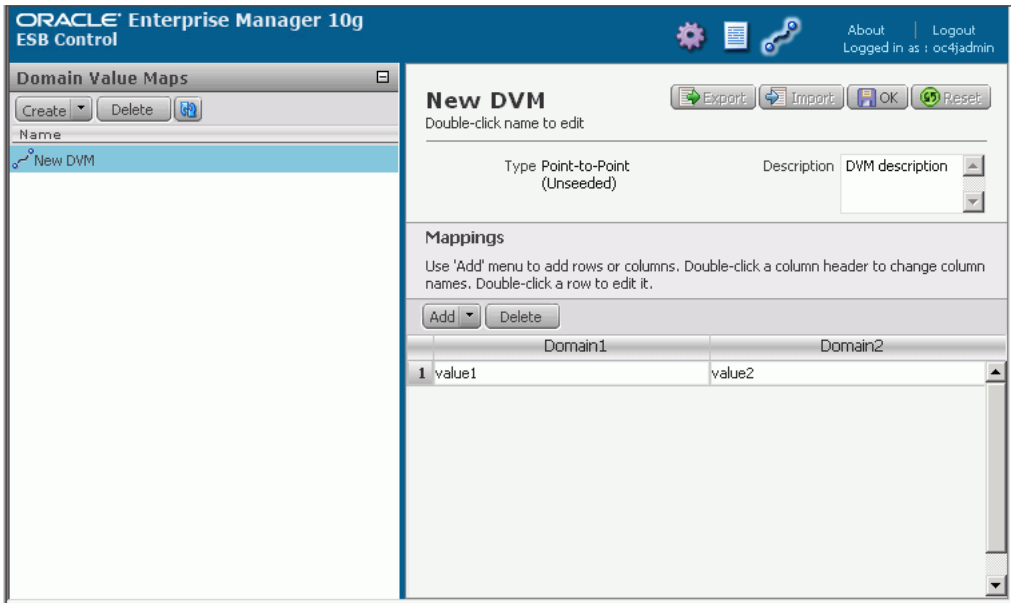
To create a domain-value map from scratch, follow these steps:

1. At the top of the Oracle ESB Control, click the **Map** icon.
2. Click the Create down-arrow on the left side of the Oracle ESB Control.

   Two choices are presented: **Create a new map** and **Import a new map.**
3. Click **Create a new map.**

   The Oracle ESB Control refreshes and appears similar to Figure 7–1.

Figure 7–1 Oracle ESB Control Map View – Create New Map



4. Replace the default **New DVM** with a name for the domain-value map file. To edit **New DVM**, click in the name field in the right pane of the screen above the **Mappings** area.

   For example, you replace **New DVM** with **StateCodes** to identify a domain-value map for state names, abbreviations, and short codes.

   If you export the domain-value map, the name you specify is used for the export file name. See "Exporting a Domain-Value Map" on page 7-4 for information about exporting a domain-value map.

5. In the **Description** field, enter a description of the domain-value map.

   For example, you might enter: Mappings of state names, abbreviations, and short codes

6. If you need additional columns, click the **Add down-arrow,** and then click **Column** once for each additional column you want to add. If you add too many columns, select the column and click **Delete.**

   Each column represents a domain. If you will be using the domain-value map to map values among four domains, for example, you click Column twice to add two more columns. There are no restrictions on what can be considered a domain; you specify domains based on your needs. A domain might be, for example, a trading partner, an organization, a department, an application, and so on.

   For example, add one column **Add** > **Column**.

7. Double-click a column name to change it from the default value of Application*n* to a more meaningful name. Each column name must be unique within the domain-value map.

   For example, you change the default column names to **Long**, **Abbrev**, and **Short**.

8. If you need additional rows, click the **Add down-arrow,** and then click **Row** once for each additional row you want to add. If you add too many rows, select the row and click **Delete,** or leave it as-is. Empty rows are deleted when you save the domain-value map.

9. Double-click a row and enter values for the domain value map.

   For example, enter **Massachusetts**, **Mass**, and **MA** in the columns under **Long**, **Abbrev**, and **Short**.

10. Repeat the previous step until you have entered the desired rows. You do not have to enter all rows that will be required by the applications. You can edit the domain-value map to add more rows later.

    Figure 7–2, shows an example of a domain-value map.

11. Review the name for the domain-value map and click the **Save** button above the Description field.

    After you click the Save button, you can change components of the domain-value map, as described in "Editing a Domain-Value Map" on page 7-10.

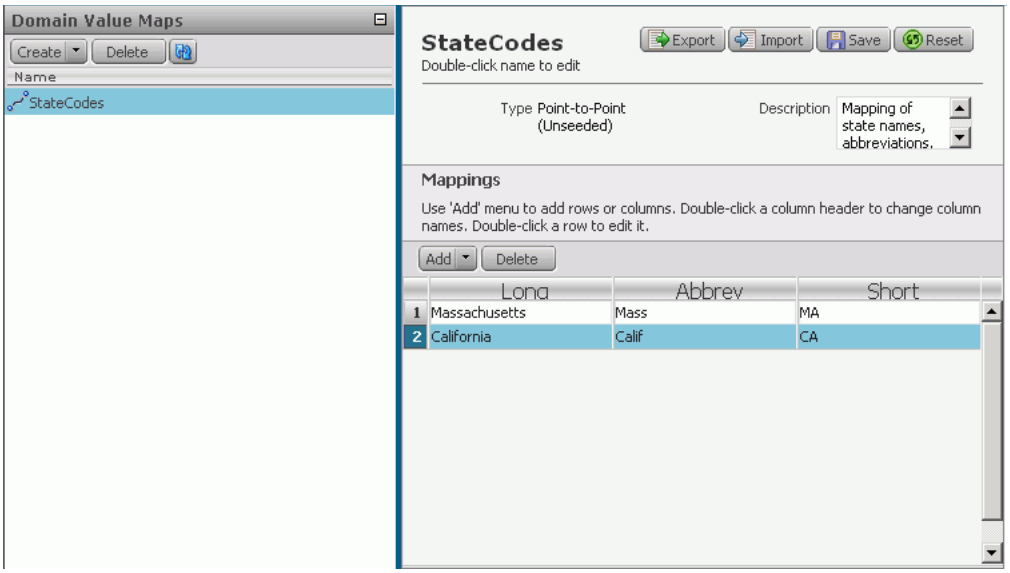
    You can change the name of a saved domain-value map by double-clicking on the map name, but any transformation map referring to the domain-value map would need to modified. See "Using a Domain-Value Map in a Transformation" on page 7-12.

    Although these instructions indicate that you click Save when you have completed a domain-value map, you can click Save at any point in the process.

12. If you want to export the map, see "Exporting a Domain-Value Map" on page 7-4.

The domain-value map can now be used when creating a transformation in Oracle JDeveloper.

Figure 7–2 Domain-Value Map – Rows Created



Exporting a Domain-Value Map

After you have created and saved a domain-value map, you can export it to a file. You might want to do this if you have created it on a test system and now want to export it so that you can import it on a production system.

To export a domain-value map, follow these steps:

1. In the Maps view of the Oracle ESB Control, create or import a domain value map, as described in "Creating a New Domain-Value Map from Scratch" on page 7-2 or "Importing an Existing Domain-Value Map File" on page 7-7.
2. Click **Save,** if you have not already done so.
3. Click **Export.**
4. Click **OK,** and then wait for the operating system dialog box to open that enables you to save the file to the local file system. If you are prompted whether to open or save the file, select **Save**.
5. In the save file dialog box, save the domain-value map file, such as `StateCodes.xml`, in the desired location.

After the map file has been exported and saved to the file system, you can edit the XML file with a text editor. However, make sure that you have carefully updated the file so that the file remains valid. See Example 7–1 for a sample domain-value map file.

Example 7–1 Sample Domain-Value Map File

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<dvm name="StateCodes" isNew="null">
<description>Mapping of state names, abbreviations, and codes</description>
  <columns>
    <column name="Long"/><column name="Abbrev"/><column name="Short"/>
  </columns>
  <rows>
    <row><cell>Massachusetts</cell><cell>Mass</cell><cell>MA</cell></row>
    <row><cell>California</cell><cell>Calif</cell><cell>CA</cell></row>
  </rows>
</dvm>
```

Domain-Value Map Template and XSD Files

This section provides examples of the domain-value map template and schema definition file.

Example 7–2 shows the domain-value map template (XML) file. This file should be used if you are creating a domain-value map file from scratch with a text editor.

Example 7–2 Domain-Value Map Template File

```
<?xml version="1.0" encoding="UTF-8"?>
<dvm name="New DVM" isNew="true">
  <description>DVM description</description>
  <columns>
    <column name="Domain1"/>
    <column name="Domain2"/>
  </columns>
  <rows>
    <row>
       <cell>value1</cell>
       <cell>value2</cell>
    </row>
  </rows>
</dvm>
```

Example 7–3 shows the domain-value map schema definition (XSD) file. All imported domain-value map XML files are validated against this schema definition file.

Example 7–3 Domain-Value Map XSD File

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Copyright (c) 2006, Oracle. All rights reserved.  -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="dvm">
    <xsd:annotation>
      <xsd:documentation>The Top Level Element
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="description" minOccurs="0">
          <xsd:annotation>
            <xsd:documentation>The DVM Description. This is optional
            </xsd:documentation>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="columns">
          <xsd:annotation>
            <xsd:documentation>This element holds DVM's column List.
            </xsd:documentation>
          </xsd:annotation>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="column" minOccurs="2" maxOccurs="unbounded">
                <xsd:annotation>
                  <xsd:documentation>This represents a DVM Column
                  </xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                  <xsd:attribute name="name" use="required"/>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="rows">
          <xsd:annotation>
            <xsd:documentation>This represents all the DVM Rows.
            </xsd:documentation>
          </xsd:annotation>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="row" minOccurs="1" maxOccurs="unbounded">
                <xsd:annotation>
                  <xsd:documentation>
                    Each DVM row of values
                  </xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="cell" minOccurs="2" maxOccurs="unbounded">
                      <xsd:annotation>
                        <xsd:documentation>This is the value for this row and for each column in
                                          the same order as defined in Columns.
                        </xsd:documentation>
                      </xsd:annotation>
                    </xsd:element>
                  </xsd:sequence>
                </xsd:complexType>
```

```
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="name" use="required"/>
    <xsd:attribute name="isNew"/>
  </xsd:complexType>
</xsd:element>
<xsd:annotation>
  <xsd:documentation>This schema is used to validate the DVM Document used for creating
                     and updating a domain-value map
  </xsd:documentation>
</xsd:annotation>
</xsd:schema>
```

Importing an Existing Domain-Value Map File

If you have an existing domain-value map on the local file system, you can import it into Oracle ESB Control as a means to add it to the ESB repository so that it can be used when you are specifying a transformation in a routing rule.

Typical scenarios for importing an existing domain-value map file are the following:

- To move a domain-value map from a test system to a production system

  If you need to move an entire ESB configuration from a test system to a production system, Oracle recommends you use the import method described in "Moving the ESB Instance to a Different Oracle Home" on page 9-13. To import a domain-value map only, first export it on the test system, as described in "Exporting a Domain-Value Map" on page 7-4.

- To import an XML lookup file exported from a database as the basis for a domain-value map.

  This scenario assumes that you exported the schema from the database using a tool such as XSU, and have converted the exported file to use the schema required for ESB domain-value maps. See "Domain-Value Map Template and XSD Files" on page 7-5. For an example of an exported domain-value map, see Example 7–1 on page 7-5.

To import an existing domain-value map file and store it in the ESB repository, follow these steps:

1. At the top of the Oracle ESB Control, click **Maps.**
2. Click the Create down-arrow on the left side of the Oracle ESB Control.

   Two choices are presented: **Create a new map** and **Import a new map.**
3. Click **Import a new map.**

   The Import a New Map dialog box opens. See Figure 7–3.
4. In the Import a New Map dialog box, follow these steps:
   a. In the Import field, enter the complete path for the file on the local file system that you want to import.
   b. Select or deselect **overwrite if a map with the same name already exists**, as desired.

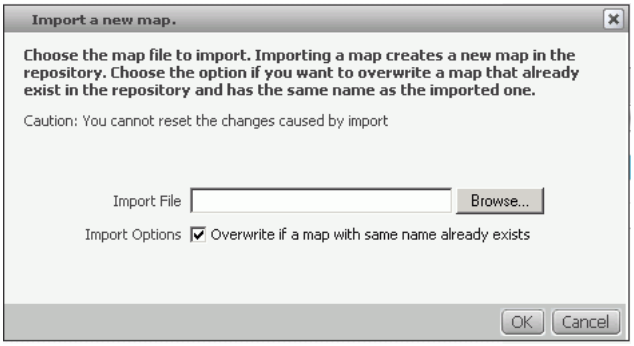
Because the name becomes the primary key for the domain-value map when it is stored in the ESB repository, the name must be unique. If you select **overwrite if a map with the same name already exists** and a domain-value map with the same name already exists that you do not want to overwrite, you can do either of the following:

- Cancel this dialog box, rename the domain-value map, and then restart the import operation. See "Editing the Name of a Domain-Value Map" on page 7-10.
- Deselect **overwrite if a map with the same name already exists.**

5. Click **OK.**

   The Oracle ESB Control refreshes and presents the imported domain-value map file. The imported file is saved to the ESB repository and the domain-value map can now be used when creating a transformation in Oracle JDeveloper.

Figure 7–3 Maps View – Import a New Map Dialog



Importing Rows Into a Domain-Value Map

If you are working on a domain-value map that currently has several rows and then want to import additional rows from a properly formatted file, you can do so as described in the list that follows. When you do so, rows from the imported file may conflict with rows in the current domain value map.

Rows are considered to conflict, when a given column contains the same value in two or more different rows, such as shown for the Oracle application in Figure 7–4.

Figure 7–4 Sample Domain-Value Map with a Conflict

When such a conflict exists you are given the following options:

- Overwrite conflicting rows

  If you select this option, conflicting rows in the domain-value map displaying in the Oracle ESB Control are overwritten with rows from the file being imported.

- Skip conflicting rows

If you select this option, conflicting rows in the domain-value map displaying in the Oracle ESB Control are not overwritten with rows from the file being imported. Those rows are not imported into the domain-value map.

- Add conflicting rows

  If you select this option, rows from the file being imported are added to the domain-value map, even if they conflict with existing rows in the domain value map. You should only select this option under particular circumstances and if you are familiar with the applications that will use the domain-value map at runtime.

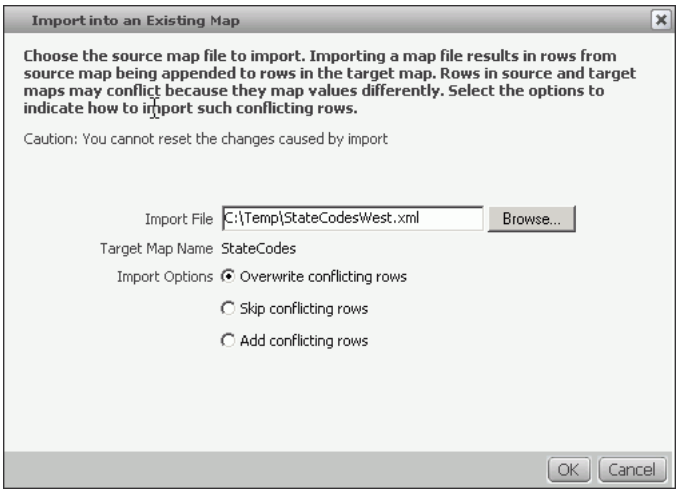
  For example, the rows presented in Figure 7–4, should remain only if the domain-value map will be used to map values from the SAP application values to the Oracle application at run time, and not the reverse. If values from the Oracle application will map to the SAP application at run time, then it is impossible for the ESB Server to determine whether, Gallon should map to Container or Gal, and therefore an error will result.

Follow these steps to import rows into a domain-value map:

1. In the Maps view of the Oracle ESB Control, create or import a domain value map, as described in "Creating a New Domain-Value Map from Scratch" on page 7-2 or "Importing an Existing Domain-Value Map File" on page 7-7.
2. Click **Save** to save any changes you have made.
3. Click **Import.**

   The Import into an Existing Map dialog box opens.
4. In the Import File field, enter the path for the file from which you want to import rows.
5. Select one of the **Import Options,** which are described in section.

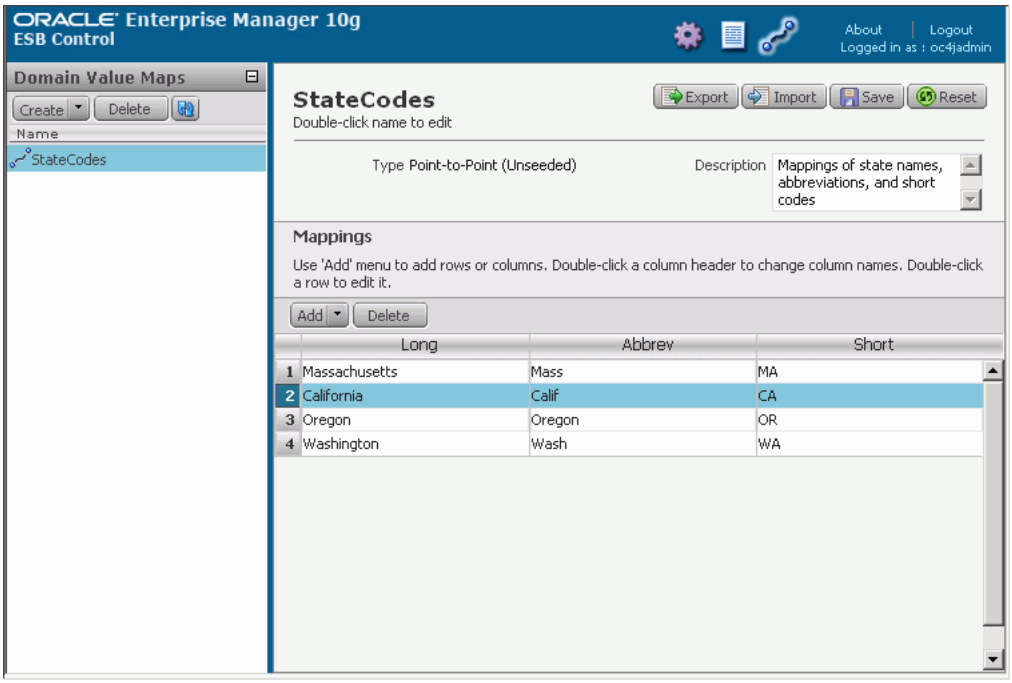
Figure 7–5 Oracle ESB Control Map View – Import into an Existing Map Dialog



6. Click **OK.**

The Oracle ESB Control refreshes the view and presents the imported rows. The domain-value map is saved to the ESB repository and it can be used when creating a transformation in Oracle JDeveloper, as described in "Using a Domain-Value Map in a Transformation" on page 7-12.

Figure 7–6 Oracle ESB Control Map View – Updated Map After Importing Rows



Editing a Domain-Value Map

This section presents the editing options that are available for editing a domain-value map and making adjustments to the presentation of data in the Map view. All of the options described here can be used while you are creating a domain-value map.

This section contains the following topics:

- Editing the Name of a Domain-Value Map on page 7-10
- Adding Rows or Columns to a Domain-Value Map on page 7-11
- Deleting a Row from a Domain-Value Map on page 7-11
- Deleting a Column from a Domain-Value Map on page 7-11
- Renaming a Column in a Domain-Value Map on page 7-11
- Reordering the Columns in a Domain-Value Map on page 7-11
- Resetting a Domain-Value Map to Its Last Saved State on page 7-12
- Resizing Columns in a Domain-Value Map on page 7-12

Editing the Name of a Domain-Value Map

Follow these steps:

1. Double-click the domain-value map name.
2. Edit the map name.

You can change the name of a saved domain-value map by double-clicking on the map name, but any transformation map referring to the domain-value map would need to modified. See "Using a Domain-Value Map in a Transformation" on page 7-12.

Adding Rows or Columns to a Domain-Value Map

Follow these steps:

1. Click the Add down-arrow

   A drop-down list presents a choice of row or column.

2. Click row or column to add the desired table element. You can click the option multiple times to add multiple rows or columns.

Deleting a Row from a Domain-Value Map

Follow these steps:

1. Click the number of the row that you want to delete.

   The row is highlighted in blue.

2. Click the **Delete** button under the Mappings heading.

Deleting a Column from a Domain-Value Map

Follow these steps:

1. Click the name of the column that you want to delete.

   The column is highlighted in blue.

2. Click the **Delete** button under the Mappings heading.

Renaming a Column in a Domain-Value Map

Follow these steps:

1. Double-click the name of the column that you want to rename.

   The text cursor appears before the first letter of the existing column name.

2. Edit the column name.

   You can delete all the existing characters and type in the new name.

3. Click the **Save** button to save the changes.

Reordering the Columns in a Domain-Value Map

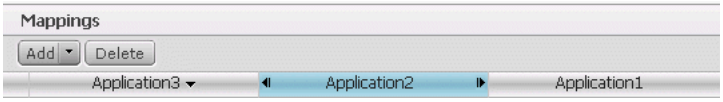
You can move a selected column one position at a time, as described in the following list. This options is provided to support user preferences, it has no effect on how the domain-value map is applied at run-time.

Follow these steps:

1. Click the column that you want to move to the right.

   Small arrows appears in the column heading, as shown in Figure 7–7.

Figure 7–7 Arrows for Rearranging Columns In Domain-Value Maps



2. Click the arrow in the direction you want to move the selected column.

3. Repeat steps 1 and 2 until all the columns appear in the desired order.

Resetting a Domain-Value Map to Its Last Saved State

As you are creating or editing a domain-value map, it is best to save it frequently, especially if you are adding numerous rows manually.

Suppose, for example, you save a domain value map, and then add multiple additional rows. You realize that entered many incorrect rows since the last time you saved the domain-value map. You can reset the domain-value map to its last saved state by clicking the **Reset** button. Any changes made after the last time you clicked the **Save** button are removed.

Resizing Columns in a Domain-Value Map

If the values you add to the cells in a domain value map are wider than the default cell width, you might want to resize the columns, by following these steps:

1. Place the cursor on the boundary between two column headers so that the cursor shape changes to a double-headed arrow; this is the resize cursor.
2. Left-click the double-headed arrow – a dotted line appears at the column boundary.
3. Drag the dotted line left or right, as desired, to indicate the new placement of the boundary.
4. Release the mouse button.

Deleting a Domain-Value Map

If you a domain-value map is no longer used in any routing rule transformations and you want to delete it, follow these steps:

1. Click **Maps** in the Oracle ESB Control.
2. In the Domain-Value Maps navigator, select the name of the domain-value map that you want to delete.
3. Click the **Delete** button within the Domain-Value Maps navigator.

   The Delete Map dialog box opens.
4. Click **Yes.**

The selected domain-value map is deleted from the Domain-Value Maps navigator and from the ESB repository.

Using a Domain-Value Map in a Transformation

To include a domain-value map in a transformation, you drag and drop a `lookup-dvm` function from the component palette onto the connection between the source and target elements where you want the domain-value map to be used.

Note: If you have previously used Oracle BPEL Process Manager, you may be familiar with the `lookup-xml` and `lookup-table` functions. Using these functions you can accomplish operations similar to using the Oracle Enterprise Service Bus `lookup-dvm` function. However, the `lookup-xml` and `lookup-table` functions are independent of the domain-value map infrastructure; a `lookup-table` function, for example, cannot be used with an ESB domain-value map.

---

1. Open an ESB project in Oracle JDeveloper.

   For example, open the **CustomerData** project to display the routing service icons in the Design tab. See Figure 2–2 on page 2-3 for a sample screenshot of the CustomerData project.

2. Select a routing icon in the Design tab and then double-click the transformation XSL icon to display the Data Mapper window.

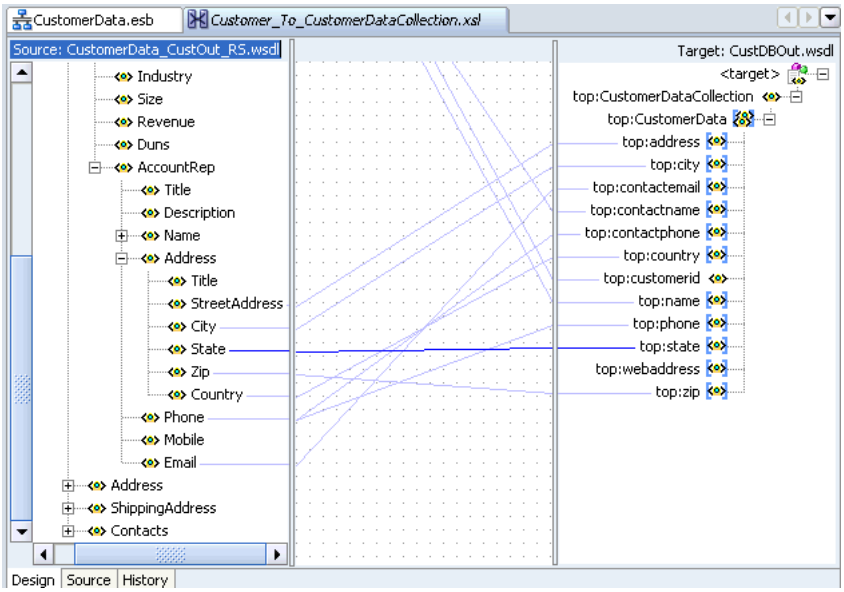
   For example, select **CustOut_RS** in the Design tab and double click the XSL icon.

3. Expand the trees in the Source and Target panes in the Data Mapper window.

4. Drag and drop the source element to the target element.

   For example, **State** to **top:state** as shown in Figure 7–8.

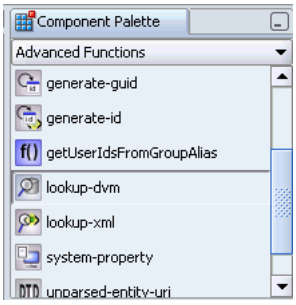
Figure 7–8 Data Mapper – Source to Target Map



5. In the Components palette, click the **down-arrow** and then select **Advanced Functions.**

   The `lookup-dvm` function is listed in the component palette, as shown in Figure 7–9.

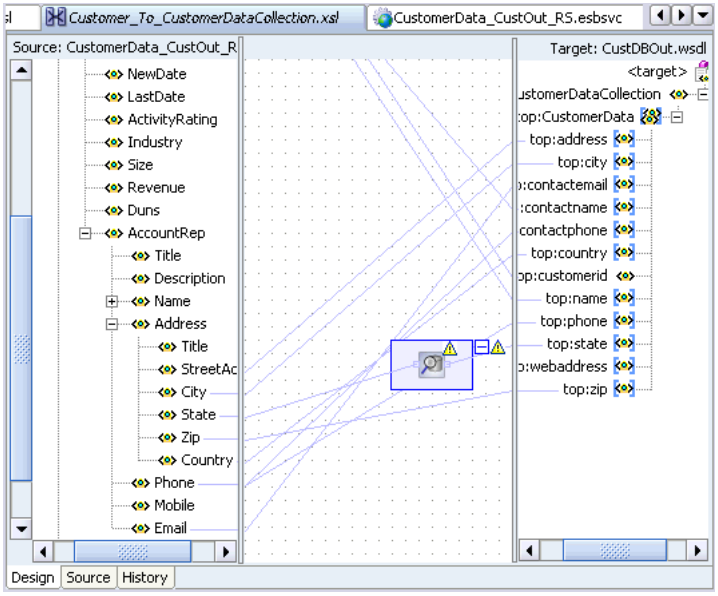
Figure 7–9 Component Palette – lookup-dvm Function



6. Drag and drop `lookup-dvm` onto the line that connects the source object to the target object.

A `lookup-dvm` icon appears on the connecting line, as shown in Figure 7–10. The yellow warning triangles indicate that the definition of the `lookup-dvm` function is not complete.

Figure 7–10 Data Mapper – look-up dvm Function Added

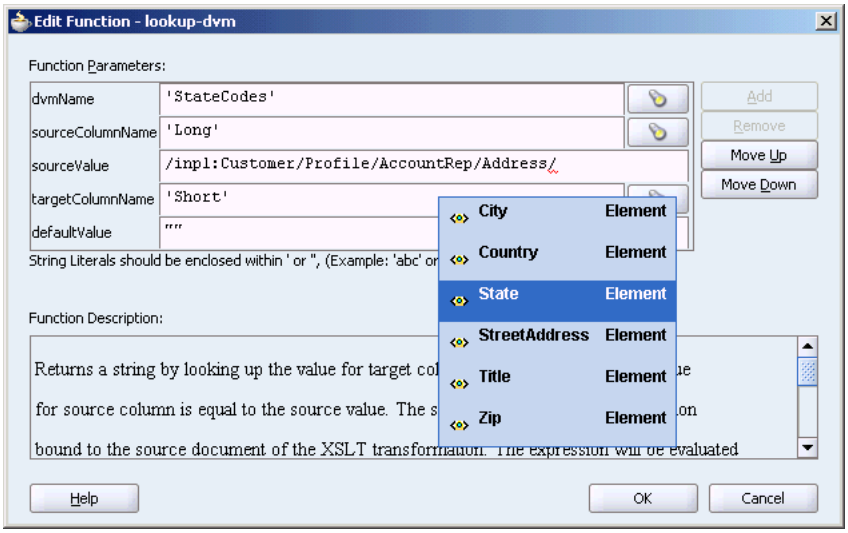


7. Double-click the **look-up dvm** icon.

   The Edit Function – look-up dvm dialog box opens.

8. Specify values for the fields in the Edit Function – look-up dvm dialog box.

Figure 7–11 Edit Function – lookup-dvm



   a. You can manually enter a value in **sourceValue** or the following methods:

      Move the source value that appears in the **dvmName** field to the **sourceValue** field by clicking **Move Down** twice.

Press **Ctrl-Space** to launch Expression Builder. Press the up and down keys to locate an object in the list and press enter to select that item.

b. In the **dvmName** field, enter the name of the domain-value map schema that you previously defined using the Oracle ESB Control.

Click the flashlight icon to the right of the **dvmName** field to select the name from a list of domain-value maps stored in the ESB repository.

For example, select `'StateCodes'` that was described in "Creating a New Domain-Value Map from Scratch" on page 7-2.

c. In the **sourceColumnName** field, enter the name of the column in the domain-value map that is associated with the source element value.

Click the flashlight icon to the right of the **sourceColumnName** field to select the name from the columns defined for the domain-value map you previously selected.

For example, `'Long'`

d. In the **targetColumnName** field, enter the name of the column in the domain-value map that is associated with the target element value.

Click the flashlight icon to the right of the **targetColumnName** field to select the name from the columns defined for the domain-value map you previously selected.

For example, `'Short'`

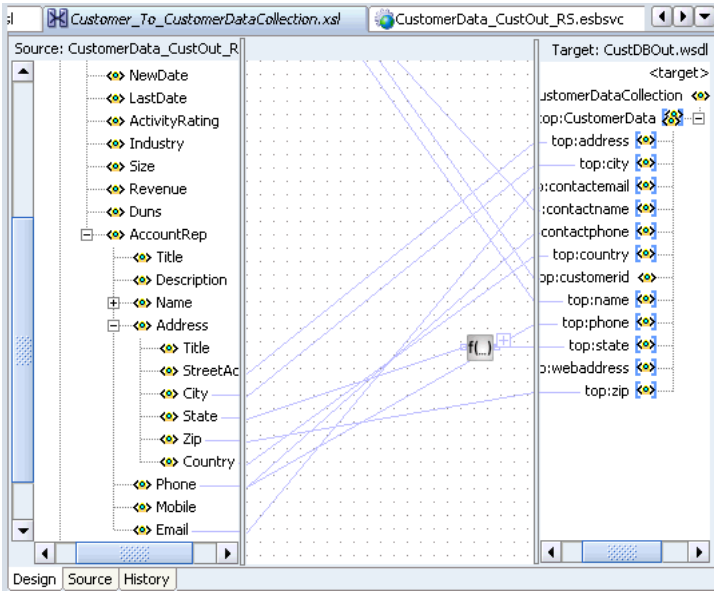
e. Enter an optional default value.

For example, `""`

f. Click **OK**.

The data mapper appears in the Design tab with the `lookup-dvm` function icon, as shown in Figure 7–12.

Figure 7–12 Data Mapper – look-up dvm Function Defined



9. From the **File** menu, click **Save All**.

10. Register the project with the ESB Server.

When the transformation is included in an ESB routing rule, the transformation is applied at run time. For example, if the **StatesCodes** domain-value map is used in the transformation, the transformation from the user-specified state value to the state code value required by the target application is applied at run time.

You can view the results of a transformation if you have set up the field in the Trackable Fields tab of the Oracle ESB Control Services view. The tracking data is displayed in the Details tab of the Oracle ESB Control Instances view, as shown in Figure 10–1 on page 10-2.

8

Creating Cross References

The cross referencing feature of Oracle Enterprise Service Bus enables you to associate identifiers for equivalent entities created in different applications. For example, you can use cross references to associate a customer entity created in one application (with native id `Cust_100`) with an entity for the same customer in another application (with native id `CT_001)`.

This chapter explains how to create, populate, and use cross references. It contains the following topics:

- Introduction to Cross References on page 8-1
- Creating, Modifying, and Deleting Cross Reference Table on page 8-4
- Populating Cross Reference Tables on page 8-5
- Looking Up Cross Reference Tables on page 8-12
- Deleting Cross Reference Table Values on page 8-15
- Importing and Exporting Cross References on page 8-17
- Schema Definition (XSD) File for Cross References on page 8-18

Introduction to Cross References

Many a time, when you create or update objects in one application, you also want to propagate the changes to another application. For example, when a new customer is created in a SAP application, you might want to create a new entry for the same customer in your Oracle E-Business Suite application named as `EBS`.

However, the applications that you are integrating could be using different entities to represent the same information. For example, for a new customer in a SAP application, a new row is inserted in its `Customer` database with a unique identifier such as `SAP_001`. When the same information is propagated to an Oracle E-Business Suite application and a Siebel application, a new row should be inserted with different identifiers, such as `EBS_1001` and `SBL001`. In such cases, you need some kind of functionality to map these identifiers with each other so that they could be interpreted by different applications to be referring to the same entity. This can be done by using cross references tables. Table 8–1 shows a cross reference table containing information about customer identifiers in different applications.

Table 8–1 Cross Reference Table Sample

SAP	EBS	SBL
SAP_001	EBS_1001	SBL001

Table 8–1 (Cont.) Cross Reference Table Sample

SAP	EBS	SBL
SAP_002	EBS_1002	SBL002

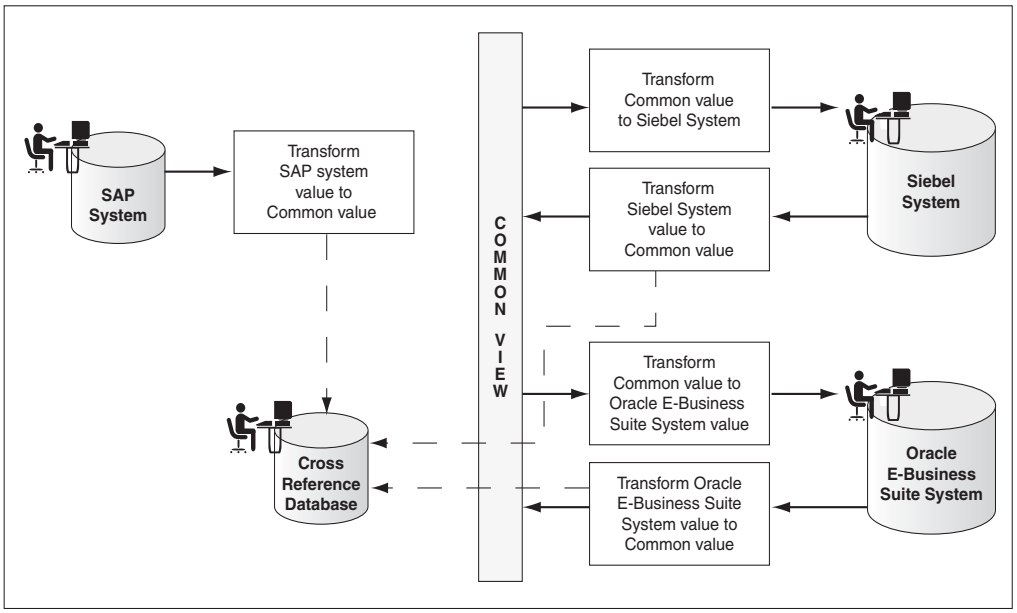
The identifier mapping is also required when information about a customer is updated in one application and the changes need to be propagated in other applications also. You can also integrate different identifiers by using a common value integration pattern, which maps to all identifiers in a cross reference table. For example, you can add one more column `Common` to the cross reference table shown in Table 8–1. The updated cross reference table would appear as shown in Table 8–2.

Table 8–2 Cross Reference Table with the Common Column

SAP	EBS	SBL	Common
SAP_001	EBS_1001	SBL001	CM001
SAP_002	EBS_1002	SBL002	CM002

Figure 8–1 shows how you can use the common value integration pattern to map identifiers in different applications.

Figure 8–1 Common Value Integration Pattern Example

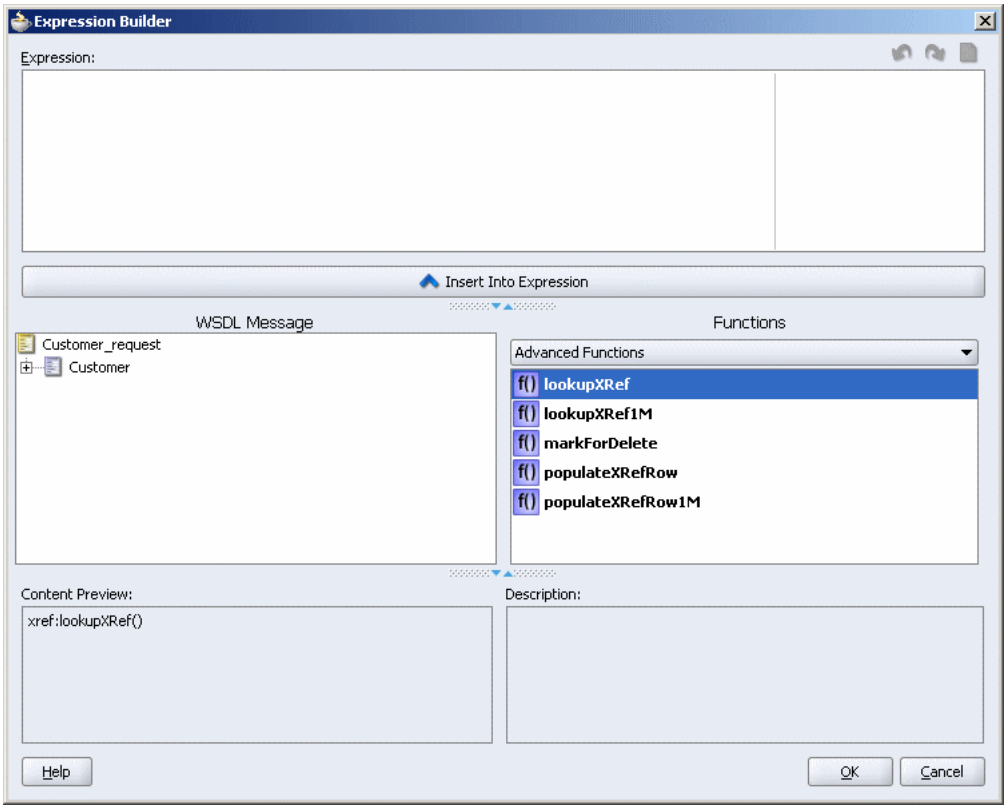


A cross reference table consists of following two parts, metadata and the actual data. The metadata is created by using the cross reference command line utilities and is stored in the repository as an XML file. The actual data is stored in the database.

You can use a cross reference table to look up column values at run time. However, before using a cross reference to look up a particular value, you need to populate it at run time. This can de done by using the cross reference XPath functions. The XPath functions enable you to populate a cross reference, perform lookups, and delete a column value. These XPath functions can be used in the Expression builder dialog box to create an expression or in the XSLT Mapper dialog box to create transformations.

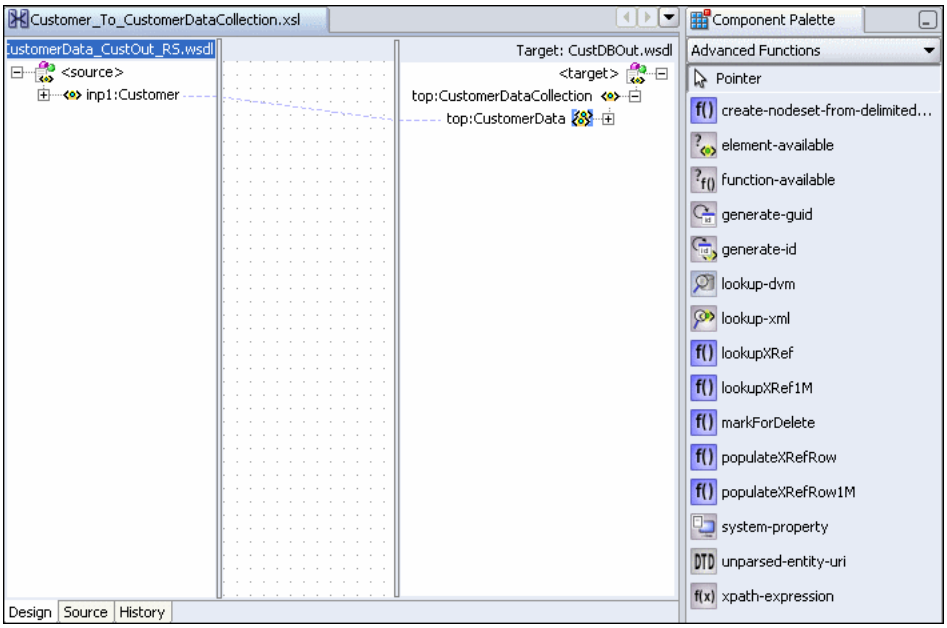
The Expression builder dialog box is displayed when you click the Invoke Expression Builder icon in the routing rules panel. Figure 8–2 shows how you can select the cross reference functions in the Expression builder dialog box.

Figure 8–2 Expression Builder Dialog Box with Cross Reference Functions



The XSLT Mapper dialog box is displayed when you create an XSL file to transform data from one XML schema to another. Figure 8–3 shows how you can select the cross reference functions in the XSLT Mapper dialog box.

Figure 8–3 XSLT Mapper Dialog Box with Cross Reference Functions



Introduction to the Cross Reference Command-Line Utility

Oracle Enterprise Service Bus provides a set of command line utilities that you can use for cross reference administration. The command line utilities are xreftool, xrefimport, and xrefexport.

Note: Before using these command line utilities, you need to start the Oracle SOA Suite Server.

The xrefimport and xrefexport utility enables you to import and export cross reference metadata and data.

The xreftool utility enables you to create cross reference metadata such as creating cross reference table and columns. However, you cannot populate the cross reference tables by using the xreftool commands. You can also use the xreftool commands to modify, and delete cross reference tables. To use the xreftool utility, perform the following steps:

1. Create two variables named `OC4J_USERNAME` and `OC4J_PASSWORD` as environment variables.
2. Start the command prompt.
3. At the prompt, type the following command:

   ```
   cd <ORACLE_HOME>\integration\esb\bin
   ```

4. At the prompt, type the following command:

   ```
   xreftool -shell
   ```

   This command starts the cross reference shell where you can run xreftool commands.

Table 8–3 lists various xreftool commands.

Table 8–3 xreftool Commands

Functionality	Command
Running multiple commands in a sequence	`xreftool` *`COMMAND1 ARGS1 COMMAND2 ARGS2`*
Running commands from a file	`xreftool -f` *`FILELOCATION`*
Viewing the description of all xreftool commands	`xreftool help`

Creating, Modifying, and Deleting Cross Reference Table

You can use the xreftool utility to create, modify, and delete cross reference tables. To create a cross reference table, use the following command in the cross reference shell:

```
createTable TableName
```

For example, the `createTable customers` command creates a cross reference table named `customers`:

Note: The table names and column names are not case-sensitive.

To view a list of all cross reference tables present in the repository, you can use the following xreftool command:

```
listTables
```

Modifying Cross Reference Tables

You can modify the cross reference tables by adding, and deleting columns. Table 8–4 lists various xreftool utility commands that you can use to modify a cross reference table.

Table 8–4 xreftool Commands for Modifying a Cross Reference Table

Functionality	Command	Example
Adding a column to a cross reference table	`addColumns` *`XREFTABLENAME COLUMNNAME`*	`addColumns orders sap`
Adding multiple columns to a cross reference table	`addColumns` *`XREFTABLENAME COLUMNNAME1,COLUMNNAME2, COLUMNNAME3`*	`addColumns orders sap,siebel`
Deleting a column When you delete a column, the data corresponding to the column is deleted from the database.	deleteColumn *`XREFTABLENAME, COLUMNNAME`*	`deleteColumn orders sap`
Viewing all columns of a cross reference table	`listColumns` *`XREFTABLENAME`*	`listColumns orders`

Deleting a Cross Reference Table

You can delete a cross reference table by using the following command:

```
deleteTable TABLENAME
```

For example:

```
deleteTable orders
```

When you delete a table, the data corresponding to the table is deleted from the database.

Populating Cross Reference Tables

A cross reference table needs to be populated at run time before being used. This can be done by using the following XPath extension functions:

- xref:populateXRefRow Function
- xref:populateXRefRow1M Function

xref:populateXRefRow Function

You can use the `xref:populateXRefRow` function to populate a cross reference column with a value. This function returns a string value which is the cross reference value being populated. The syntax of the `xref:populateXRefRow` function is as follows:

```
xref:populateXRefRow(xrefTableName as string, xrefReferenceColumnName as string,
 xrefReferenceValue as string, xrefColumnName as string, xrefValue as string, mode
 as string) as string
```

Parameters

- `xrefTableName`: The name of the cross reference table.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to the reference column name.
- `xrefColumnName`: The name of the column to be populated.
- `xrefValue`: The value to be populated in the column.
- `mode`: The mode in which the `xref:populateXRefRow` function populates the column. You can specify any of the following values: `ADD`, `LINK`, or `UPDATE`. Table 8–5 describes these modes.

Table 8–5 xref:populateXRefRow Function Modes

Mode	Description	Exception Reasons
ADD	Adds the reference value and the value to be added. For example, `xref:populateXRefRow("customers","SAP","SAP_100","Common","CM001","ADD")` adds the reference value `SAP_100` in the `SAP` reference column and value `CM001` in the `Common` column.	Exceptions can occur due to the following reasons: ■ The specified cross reference table is not found. ■ The specified columns are not found. ■ The values provided are empty. ■ The value being added is not unique across that column for that table. ■ The column for that row already contains a value. ■ The reference value exists.
LINK	Adds the cross reference value corresponding to the existing reference value. For example, `xref:populateXRefRow("customers","SAP","SAP_100","Common","CM001","Link")` links the value `CM001` in the `Common` column to the `SAP_100` value in the `SAP` column.	Exceptions can occur due to the following reasons: ■ The specified cross reference table is not found. ■ The specified columns are not found. ■ The values provided are empty. ■ The reference value is not found. ■ The value being linked exists in that column for that table.
UPDATE	Updates the cross reference value corresponding to an existing reference column-value pair. For example, `xref:populateXRefRow("customers","SAP","SAP_100", "SAP","SAP_1001","Update")` updates the value `SAP_100` in the `SAP` column to value `SAP_1001`.	Exceptions can occur due to the following reasons: ■ The specified cross reference table is not found. ■ The specified columns are not found. ■ The values provided are empty. ■ Multiple values are found for the column being updated. ■ The reference value is not found. ■ The column for that row does not have a value.

> **Note:** The `mode` parameter values are case-sensitive and should be specified in the upper case only as shown in Table 8–5.

Table 8–6 describes the `xref:populateXRefRow` function modes and exception conditions for these modes.

Table 8–6 xref:populateXRefRow Function Results with Different Modes

Mode	Reference Value	Value to be Added	Result
ADD	Absent	Absent	Success
	Present	Absent	Exception
	Present	Present	Exception
LINK	Absent	Absent	Exception
	Present	Absent	Success
	Present	Present	Exception
UPDATE	Absent	Absent	Exception
	Present	Absent	Exception
	Present	Present	Success

Using the xref:populateXRefRow Function

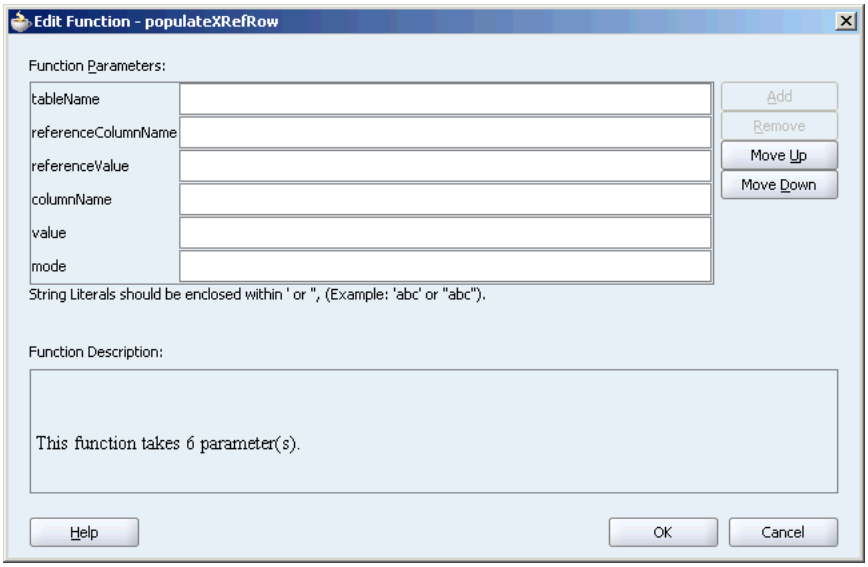
The `xref:populateXRefRow` function can be used in transformation to populate a column of a cross reference table by performing the following steps:

1. In the XSLT Mapper dialog box, expand the trees in the Source and Target panes.
2. Drag and drop the source element to the target element.
3. In the Component Palette, click the down arrow list and then select **Advanced Functions**.
4. Drag and drop **populateXRefRow** onto the line that connects the source object to the target object.

   A populateXRefRow icon appears on the connecting line.
5. Double-click the **populateXRefRow** icon.

   The Edit Function – populateXRefRow dialog box is displayed, as shown in Figure 8–4.

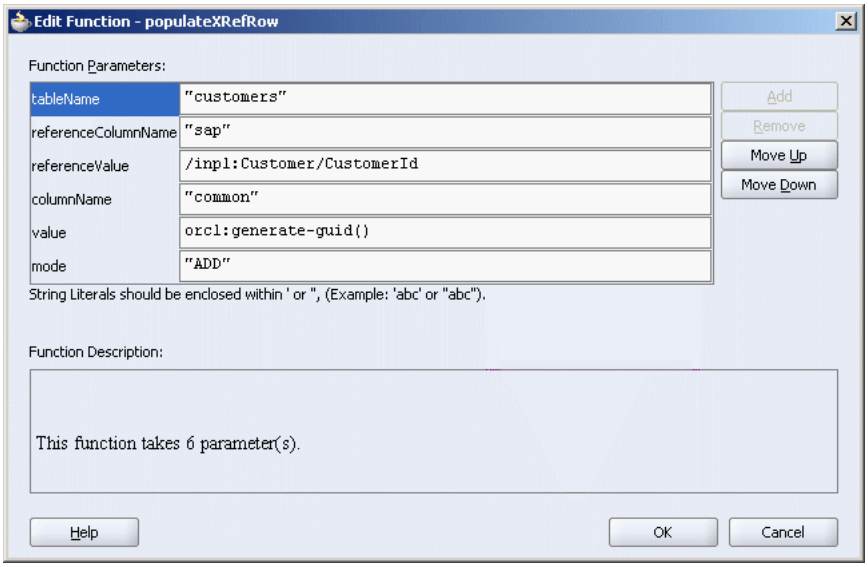
Figure 8–4 The Edit Function – populateXRefRow Dialog Box



6. Specify the following values for the fields in the Edit Function – populateXRefRow dialog box:
   a. In the **tableName** field, enter the name of the cross reference table.
   b. In the **referenceColumnName** field, enter the name of the cross reference column.
   c. In the **referenceValue** field, you can manually enter a value or press **Ctrl-Space** to launch XPath Building Assistant. Press the Up and Down arrow keys to locate an object in the list and press Enter to select that object.
   d. In the **columnName** field, enter the name of the cross reference column.
   e. In the **value** field, you can manually enter a value or press **Ctrl-Space** to launch the XPath Building Assistant.
   f. In the **mode** field, enter a mode in which you want to populate the cross reference table column, for example, `ADD`.
7. Click **OK**.

   A populated Edit Function – populateXRefRow dialog box is shown in Figure 8–5.

Figure 8–5 The Populated Edit Function – populateXRefRow Dialog Box



xref:populateXRefRow1M Function

Many a time, two values in a system can correspond to a single value in another system. For example, as shown in Table 8–7, the `SAP_001` and `SAP_0011` values refer to one value of the EBS and the SBL application.

Table 8–7 A Cross Reference Table with Multiple Column Values

SAP	EBS	SBL
SAP_001	EBS_1001	SBL001
SAP_0011		
SAP_002	EBS_1002	SBL002

To populate a column in the cross reference table with multiple values, you can use the `xref:populateXRefRow1M` function. The syntax of the `xref:populateXRefRow1M` function is as follows:

```
xref:populateXRefRow1M(xrefTableName as string, xrefReferenceColumnName as string,
 xrefReferenceValue as string, xrefColumnName as string, xrefValue as string, mode
 as string) as string
```

Parameters

- `xrefTableName`: The name of the cross reference table.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to reference column name.
- `xrefColumnName`: The name of the column to be populated.
- `xrefValue`: The value to be populated in the column.
- `mode`: The mode in which the `xref:populateXRefRow` function populates the column. You can specify either of the two values, `ADD` or `LINK`. Table 8–8 describes these modes:

Table 8–8 xref:populateXRefRow1M Function Modes

Mode	Description	Exception Reasons
ADD	Adds the reference value and the value to be added. For example, `xref:populateXRefRow1M("customers","SAP","SAP_100","Common","CM002","ADD")` adds the reference value `SAP_100` in the reference column `SAP` and the value `CM002` in the `Common` column.	Exceptions can occur due to the following reasons: ▪ The specified cross reference table is not found. ▪ The specified columns are not found. ▪ The values provided are empty. ▪ The value being added is not unique across that column for that table. ▪ The reference value exists.
LINK	Adds the cross reference value corresponding to the existing reference value. For example, `xref:populateXRefRow1M("customers","SAP","SAP_100","Common","CM001","Link")` links the value `CM001` in the `Common` column to the `SAP_100` value in the `SAP` column.	Exceptions can occur due to the following reasons: ▪ The specified cross reference table is not found. ▪ The specified columns are not found. ▪ The values provided are empty. ▪ The reference value is not found. ▪ The value being added is not unique across the column for that table.

Table 8–9 describes the `xref:populateXRefRow1M` function modes and exception conditions for these modes.

Table 8–9 xref:populateXRefRow1M Function Results with Different Modes

Mode	Reference Value	Value to be Added	Result
ADD	Absent	Absent	Success
	Present	Absent	Exception
	Present	Present	Exception
LINK	Absent	Absent	Exception
	Present	Absent	Success
	Present	Present	Exception

The design-time steps for using the `xref:populateXRefRow1M` function are similar to the `xref:populateXRefRow` function described in "Using the xref:populateXRefRow Function" on page 8-8.

Looking Up Cross Reference Tables

After populating the cross reference table, you can use it to look up for a value. This can be done by using the following XPath extension functions:

- xref:lookupXRef Function
- xref:lookupXRef1M Function

xref:lookupXRef Function

You can use the `xref:lookupXRef` function to look up a cross reference column for a value that corresponds to a specific value in a reference column. For example, the following function looks up the `Common` column of the cross reference table described in Table 8–2 for a value corresponding to the `SAP_001` value in the `SAP` column.

```
xref:lookupXRef("customers","SAP","SAP_001", "Common", true())
```

The syntax of the `xref:lookupXRefRow` function is as follows:

```
xref:lookupXRef(xrefTableName as string, xrefReferenceColumnName as string,
 xrefReferenceValue as string, xrefColumnName as string, needAnException as
 boolean) as string
```

Parameters

- `xrefTableName`: The name of the cross reference table.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to the reference column name.
- `xrefColumnName`: The name of the column to be looked up for the value.
- `needAnException`: Specify `true` or `false`.

  If `needAnException` parameter is set to `true`, an exception is thrown if the value, being looked up in the table, is not found. If `needAnException` parameter is set to `false`, an empty value is returned if the value, being looked up in the table, is not found.

Exception Reasons

An exception can occur because of the following reasons:

- The cross reference table with the given name is not found.
- The specified column names are not found.
- The specified reference value is empty.
- Multiple values are found.

Using the xref:lookupXRef Function

You can use the `xref:lookupXRef` function to look up a cross reference table column by performing the following steps during transformation:

1. In the XSLT Mapper dialog box, expand the trees in the Source and Target panes.
2. Drag and drop the source element to the target element.
3. In the Component Palette, click the down arrow list and then select **Advanced Functions**.

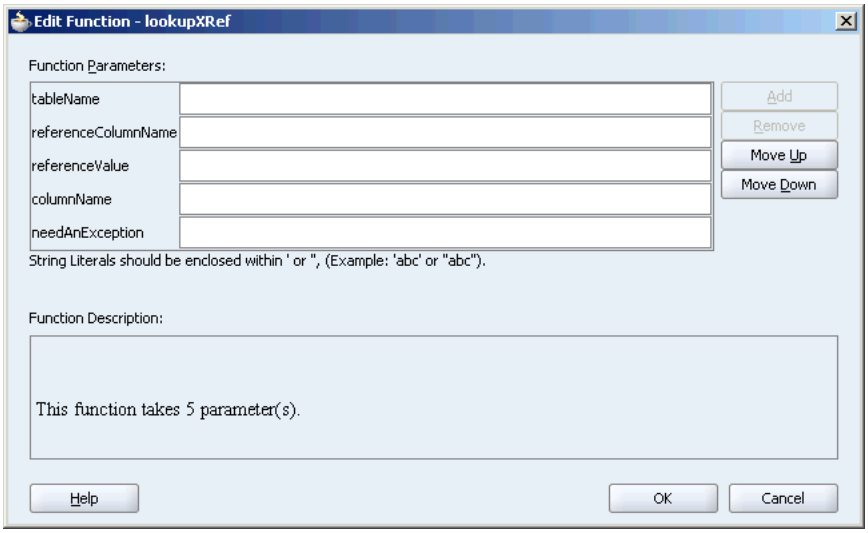
4. Drag and drop **lookupXRef** onto the line that connects the source object to the target object.

   A lookupXRef icon appears on the connecting line.

5. Double-click the **lookupXRef** icon.

   The Edit Function – lookupXRef dialog box is displayed, as shown in Figure 8–6.

Figure 8–6 The Edit Function – lookupXRef Dialog Box



6. Specify the following values for the fields in the Edit Function – lookupXRef dialog box:

   a. In the **tableName** field, enter the name of the cross reference table.

   b. In the **referenceColumnName** field, enter the name of the cross reference column.

   c. In the **referenceValue** field, you can manually enter a value or press **Ctrl-Space** to launch the XPath Building Assistant. Press the Up and Down arrow keys to locate an object in the list and press Enter to select that object.

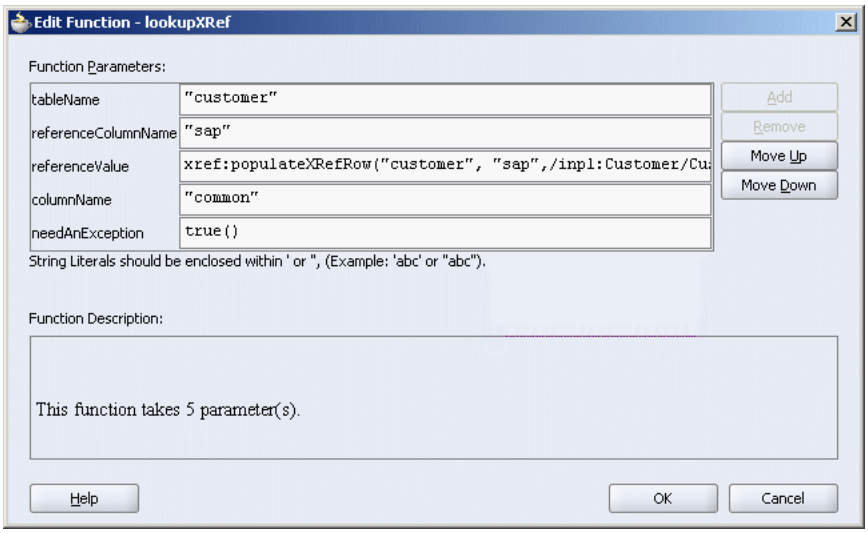
   d. In the **columnName** field, enter the name of the cross reference column.

   e. In the **needException** field, enter **Yes** to raise an exception if no value is found, else enter **No**.

7. Click **OK**.

   A populated Edit Function – lookupXRef dialog box is shown in Figure 8–7.

Figure 8–7 Populated Edit Function – lookupXRef Dialog Box



xref:lookupXRef1M Function

You can use the `xref:lookupXRef1M` function to look up a cross reference column for multiple values corresponding to a specific value in a reference column. This function returns a node-set containing the multiple nodes. Each node in the node-set contains a value.

For example, the following function looks up the `SAP` column of Table 8–7 for multiple values corresponding to `EBS_1001` value in the `EBS` column:

```
xref:lookupXRef1M("customers","EBS","EBS_1001", "Common", true())
```

The syntax of the `xref:lookupXRefRow1M` function is as follows:

```
xref:lookupXRef1M(xrefTableName as string, xrefReferenceColumnName as string,
 xrefReferenceValue as string, xrefColumnName as string, needAnException as
 boolean) as node-set
```

Parameters

- `xrefTableName`: The name of the cross reference table.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to the reference column name.
- `xrefColumnName`: The name of the column to be looked up for the value.
- `needAnException`: Specify `true` or `false`.

  If `needAnException` parameter is set to `true`, an exception is thrown if the value, being looked up in the table, is not found. If `needAnException` parameter is set to `false`, an empty value is returned if the value, being looked up in the table, is not found.

Exception Reasons

An exception can occur because of the following reasons:

- The cross reference table with the given name is not found.
- The specified column names are not found.

- The specified reference value is empty.

The design-time steps for using the `xref:lookupXRef1M` function are similar to the `xref:lookupXRef` function explained in "Using the xref:lookupXRef Function" on page 8-12.

Deleting Cross Reference Table Values

You can use the `xref:markForDelete` function to delete a value in a cross reference table. The value in the column is marked as deleted. This function returns `true` if deletion was successful, else it returns `false`.

A cross reference table row should have at least two mappings. Therefore, if you have only two mappings in a row and you mark one value for deletion, then the value in the other column is also deleted.

Any column value marked for deletion is treated as if the value does not exist. Therefore, you can populate the same column with the `xref:populateXRefRow` function in the `ADD` mode.

However, using the column value marked for deletion as a reference value in the `LINK` mode of the `xref:populateXRefRow` function, would raise an error.

The syntax for the `xref:markForDelete` function is as follows:

```
xref:markForDelete(xrefTableName as string, xrefColumnName as string,
xrefValueToDelete as string) return as boolean
```

Parameters

- `xrefTableName`: The name of the cross reference table.
- `xrefColumnName`: The name of the column from which you want to delete a value.
- `xrefValueToDelete`: The value to be deleted.

Exception Reasons

An exception can occur due to the following reasons:

- The cross reference table with the given name is not found.
- The specified column name is not found.
- The specified value is empty.
- The specified value is not found in the column.
- Multiple values are found.

Perform the following steps to delete a value from a cross reference table column:

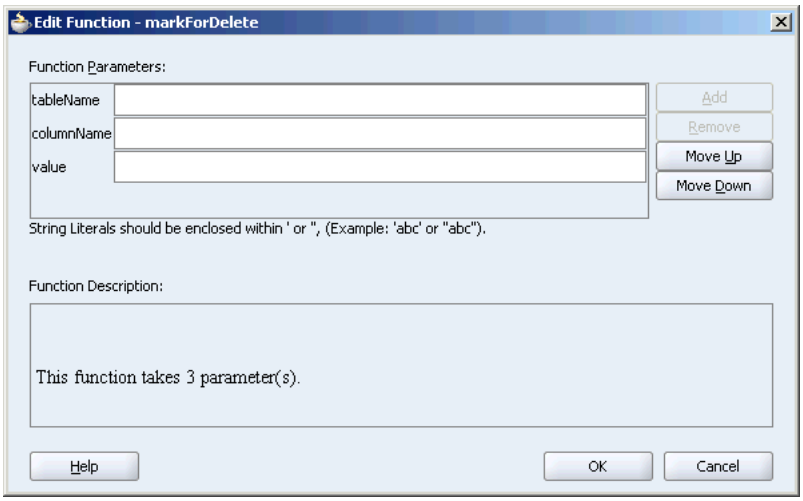
1. In the XSLT Mapper dialog box, expand the trees in the Source and Target panes.
2. Drag and drop the source element to the target element.
3. In the Component Palette, click the down arrow list and then select **Advanced Functions**.
4. Drag and drop **markForDelete** onto the line that connects the source object to the target object.

   A markForDelete icon appears on the connecting line.

5. Double-click the **markForDelete** icon.

   The Edit Function – markForDelete dialog box is displayed, as shown in Figure 8–8.

Figure 8–8 Edit Function – markForDelete Dialog Box



6. Specify the following values for the fields in the Edit Function – markForDelete dialog box:

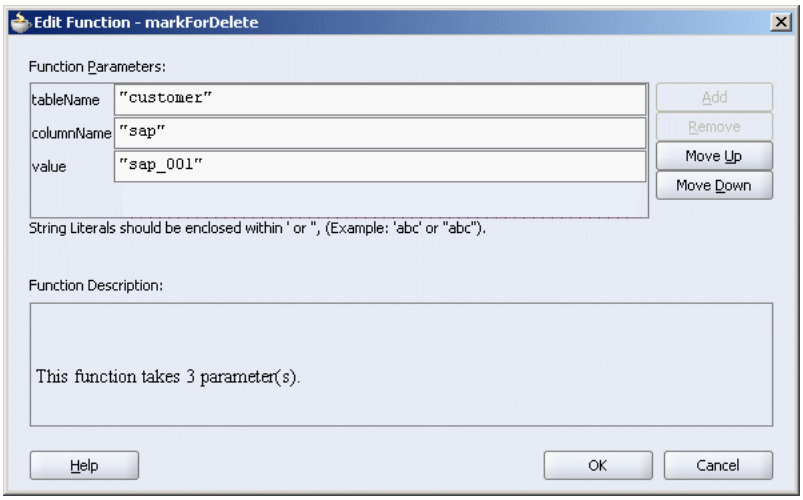
   a. In the **tableName** field, enter the name of the cross reference table.

   b. In the **columnName** field, enter the name of the column.

   c. In the **value** field, you can manually enter a value or press **Ctrl-Space** to launch the XPath Building Assistant. Press the Up and Down arrow keys to locate an object in the list and press Enter to select that object.

   A populated Edit Function – markForDelete dialog box is shown in Figure 8–9.

Figure 8–9 Populated Edit Function – markForDelete Dialog Box



7. Click **OK**.

Importing and Exporting Cross References

You can import and export the cross reference tables by using the xrefimport and xrefexport utilities. However, before using the xrefimport and xrefexport utilities, you need to create the following environment variables:

- `DB_URL`: Contains the connection string of the database.

  Example: `jdbc:oracle:thin:@stapm21.us.oracle.com:1521:orcl`

- `DB_USER`: The user name of the schema where the cross reference tables are created.

  Example: `DB_PASSWORD=oraesb`

- `DB_PASSWORD`: The password associated with the user specified in the `DB_USER` variable.

  Example: `DB_USER=oraesb`

> **Note:** If you have not created `OC4J_USERNAME` and `OC4J_PASSWORD` environment variables earlier, then you also need to create these environment variables.

The following sections explain how to import and export cross reference tables:

- Exporting Cross Reference Tables
- Importing Cross Reference Tables

Exporting Cross Reference Tables

The xrefexport utility enables you to export a cross reference table metadata along with the values. The exported data is stored in an XML file which is based on the schema defined in "Schema Definition (XSD) File for Cross References" on page 8-18. If the table that you are exporting contains columns without any values, then the missing column values are replaced with an empty `cell` element in the exported XML file.

To export a cross reference table, use the following command:

```
xrefexport -file FILENAME -table TABLENAME
```

The `FILENAME` and `TABLENAME` parameters are mandatory. The `FILENAME` parameter specifies the location of the file to which the data will be exported. The `TABLENAME` parameter specifies the name of the cross reference table to be exported.

Importing Cross Reference Tables

The xrefimport utility enables you to import a cross reference table metadata from an XML file. The XML file that you are importing should be based on the schema defined in "Schema Definition (XSD) File for Cross References" on page 8-18.

To import cross reference metadata, use the following command:

```
xrefimport -file FILENAME [-mode <ignore | overwrite>] [ -generate <columnName>]
```

The following list explains the various parameters of the xrefimport function:

- `FILENAME`: The `FILENAME` parameter specifies the location of the file from which the data will be imported.

- `mode`: The `mode` parameter specifies how the conflicts with existing data will be handled. The `mode` parameter can consist of one of the two values, `ignore` or `overwrite`. The value `ignore` specifies that the existing data should be kept in the repository. The value `overwrite` specifies that existing data should be overwritten with the data present in the XML file. The mode parameter is optional and is used only in case of a conflict. The default value is `ignore`.
- `generate`: The `generate` parameter can be used to create a column automatically while importing the metadata. For example, `generate SAP` creates a `SAP` column automatically while importing the metadata.

Schema Definition (XSD) File for Cross References

Example 8–1 shows the cross reference XSD file. All imported cross reference XML files are validated against this schema definition file. All functions in the schema definition file should be in the following namespace:

```
http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions
```

Example 8–1 Cross Reference XSD File

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://xmlns.oracle.com/xref"
        xmlns:tns="http://xmlns.oracle.com/xref" elementFormDefault="qualified">
  <element name="xref" type="tns:xrefType"/>
  <complexType name="xrefType">
    <sequence>
      <element name="table">
        <complexType>
          <sequence>
            <element name="columns" type="tns:columnsType" minOccurs="0"
                     maxOccurs="1"/>
            <element name="rows" type="tns:rowsType" maxOccurs="1"
                     minOccurs="0"/>
          </sequence>
          <attribute name="name" use="required">
            <simpleType>
             <restriction base="string">
              <minLength value="1"/>
             </restriction>
            </simpleType>
  </attribute>
        </complexType>
      </element>
    </sequence>
  </complexType>

  <complexType name="columnsType">
    <sequence>
      <element name="column" minOccurs="1" maxOccurs="unbounded">
        <complexType>
          <attribute name="name" use="required">
            <simpleType>
             <restriction base="string">
               <minLength value="1"/>
             </restriction>
            </simpleType>
          </attribute>
```

```
      </complexType>
    </element>
  </sequence>
</complexType>

<complexType name="rowsType">
  <sequence>
    <element name="row" minOccurs="1" maxOccurs="unbounded">
      <complexType>
        <sequence>
          <element name="cell" minOccurs="1" maxOccurs="unbounded">
            <complexType>
              <simpleContent>
                <extension base="string">
                  <attribute name="colName" use="required">
                    <simpleType>
                      <restriction base="string">
                         <minLength value="1"/>
                      </restriction>
                    </simpleType>
                  </attribute>
                </extension>
              </simpleContent>
            </complexType>
          </element>
        </sequence>
      </complexType>
    </element>
  </sequence>
</complexType>
</schema>
```

9

Administering the Enterprise Service Bus

This chapter discusses various administration tasks for Oracle Enterprise Service Bus.

This chapter contains the following topics:

- Administrative Stages on page 9-1
- Understanding Oracle Enterprise Service Bus Clusters on page 9-2
- Providing Security on page 9-4
- Setting Up Notification Channels on page 9-4
- Testing the ESB Services on page 9-9
- Checking Log Files on page 9-11
- Moving the ESB Instance to a Different Oracle Home on page 9-13
- Configuring the InterConnect Adapter with ESB on page 9-16

Administrative Stages

At different stages in setting up and running Oracle Enterprise Service Bus, administrative attention is required, as described in the following list:

- Planning Resources for the ESB

  This stage involves choosing the right amount of resources to provide the necessary capacity. This includes simple resources such as CPU, memory, and disk space; and also more complex resources including the type and size of the database to use for the metadata repository.

- Installing the ESB software

  This stages involves determining the system topology, deciding whether or not to create a cluster of Oracle Enterprise Service Bus Servers installed on different machines for high availability, choosing an active/active or active/passive model for failover, and choosing firewalls.

- Designing the enterprise service bus

  This stage involves testing and stress testing the enterprise service bus at design time

- Functional testing

  This stage involves testing to make sure the system supports the message flow as expected, coverage to ensure that the testers do not overlook any key elements, the number of different environments, and production once all testing is complete

- Operational tuning

  This stage involves tuning the Oracle Enterprise Service Bus to provide the best performance, generating and reading log files, and performing backups and restoring from a backup if necessary.

For information about administration tasks, see *Oracle Application Server Administrator's Guide*.

Planning Resources for the ESB

To estimate capacity, consider the needs of the business that will use the ESB system, including the following:

- The number of message instances
- The path through the ESB for each message
- Throughput of ESB messages, including:
  - Peak throughput per hour
  - Sustained throughput per hour
- Load of ESB messages, including:
  - Peak load start time and duration
  - Sustained load duration
- Resource sizing
  - The number and sizes of documents, or payloads, passed between the ESB processes and the web services they use
  - The average response time for synchronous process during normal load
  - Average response time for synchronous process during peak load
- Fault-tolerance – is High Availability required, or is cold failover acceptable?

In addition to business requirements, the needs of the ESB processes that will be used need to be considered and estimated. These needs include:

- Payload size

  Select the system resource capacity to accommodate the anticipated payloads plus reserve capacity. This is particularly an issue for Java-based application server components. For example, a 50 MB XML document in memory is large and therefore a challenge to process unless system resources are unusually generous. For efficiency, the best practice is to break up large payloads into smaller pieces.
- Data transformations performed
- End points and integration points
- Number of Web services used and how often the ESB process invokes them
- Type of Web service invocations, synchronous or asynchronous
- Anticipated response times for both synchronous callbacks

Understanding Oracle Enterprise Service Bus Clusters

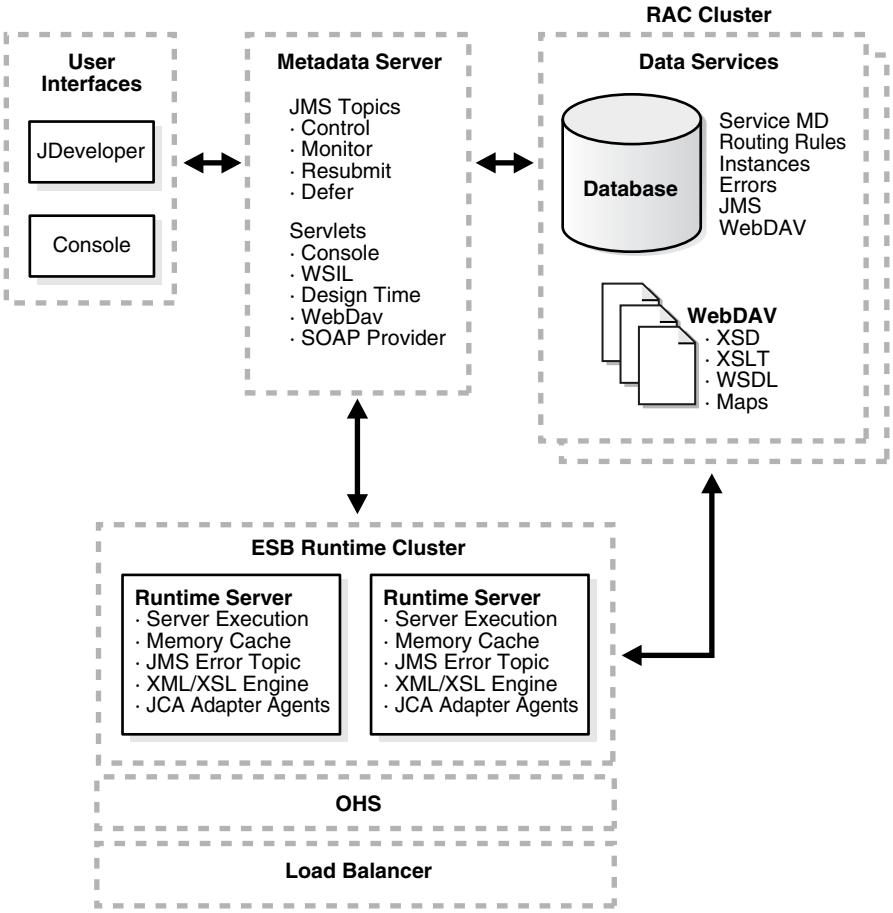
To configure Oracle Enterprise Service Bus for high availability, you configure it as a cluster. An ESB cluster is a collection of instances with identical configuration and

deployment. Clusters enforce homogeneity between member instances so that a group of ESB systems can appear and function as a single instance. With appropriate front-end load balancing, any instance in an ESB cluster can serve requests. This simplifies configuration and deployment across multiple instances and enables fault tolerance among clustered instances.

When you set up Oracle Enterprise Service Bus as a cluster, you first need to install the J2EE Server and Web Server option of the Application Server Advanced Installation on all the computers that belong to the cluster. Next, you need to install the Design Time (repository) OracleAS Middle Tier option of the ESB installation on one computer of the ESB cluster. Then install the Runtime OracleAS Middle Tier option of the ESB installation on the remaining computers of the ESB cluster. You can have multiple runtime installations, but only one design time (repository) installation. After the installations are completed, you need to configure the systems in the ESB cluster.

For an illustration of an ESB architecture showing a clustered environment, see Figure 9–1.

Figure 9–1 ESB Architecture - Clustered Environment



By configuring ESB systems into ESB clusters, you can take advantage of the high availability and load balancing features that OC4J instances configured with multiple processes provide. Configuring ESB systems in ESB clusters do not limit the flow of messages; messages can flow across ESB clusters.

When you install Oracle Enterprise Service Bus, an ESB cluster named `ESB` is created and configured for you. This default ESB cluster is provided so that you can use ESB

tutorial documented in the *Oracle Enterprise Service Bus Quick Start Guide* without the prerequisite of configuring an ESB cluster manually. For production systems, you will want to create and configure additional ESB clusters, if you want to take advantage of the load-balancing and high-availability features enabled by OC4J clusters.

For information about high availability solutions for Oracle Enterprise Service Bus, see "High Availability for Oracle SOA Suite" in *Oracle Application Server High Availability Guide*. For information about high availability deployment scenarios for Oracle Enterprise Service Bus, see *Oracle Application Server Enterprise Deployment Guide*.

Providing Security

Security for Oracle Enterprise Service Bus is provided by Oracle Application Server. See the following resources information about setting up security:

- For information about Secure Sockets Layer (SSL) and HTTPS protocol in Oracle Application Server, see "Enabling SSL in the Infrastructure" and "Enabling SSL in the Middle Tier" in *Oracle Application Server Administrator's Guide*.
- For information about JDBC Client-Side security features, see *Oracle Containers for J2EE Security Guide*.
- For information about Oracle Web Services Manager's authentication capabilities, see *Oracle Web Services Manager Administrator's Guide*.

Setting Up Notification Channels

You can set up notification channels in the event that an error occurs in the Oracle Enterprise Service Bus message processing. These notifications can be sent to the email ID, pager number, mobile number, or phone number of a system administrator or the person assigned to this task. You can set notification channels for each ESB system you have created.

Before notifications are sent, the channels must be configured using the XML files in the `Oracle_Home\integration\esb\config\` directory.

> **Note:** If you have installed the SOA suite, or ESB and BPEL are both on the same middle tier, the configuration files are located in the `Oracle_Home\bpel\system\services\config` directory.

This sections contains the following topics:

- Specifying Notification Channels on page 9-4
- Configuring the Email Notification on page 9-5
- Configuring the Wireless Provider for Voice on page 9-7
- Configuring Paging Notification on page 9-9
- Configuring Mobile Notification on page 9-9
- Configuring Phone Notification on page 9-9

Specifying Notification Channels

To specify the notification channels:

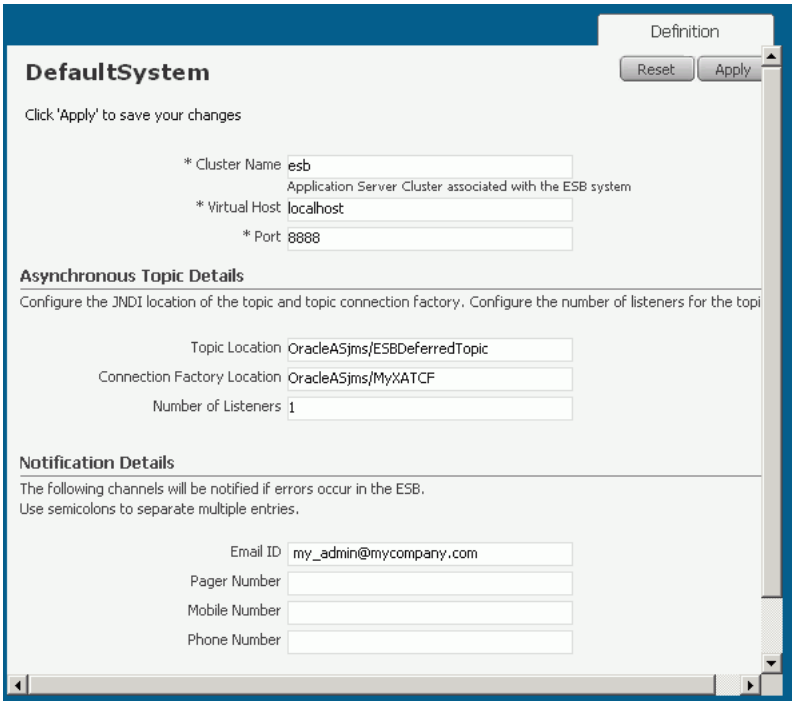
1. Start the ESB Server and open the Oracle ESB Control.

See "Opening the Oracle ESB Control" on page 1-15 for instructions.

2. In the **Services** navigation tree, select the ESB system for which you want to set channels under Notification Details.

   The Definition tab appears similar to Figure 9–2.

Figure 9–2 ESB System – Definition Tab



3. Enter or update the values in the **Email ID**, **Pager Number**, **Mobile Number**, or **Phone Number** fields. Separate multiple entries with commas.

   For example, enter `sys_admin@mycompany.com,it_mgr@mycompany.com` in the **Email ID** field.

4. If you are satisfied with your changes, click **Apply;** otherwise, click **Reset** to return the properties to the settings that were presented when you opened the page.

Configuring the Email Notification

To send email for notifications, ensure that the mail server for the Oracle Enterprise Service Bus has been set up properly for each account in the *`Oracle_Home`*`\integration\esb\config\ns-emails.xml` file.

Each `EmailAccount` element sets the configuration of a specific e-mail account. The `name` attribute in the `EmailAccount` element is the name of the account.

A default e-mail account is specified in the e-mail configuration file. This account is used when there is no account specified to send an e-mail notification. This account is also used to send task-related notifications. A default e-mail account must always be specified in the configuration file.

The `EmailAccount` element contains the `OutgoingServerSettings` and `IncomingServerSettings` attributes. For actionable notifications in a workflow, both `IncomingServerSettings` and `OutgoingServerSettings` are required.

Table 9–1 describes the XML elements for the e-mail notification configuration stored in the `ns_emails.xml` file.

Table 9–1 XML Elements for the E-mail Notification Configuration File

Name	Description
`EmailAccount/Name`	Name of the account. This can be any name, but must be unique within this server.
`EmailAccount/GeneralSettings/FromName`	Name of the `From` e-mail address
`EmailAccount/GeneralSettings/FromAddress`	E-mail address for the `From` e-mail address
`EmailAccount/OutgoingServerSettings/SMTPHost`	Name of the outgoing SMTP server
`EmailAccount/OutgoingServerSettings/SMTPPort`	Port of the outgoing SMTP server
`EmailAccount/IncomingServerSettings/Server`	Name of the incoming e-mail server
`EmailAccount/IncomingServerSettings/Port`	Port of the incoming e-mail server
`EmailAccount/IncomingServerSettings/UserName`	User ID of the e-mail address
`EmailAccount/IncomingServerSettings/Password`	User password
`EmailAccount/IncomingServerSettings/Password[encrypted`	Encrypted attribute of the password. It is `true` if the password is encrypted and `false` if it is not. Generally, you should set this to `false` when you first enter the password. The server automatically encrypts the password the first time it reads the configuration file and sets the attribute to `true`.
`EmailAccount/IncomingServerSettings/UseSSL`	Secure sockets layer (SSL) attribute. It is `true` if the incoming server requires SSL and `false` if it does not.
`EmailAccount/IncomingServerSettings/Folder`	Name of the folder from which to read the incoming messages
`EmailAccount/IncomingServerSettings/PollingFrequency`	Polling interval for reading messages from the incoming messages folder

The following is an example of an `ns-emails.xml` file:

Example 9–1 Sample ns-emails.xml File

```
<!-- ========================================================================================== -->
<!--  1) The default values in this file have to be changed for email to work.                 -->
<!--  2) In addition to setting the email account(s), you also need to change the attribute     -->
<!--    "NotificationMode" to either (the default value of this attribute is "NONE")            -->
<!--     * "EMAIL" - if you have only email set up, but not other channels                     -->
<!--     * "ALL" - if you have email and other channels like voice, SMS, fax, etc. set up.      -->
<!-- ========================================================================================== -->

<EmailAccounts xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"
               EmailMimeCharset=""
               NotificationMode="NONE">
   <EmailAccount>
      <Name>Default</Name>
      <GeneralSettings>
         <FromName>Oracle ESB</FromName>
         <FromAddress>accountId@yourdomain.com</FromAddress>
      </GeneralSettings>
      <OutgoingServerSettings>
         <SMTPHost>yourdomain.com</SMTPHost>
         <SMTPPort>25</SMTPPort>
```

```
      </OutgoingServerSettings>
      <IncomingServerSettings>
         <Server>yourdomain.com</Server>
         <Port>110</Port>
         <Protocol>pop3</Protocol>
         <UserName>accountId</UserName>
         <Password ns0:encrypted="false"
xmlns:ns0="http://xmlns.oracle.com/ias/pcbpel/NotificationService">password</Password>
         <UseSSL>false</UseSSL>
         <Folder>Inbox</Folder>
         <PollingFrequency>1</PollingFrequency>
         <PostReadOperation>
            <MarkAsRead/>
         </PostReadOperation>
      </IncomingServerSettings>
   </EmailAccount>
   <!--EmailAccount>
      <Name>TaskServiceReceiving</Name>
      <GeneralSettings>
         <FromName>Oracle BPM</FromName>
         <FromAddress>accountId@yourdomain.com</FromAddress>
      </GeneralSettings>
      <OutgoingServerSettings>
         <SMTPHost>yourdomain.com</SMTPHost>
         <SMTPPort>25</SMTPPort>
      </OutgoingServerSettings>
      <IncomingServerSettings>
         <Server>yourdomain.com</Server>
         <Port>110</Port>
         <Protocol>pop3</Protocol>
         <UserName>accountId</UserName>
         <Password ns0:encrypted="false"
xmlns:ns0="http://xmlns.oracle.com/ias/pcbpel/NotificationService">password</Password>
         <UseSSL>false</UseSSL>
         <Folder>Inbox</Folder>
         <PollingFrequency>1</PollingFrequency>
         <PostReadOperation>
            <MarkAsRead/>
         </PostReadOperation>
      </IncomingServerSettings>
   </EmailAccount-->
</EmailAccounts>
```

Configuring the Wireless Provider for Voice

The wireless provider for voice is set up in the *Oracle_Home*\integration\esb\config\ns_iaswconfig.xml file.

Example 9–2 describes the XML elements for the voice notification configuration stored in ns_iaswconfig.xml on the *Oracle_Home* server.

Table 9–2 XML Elements for the Voice Notification Configuration File

Name	Description
/IASWConfiguration/SoapURL	URL of the wireless service provider
/IASWConfiguration/UserName	Name of the user account with the wireless service provider
/IASWConfiguration/Password	User password

Table 9–2 (Cont.) XML Elements for the Voice Notification Configuration File

Name	Description
`/IASWConfiguration/Password[encrypted`	Encrypted attribute of the password. It is `true` if the password is encrypted and `false` if it is not. Generally, you should set this to `false` when you first enter the password. The server automatically encrypts the password the first time it reads the configuration file and sets the attribute to `true`.
`/IASWConfiguration/ProxyHost`	Name of the proxy server
`/IASWConfiguration/ProxyPort`	Port number of the proxy server

Note: The username and password are intentionally left blank at installation. If a username or password is not specified, the wireless server allows up to 50 notifications from a specific IP address. After 50 notifications, you must get a paid account from:

http://messenger.oracle.com

You then specify the appropriate username and password in the configuration file, `ns_iaswconfig.xml`, or by using Oracle Enterprise Manager Application Server Control.

Example 9–2 is a an example of the `ns_iaswconfig.xml` file.

Example 9–2 Sample ns_iaswconfig.xml File

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!-- ========================================================================================== -->
<!-- 1) This XML file stores the details of the IAS Wireless Notification Service.              -->
<!-- 2) In addition to setting the email account(s) in ns_emails.xml and ns_iaswconfig.xml,     -->
<!--    you also need to change the attribute "NotificationMode" in ns_emails.xml to either     -->
<!--    EMAIL or ALL (the default value of this attribute is "NONE")                            -->
<!--     * "EMAIL" - if you have only email set up, but not other channels                      -->
<!--     * "ALL" - if you have email and other channels like voice, SMS, fax, etc. set up.      -->
<!-- ========================================================================================  -->

<IASWConfiguration xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
  <!-- URL to the SOAP Service -->
  <SoapURL>http://messenger.oracle.com/xms/webservices</SoapURL>

  <!-- UserName - this username should exist in iAS Wireless schema -->
  <UserName></UserName>
  <Password ns0:encrypted="false"
xmlns:ns0="http://xmlns.oracle.com/ias/pcbpel/NotificationService"></Password>

  <!-- Proxy information to connect to the hosted wireless service -->
  <ProxyHost>www-proxy.us.oracle.com</ProxyHost>
  <ProxyPort>80</ProxyPort>

  <!-- Specify any custom implementation for sending notification through Voice, Fax,  -->
  <!-- Pager, SMS or IM channels.                                                        -->
  <!--  All refers to an implementation for all of the above specified channels.        -->
  <!--  Email service can't have a custom implementation                                -->
  <CustomNotificationServices>
    <All/>
```

```
    <Email/>
    <Voice/>
    <Fax/>
    <Pager/>
    <SMS/>
    <IM/>
  </CustomNotificationServices>
</IASWConfiguration>
```

Configuring Paging Notification

To send paging notifications, make sure that paging for the Oracle Enterprise Service Bus has been set up properly for each account in the *Oracle_Home*\integration\esb\config\ns_iaswconfig.xml file.

See "Configuring the Wireless Provider for Voice" on page 9-7.

Configuring Mobile Notification

To send mobile notifications, make sure that mobile notification for the Oracle Enterprise Service Bus has been set up properly for each account in the *Oracle_Home*\integration\esb\config\ns_iaswconfig.xml file.

See "Configuring the Wireless Provider for Voice" on page 9-7.

Configuring Phone Notification

To send phone notifications, make sure that the phone notification for the Oracle Enterprise Service Bus has been set up properly for each account in the *Oracle_Home*\integration\esb\config\ns_iaswconfig.xml file.

See "Configuring the Wireless Provider for Voice" on page 9-7.

Configuring the FAX Cover Page

To add cover pages for a fax, you need to edit the *Oracle_Home*\integration\esb\config\ns_faxcoverpages.xml. Use the cover page name to specify the cover page in the fax message.

Example 9–3 Example of FAX Cover Page File

```
<FaxCoverPages xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
  <FaxCoverPage>
     <Name>legal</Name>
     <MimeType>application/pdf</MimeType>
<FileLocation>C:\esb\runtime\config\faxcoverpages\003288.pdf</FileLocation>
  </FaxCoverPage>
</FaxCoverPages>
```

Testing the ESB Services

To test the services you are creating in Oracle JDeveloper, use the Test Web Service feature of Oracle Enterprise Manager.

To test using Oracle Enterprise Manager, perform the following steps:

1. Enter the URL for Oracle Enterprise Manager:

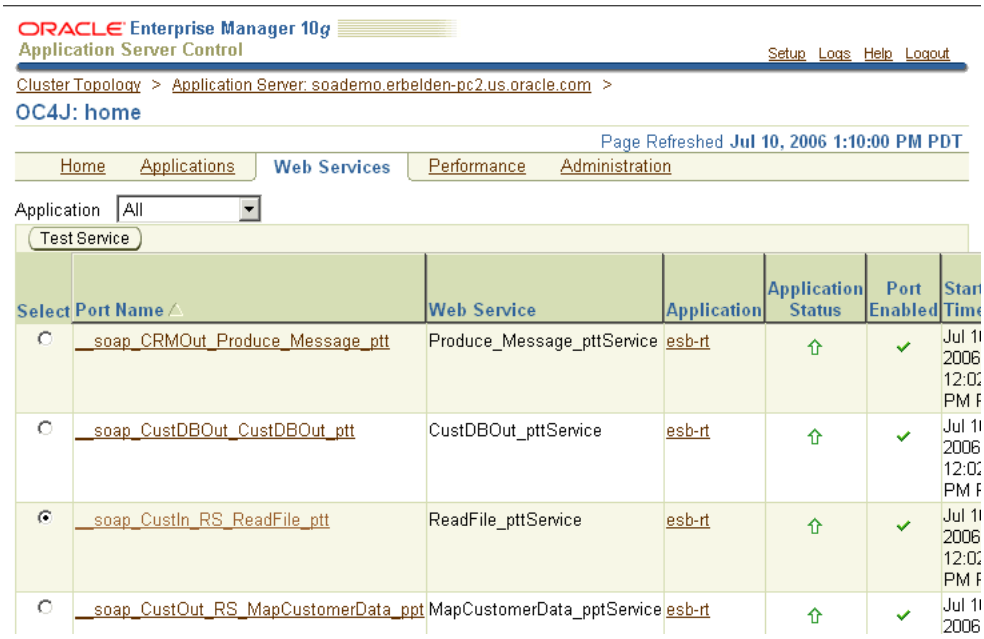
   For example: `http://localhost:8888/em`

2. Enter the username and password to log in.
3. Click an application server in the list of All Application Servers.

   For example, click the **home** application server.
4. In Application Server page, click the **Web Services** link.

   The list of Web services displays.

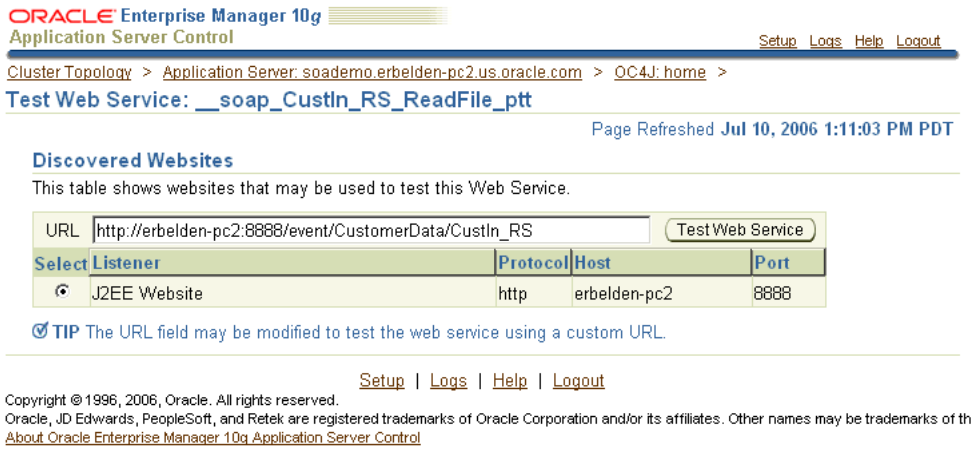
Figure 9–3 Oracle Enterprise Manager Web Services



5. Select a Web service and click the Test Service button.

   The Test Web Service page displays.

Figure 9–4 Oracle Enterprise Manager Test Web Service



6. In the Test Web Service page, click the Test Web Service button.

   The test page displays.

7. Enter data in the fields of the Test page to test the service.

   For example, enter sample data for a customer order that would be processed by the `CustIn_RS` routing service.

Figure 9–5 Oracle Enterprise Manager Test Page

8. After providing sample data, click **Invoke** to display the Test Results page.

   Check whether there are any errors in the page. You can close the test page after viewing the results.

9. View Instances in the Oracle ESB Control to check whether the message instance was successfully processed with the sample data. For an example of successful processing in the Instances in the Oracle ESB Control, see Figure 3–2 on page 3-5.

10. If necessary, correct any errors and repeat the test process.

Checking Log Files

You can view logs for all the OC4J components running on an application server with Oracle Enterprise Manager Application Server Control. Oracle Application Server components generate log files containing messages that record all types of events, including startup and shutdown information, errors, warning messages, access information about HTTP requests, and additional information. You can set the level of information in a log file with the Logger Configuration page.

This sections contains the following topics:

- Viewing Log Files on page 9-12
- Configuring Log Files on page 9-13

> **See Also:** "Managing Log Files" in *Oracle Application Server Administrator's Guide* for more information about the log files of Oracle Application Server components

Viewing Log Files

You can view log files using Oracle Enterprise Manager, as follows:

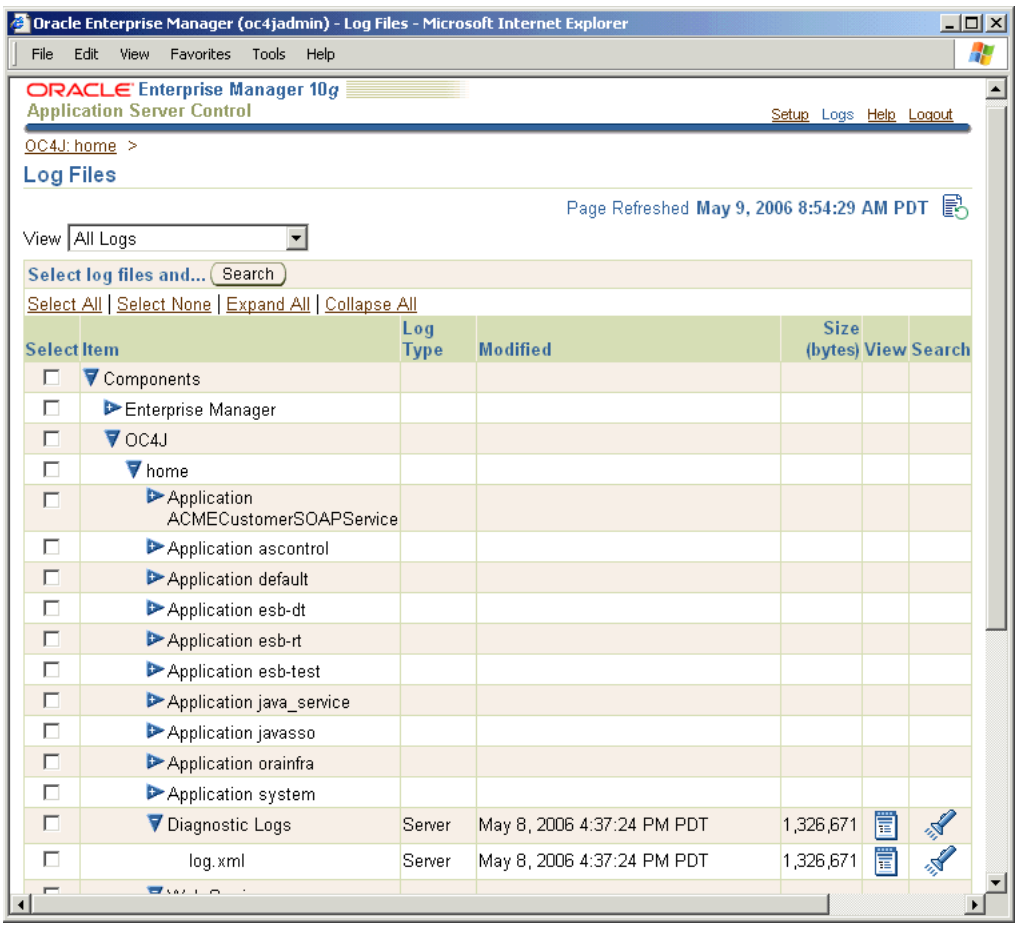
1. Go to the following URL:

   `http://host_name:port_number/em`

2. Log in with username/password.
3. Click an application server to display the home page of the server.
4. Click **Logs** in the upper right corner to display the Log Files page.
5. Expand **Components** > **OC4J** > *name-of-application server*

   The page appears similar to Figure 9–6.

Figure 9–6 Oracle Application Server Control – OC4J Log Files



6. Expand the items in the Select list until you locate a specific log file, then click **View**.

   For example, expand the Application esb-dt, Application esb-rt, and Diagnostic Logs items. Click the icon in the View column to display the log file text.

   The diagnostic file is located in the `log.xml` file and can be viewed with a text editor.

Configuring Log Files

You can configure log file settings for individual logger classes to display more information in the log files.

To configure the log file settings:

1. Go to the following URL:

   `http://host_name:port_number/em`

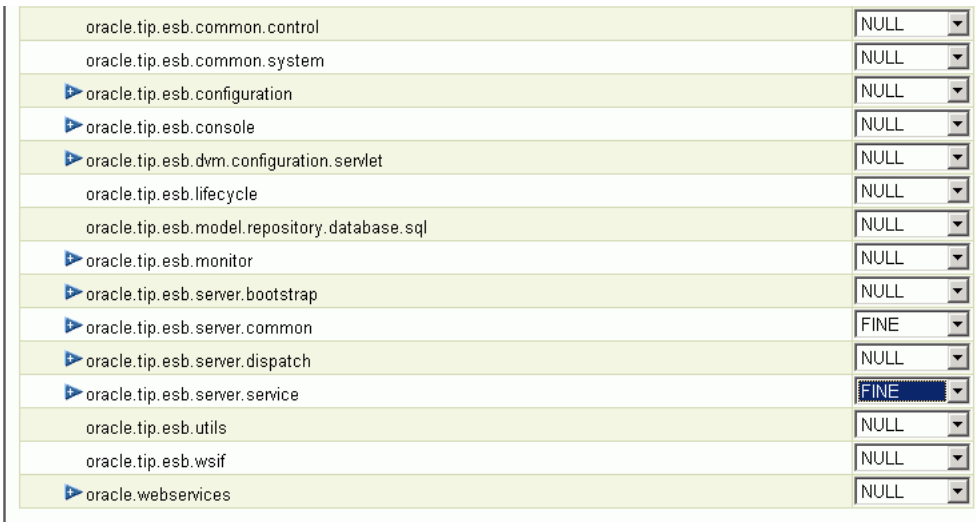
2. Log in with username/password.
3. Click an application server to display the home page of the server.
4. Click the **Administration** tab.
5. Under Administration Tasks > Properties, click **Go to Task** next to Logger Configuration.
6. Expand Root Logger > oracle to display the logger classes and the Log Level settings.
7. You can locate specific logger classes by name with the search option. In the **Search by Logger Name** field, enter a string and then click **Go**.

   For example, enter **esb** in the **Search by Logger Name** field.

8. In the Log Level list, select a level option for the logger classes you want to configure.

   For example, set `oracle.tip.esb.server.common` and `oracle.tip.esb.server.service` to `FINE`, as shown in Figure 9–6.

Figure 9–7 Oracle Application Server Control – Logger Configuration



9. Click **Apply**.

Moving the ESB Instance to a Different Oracle Home

You can move the ESB instance to a different database if you want to change the Oracle Home of the ESB Server.

To move the ESB instance, follow these steps:

1. If the new Oracle Home does not have an ESB schema, create a new database schema by executing the `createuser.sql` and `createschema.sql` scripts that are located in the *`Oracle_Home`*`/integration/esb/sql/oracle/` directory.

   For example:

   ```
   SQL>@Oracle_Home\integration\esb\sql\oracle\createuser.sql
   SQL>@Oracle_Home\integration\esb\sql\oracle\createschema.sql
   ```

   If the new Oracle Home contains already contains an ESB schema:

   - You can back up the ESB metadata with the ESB `export` utility. See "Using the ESB Import and Export Utilities" on page 9-15.
   - You can remove the existing schema with the `drop_esb_tables.sql` script that is located in the *`Oracle_Home`*`/integration/esb/sql/oracle/` directory.

2. Configure the ESB JDBC connection pooling to point to the new Oracle Home in the *`Oracle_Home`*`/j2ee/home/config/data-sources.xml` file. You need to update the `connection-factory` details in the `data-sources.xml` file:

   ```
   <connection-pool name="ESBPool">
     <connection-factory factory-class="driver_class_name_for_db"
        url="url_to_database" user="db_user_name" password="db_user_password"/>
   </connection-pool>
   ```

3. Export the ESB parameters from the old Oracle Home by running the `ant` utility as follows:

   ```
   ant export-params -Dparamfile=file_path_specfication\esb.properties>
                     -DDB_URL=jdbc_url_to_old_database
                     -DDB_USER=db_user_name
                     -DDB_PASSWORD=db_user_password
   ```

   You do not need to provide the `DDB_URL`, `DDB_USER`, and `DDB_PASSWORD` parameters if you are using a standalone Oracle Lite database.

4. Export the metadata from the old Oracle Home with the ESB `export` utility.

   For example:

   ```
   > Oracle_home/integration/esb/bin/export c:\temp\metadata.zip
   ```

   See "Using the ESB Import and Export Utilities" on page 9-15.

5. Import the ESB parameters from the old Oracle Home to the new Oracle Home by running the `ant` utility as follows:

   ```
   ant import-params -Dparamfile=file_path_specfication\esb.properties>
                     -DDB_URL=jdbc_url_to_new_database
                     -DDB_USER=db_user_name
                     -DDB_PASSWORD=db_user_password
   ```

   You do not need to provide the `DDB_URL`, `DDB_USER`, and `DDB_PASSWORD` parameters if you are using a standalone Oracle Lite database.

6. Restart the ESB Server with the `opnmctl` utility as follows:

   ```
   > Oracle_Home\opmn\bin\opmnctl stopall
   > Oracle_Home\opmn\bin\opmnctl startall
   ```

7. Import the metadata from the old Oracle Home to the new Oracle Home with the ESB `import` utility.

For example:

```
> Oracle_home/integration/esb/bin/import c:\temp\metadata.zip
```

See "Using the ESB Import and Export Utilities" on page 9-15.

Using the ESB Import and Export Utilities

You can use the Oracle Enterprise Service Bus `import` and `export` utilities to export ESB metadata to and from an ESB Server on different Oracle instances.

To export metadata from anESB Server:

1. Start the ESB Server if it is not currently running. See "Starting, Stopping, and Accessing Oracle Enterprise Service Bus Components" on page 1-13.
2. Open a operating system command window.
3. Change the directory to the *Oracle_home*`/integration/esb/bin` directory.

   For example, `cd c:\product\10.1.3.1\integration\esb\bin`
4. Enter the following command:

```
export file-spec
   -system system_name
   -serviceGroup service_group_name
   -service  service_name
   -flow
```

   - *file-spec* is the full file specification for the file in which you want to export the metadata
   - `-system` is the name of the system to be exported. All service groups/services belonging to the specified system are exported recursively.
   - `-serviceGroup` is the name of the service group to be exported. All service groups/services belonging to the specified service group are exported recursively.
   - `-service` is the name of the service to be exported
   - `-flow` indicates that the entire flow of the service specified using `-service` option is to be exported. All the down stream services along with this service are exported. If inbound adapter service is specified using `-service` option, then all services involving in the flow are exported. If a routing service is specified, the routing service and all the down stream services involving in the flow are exported.

   The options `-system`, `-serviceGroup`, and `-service` can be used multiple times for different entities. If no option is specified, the entire metadata is exported.

   Examples:

```
export c:\temp\metadata.zip
```

```
export c:\esb\metadata.zip -system T-Vox -system T-Vox1
-serviceGroup T-Vox2.Stores -service
T-Vox3.Stores.SFOProductLanuch -flow
```

To import metadata to an ESB Server:

1. Start the ESB Server if it is not currently running. See "Starting, Stopping, and Accessing Oracle Enterprise Service Bus Components" on page 1-13.

2. Open a operating system command window.
3. Change the directory to the *Oracle_home*/integration/esb/bin directory.

   For example, `cd c:\product\10.1.3.1\integration\esb\bin`
4. Enter the following command:

```
import file-spec
   -system system_name
   -serviceGroup service_group_name
   -service  service_name
   -flow
```

   - *file-spec* is the full file specification for the file that contains the metadata that you want to import
   - `-system` is the name of the system to be imported. All service groups/services belonging to the specified system are imported recursively.
   - `-serviceGroup` is the name of the service group to be imported. All service groups/services belonging to the specified service group are exported recursively.
   - `-service` is the name of the service to be imported
   - `-flow` indicates that the entire flow of the service specified using `-service` option is to be imported. All the down stream services along with this service are imported. If inbound adapter service is specified using `-service` option, then all services involving in the flow are imported. If a routing service is specified, the routing service and all the down stream services involving in the flow are imported.

   The options `-system`, `-serviceGroup`, and `-service` can be used multiple times for different entities. If no option is specified, the entire metadata is imported.

   Examples:

```
import c:\temp\metadata.zip
```

```
import c:\esb\metadata.zip -system T-Vox -system T-Vox1
-serviceGroup T-Vox2.Stores -service
T-Vox3.Stores.SFOProductLanuch -flow
```

Configuring the InterConnect Adapter with ESB

To use Oracle Enterprise Service Bus with Oracle InterConnect, you need to install and configure the ICAdapter for use with ESB.

To install and configure the InterConnect Adapter, perform the following steps. For a SOA basic (instead of advanced) installation, replace `oc4j_soa` with `home` in the java commands.

1. Deploy ICAdapter with the following:

```
java -jar admin_client.jar
  deployer:oc4j:opmn://hostname:opmn_request_port/oc4j_soa oc4jadmin welcome1
    -deploy -file oracle_home\integration\esb\lib\icadapter.rar
    -deploymentName IcAdapter
```

2. Deploy ICwsilplugin with the following:

```
java -jar admin_client.jar
```

```
  deployer:oc4j:opmn://hostname:opmn_request_port/oc4j_soa oc4jadmin welcome1
    -deploy -file oracle_home\integration\esb\lib\icwsilplugin.ear
    -deploymentName icwsilplugin -parent default
```

3. Bind ICWsilPlugin with the following:

```
java -jar admin_client.jar
  deployer:oc4j:opmn://hostname:opmn_request_port/oc4j_soa oc4jadmin welcome1
    -bindWebApp -appName icwsilplugin -webModuleName icwsilplugin
    -webSiteName default-web-site -contextRoot /ic
```

4. Update the `oc4j-ra.xml` file in the *`Oracle_home`*`\j2ee\home\application-deployments\defaults\ICAdapter` directory. The following is an example of the `oc4j-ra.xml` file.

```
<?xml version="1.0"?>
<oc4j-connector-factories ...... schema-minor-version="0" >
 <connector-factory location="eis/ICAdapter"
                    connector-name="Interconnect Adapter">
   <config-property name="applicationName" value="BPELPM"/>
   <config-property name="driverClassName"
                    value="oracle.jdbc.driver.OracleDriver"/>
   <config-property name="connectionString"
                    value="jdbc:oracle:thin:@bpel-db-as:1521:ORCL"/>
   <config-property name="userName" value="ichub"/>
   <config-property name="password" value="Manager1"/>
   <config-property name="repoName" value="InterConnectRepository"/>
   <connection-pooling use="none"></connection-pooling>
   <security-config use="none"></security-config>
 </connector-factory>
</oc4j-connector-factories>
```

5. Add the `oai.jar` path to the `server.xml` file in the *`Oracle_home`*`\j2ee\home\config` directory. The following is an example of the `server.xml` file.

```
<shared-library name="bpel" version="1.0" library-compatible="true">
  <code-source path="c:/oracle/mid/integration/interconnect/lib/oai.jar"/>
  <code-source path="../../../integration/esb/lib/orabpel-common.jar"/>
  <code-source path="../../../integration/esb/lib/orabpel-thirdparty.jar"/>
  <code-source path="../../../integration/esb/lib/orabpel.jar"/>
  <code-source path="../../../integration/esb/lib/orabpel-boot.jar"/>
  <code-source path="../../../integration/esb/lib/bpm-infra.jar"/>
  <code-source path="../../../integration/esb/lib/olite40.jar"/>
  <code-source path="../../../integration/esb/lib/wdk.jar"/>
  <import-shared-library name="apache-commons"/>
  <import-shared-library name="oracle.xml"/>
  <import-shared-library name="oracle.jdbc"/>
  <import-shared-library name="oracle.jwsdl"/>
  <import-shared-library name="oracle.dms"/>
  <import-shared-library name="soap"/>
  <import-shared-library name="oracle.ws.client"/>
</shared-library>
```

6. Restart the ESB Server.

7. Register the InterConnect events and procedures under the ESB application in InterConnect to ESB as ESB services with the following:

```
oracle_home\integration\esb\bin\regadapters.bat
```

On a Linux system, run the `regadapters.sh` script.

10

Tracking Message Instances Across the Enterprise Service Bus

This chapter describes how to use the Oracle ESB Control Instances view to track message instances across the enterprise service bus.

This chapter contains the following topics:

- Overview of the Oracle ESB Control Instances View on page 10-1
- Understanding Instances View Elements and Controls on page 10-2
- Administering Message Instances on page 10-6

Overview of the Oracle ESB Control Instances View

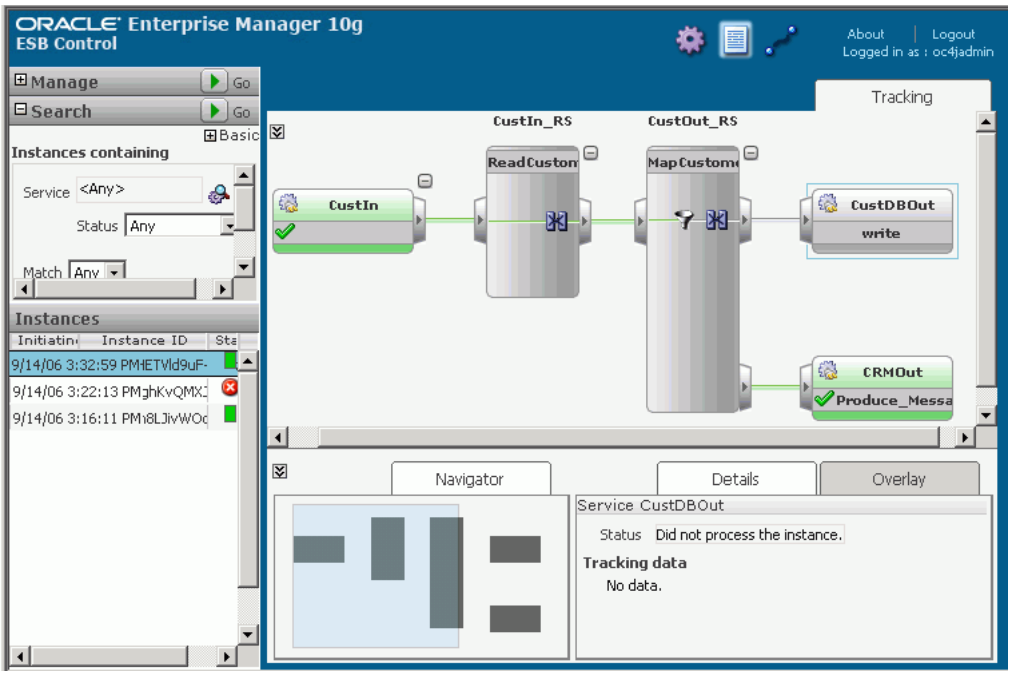
The Instances view enables you to view details about instance processing across an ESB system. It enables you to filter message instances based on any of the following properties:

- The service that processed them
- The status of the messages (Any, Error, Faulted, Resubmittable)
- Tracking name and tracking value (trackable fields)
- Instance IDs
- The time frame during which the message was processed

When you select a message instance from the Instances panel (shown in the lower left of Figure 10–1), the message instance's path through the enterprise service bus is presented in the Tracking tab diagram. Within the diagram, services that successfully processed the message instance are represented in green, services where an error occurred are represented in red, and services that were not invoked (or have not yet been invoked) in the processing of the message are represented in gray. If the transaction is rolled back, the service is represented in yellow. You can select a service in the Tracking tab diagram and view the processing details in the Details subtab, shown in the lower right of Figure 10–1.

Figure 10–1 shows that the CRM adapter service (CRMOut) was successfully invoked and that the database adapter service (CustDBOut) was not invoked because a filter expression excluded the message from delivery to the database adapter service.

Figure 10–1 Instances View – Tracking Tab and Details Subtab



Understanding Instances View Elements and Controls

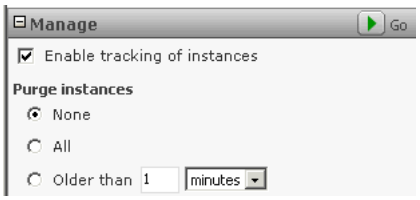
The Instances view of the Oracle ESB Control, as shown in Figure 10–1 is divided into four main regions, as follows:

- Manage panel

  This panel, shown in Figure 10–2, enables you to enable and disable message instance tracking as well as purge the message instances. Note that when instances that are purged, they are lost forever.

  The control to the left of the Manage title enables you to open or close this panel; the green arrow icon to the right of the Manage title applies changes you make in this panel.

Figure 10–2 Instances View – Manage Panel



- Search panel

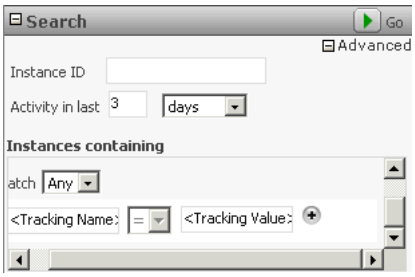
  Because the volume of messages processed during that time can be quite large, the Search panel, shown in Figure 10–3, enables you to limit the messages displayed using a variety of filter criterion.

  The control to the left of the Search title enables you to open or close this panel; the green arrow icon to the right of the Search title applies changes you make in this panel. The Advanced control toggles to Basic and back; it enables you to show or hide the Instances containing region of this panel.

By default, the Instances panel displays all the messages processed during the past day. If you want to view messages processed prior to the past day, then you need to increase the number of days in the **Activity in last** field in the Advanced options before searching for messages.

Only the first 100 instances are returned after any search filtering.

Figure 10–3 Instances View – Search Panel



- Instances panel

  The Instances panel lists the message instances processed or being processed by the ESB services, optionally filtered by search options you select. The maximum number of displayed is 100.

  For each message, the time at which each message instance entered the enterprise service bus, instance ID for the message, and the status of that processing is provided. The status is represented by the following color-coded icon(s) in the Status column (shown in Figure 10–4):

  – Green rectangle - No faults or errors during message processing

  – Solid red circle overlaid with an X - Error occurred

  – Yellow triangle overlaid with an exclamation point - Faulted message (the target service returned an application fault)

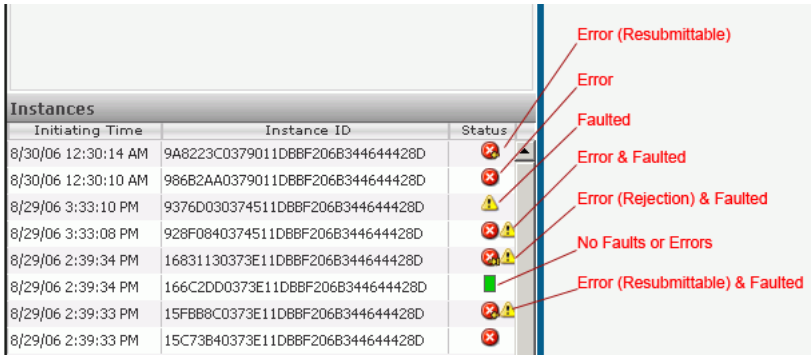
  – An error icon plus a faulted icon - Error and faulted

  – An error icon overlaid with a small yellow left arrow – Error is resubmittable (for asynchronous processing)

  – An error icon overlaid with a small yellow left arrow plus a faulted icon – Error (resubmittable) and faulted

  – An error icon overlaid with a small yellow right arrow plus a faulted icon – Error (rejection by inbound adapter) and faulted

Figure 10–4 Instances Panel Showing Status Codes



- Tracking Tab (with associated subtabs)

  The Tracking tab has a diagram region that provides a schematic of the message processing, as well as Navigator, Details, and Overlay subtabs, as described in the following list:

  – Schematic

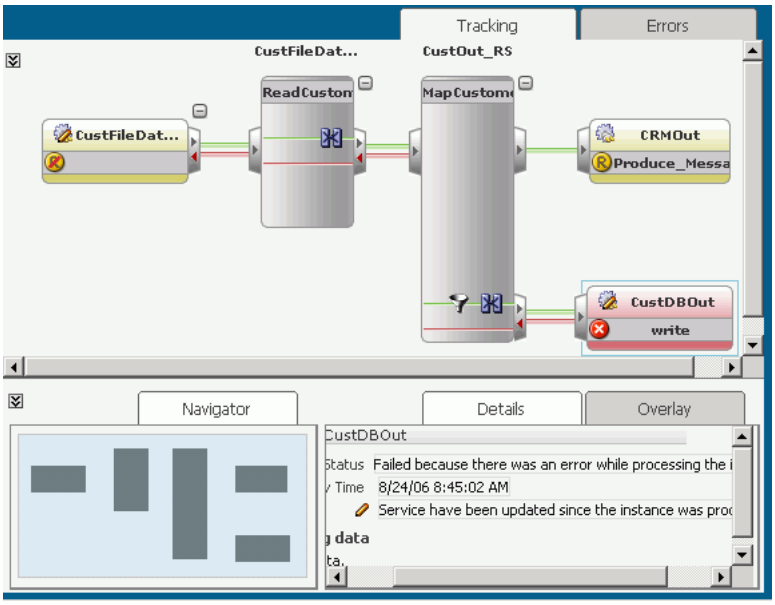
    The Tracking tab schematic shows the path the message selected from the Instances panel has taken (or is taking) across the enterprise service bus. If you select a particular service within the Instances panel, that service is enclosed in a dotted grey box, as shown for the service labeled CustOut_RS in Figure 10–1.

    If a service has successfully processed the message, that service appears in green, as shown for the service labeled CRMOut in Figure 10–1.

    If any service has not processed the selected message because processing is incomplete or due to filter criterion, that service appears in grey, as shown for the service labeled CustDBOut in Figure 10–2.

    If an error occurred during processing, the service at which the error occurred is red, as shown in Figure 10–5. A yellow service icon indicates a non-transactional endpoint.

Figure 10–5 Oracle ESB Control – Instances View with Error



  – Navigator subtab

    For complex ESB configurations, the schematic can be quite large and you may not be able to view it in its entirety all at once. The Navigator subtab provides a birds-eye view of the entire schematic. The purple region of the Navigator tab highlights the area of the schematic currently in showing.

    Use the scroll bars around the schematic to view areas currently outside the purple shaded region. Click the down-arrow button to the left of the Navigator tab to hide the contents of the subtabs (and thereby increase the available viewing area for the schematic.)

  – Details subtab

The details subtab provides information about the status of a message with respect to a selected service. Note that a pencil icon on the service designates that the service has been updated since the message instance was processed.

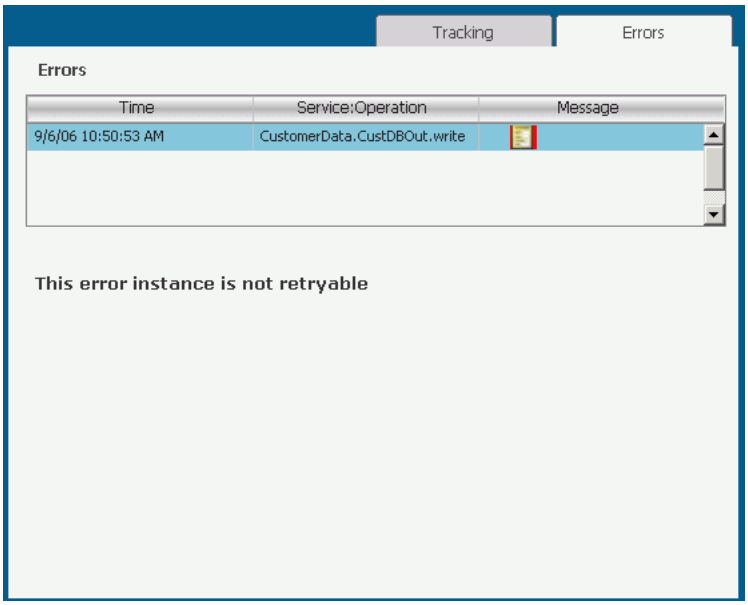
– Overlay subtab

The Overlay subtab enables you to view statistics on the messages processed by each service.

- Error tab

The Error tab is presented only when you select a service in the Instances panel where an error occurred, as shown in Figure 10–6. The Errors tab contains an error table that lists the time, service:operation, and message associated with each error. Click the Error Details icon in the Message column to view the error message and the stack trace. See Chapter 12, "Error Handling" for information about handling error conditions.

Figure 10–6 Instances View – Errors Tab



Administering Message Instances

You can view details and statistics of, search for, and delete message instances. You can also enable and disable message instance tracking.

This section discusses the following topics:

- Enabling and Disabling Instance Tracking on page 10-6
- Viewing Instance Details on page 10-6
- Viewing Instance Statistics on page 10-7
- Searching for Message Instances on page 10-7
- Purging Message Instances on page 10-9

Enabling and Disabling Instance Tracking

Instance tracking is enabled by default. However, for maximum throughput at high volumes, it is recommended that instance tracking is disabled.

Follow these steps to enable or disable message tracking:

1. At the top of the Oracle ESB Control, click the **Instances** button if the Instances view is not currently displaying.
2. Open the Manage panel if it is not already open.
3. Select or deselect **Enable tracking of instances** to enable or disable instance tracking, respectively.
4. In the Manage panel title bar, click the Apply icon (green arrow).

Viewing Instance Details

The Details tab, below the schematic in the Oracle ESB Control Instances view, shows the following details about a message instance, as follows:

- If you select a service in the Instances panel or the schematic:
  - Status

    This field specifies the status of the message, which can be successful or failed. If a message was processed successfully, but the transaction was rolled back due to some other error, that information is included in the Status field. For example, the status might be: `Successful; however, the transaction was rolled back due to an error in CRMOUT:Write.`
  - Activity Time

    This field specifies the time at which the message was processed by the service selected in the Instances panel.
  - Tracking data
- If you click the connection between two services in the schematic:
  - Source

    Specifies the service and operation from which the message instance is being sent, in the format *source_service*::*operation*.
  - Target

    Specifies the service and operation to which the message instance is being sent, in the format *target_service*::*operation*.
  - Status

    Specifies the status of the target service. For example, the status might be one of the following:

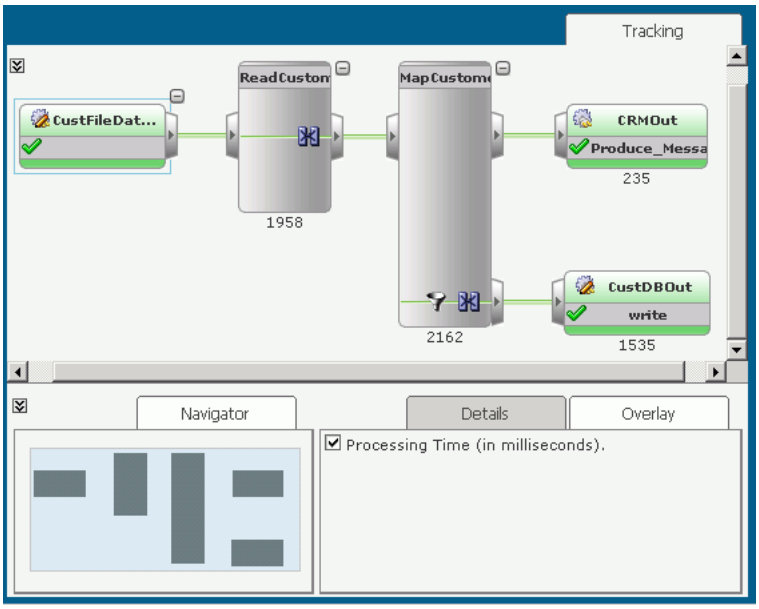
    * Did not process the message.
    * Not executed as part of the message flow.
    * Not successful because there was an error during the invocation of the target.

Viewing Instance Statistics

The Overlay tab, below the schematic in the Oracle ESB Control Instances view, enables you to view statistics as overlays on the schematic.

If you enable Processing Time, the schematic shows the time in milliseconds it took for each service within the enterprise service bus to process the message instance, as shown in Figure 10–7.

Figure 10–7 Sample Overlay Statistics in Instances View



Searching for Message Instances

If the Instances panel contains too many message instances for you to find the message instances of interest quickly, you can search the messages, as follows:

1. At the top of the Oracle ESB Control, click the **Instances** button if the Instances view is not currently displaying.
2. Open the Search panel, if it is not already open.
3. Specify the desired search criteria, as described in Table 10–1.
4. In the Search panel title bar, click the **Apply icon** (green arrow).

Table 10–1 Specifying Message Instance Search Criteria

Search Criteria	Description
Instance ID	To limit the messages to those with a specific instance ID: 1. If they are not already displaying, open the Advanced options by click the expand icon. (The plus (+) box next to **Basic.** in the Instances panel.) 2. In the Instance ID field, enter the ID of the desired message instance. Multiple Instance IDs cannot be specified. If an Instance ID is provided as search criteria, other filter conditions are ignored.

Table 10–1 (Cont.) Specifying Message Instance Search Criteria

Search Criteria	Description
Activity in last <time frame>	To limit the messages to processed in a specific time frame: **1.** If they are not already displaying, open the Advanced options by click the expand icon. (The plus (+) box next to **Basic.** in the Instances panel.) **2.** From the drop-down list next to the **Activity in last** field, select a time period. **3.** In the **Activity in last** field, select the number of time periods specified in step 2 that you want to include. Be aware that message instances for a given time frame may not be available if messages have been purged, as described in "Purging Message Instances" on page 10-9.
Service	To limit the messages to those processed by a particular service: **1.** To the right of the **Service** field, click the **ESB Service Browser icon**. A dialog box opens. **2.** Navigate to the ESB system that contains the service of interest. **3.** Select the service of interest. **4.** Click **Select**.
Status	To limit the messages to those in a particular status, in the **Status** field, select one of the following: ■ Any - Select this option to display all message instances. ■ Error – Select this option to limit the message instances to those for which an error occurred in processing. ■ Faulted – Select this option to limit the message instances to those for which a fault, as defined by the service WSDL, occurred. ■ Resubmittable - Select this option to limit the message instances to those that are resubmittable.
Match	Specify whether to match Any or All of the criteria in the Tracking Name and Value fields.

Table 10–1 (Cont.) Specifying Message Instance Search Criteria

Search Criteria	Description
Tracking Name and Tracking Value (trackable fields)	To limit the messages to those which have a particular value in a trackable field or fields: 1. Replace **<Tracking Name>** with the name of a previously defined trackable field. See "Defining and Managing Tracking Fields" on page 3-16 for instructions on defining trackable fields. 2. The logical operator cannot be selected. It is always equals (=) for an exact match. 3. Replace **<Tracking Value>** with the value for the trackable field. 4. If you want to specify multiple trackable field/value pairs, click the plus (+) button, and then repeat steps 1 and 2 for each additional pair. 5. In the Match field, select Any if you want a message instance listed if it matches one or more of the listed tracking field/tracking value pairs specified; select All if you want only messages listed that match all of the listed tracking field/tracking value pairs. If you want to delete a trackable field/tracking value pair, click the minus (-) button that appears next to that row.

Purging Message Instances

By default, no message instances are purged. As a result the list of message instances in the Instances panel can become unwieldy. You can direct the ESB Server to purge all messages or all message instances that occurred outside a specified time range. Instances that are purged are lost forever.

To purge messages:

1. At the top of the Oracle ESB Control, click the **Instances** button if the Instances view is not currently displaying.
2. Open the Manage panel if it is not already open.
3. Select the frequency with which you want messages purged:
   - To purge all message instances, under **Purge instances,** select **All.**
   - To purge all message instances that occurred outside a time range, under **Purge instances,** select **Older than** and indicate the time range.
4. In the Manage title bar, click the **Apply icon** (green arrow).

See Also: Appendix B, "Oracle Enterprise Service Bus API"

11

Support for Resequencing

Oracle Application Server release 10.1.3.4 introduces support for implementing the Resequencer in ESB. This section contains the following topics:

- Overview of the Resequencer on page 11-1
- Using the Resequencer on page 11-3
- Limitations in the Resequencer on page 11-7
- Configuring Database Adapters and File Adapters for FIFO on page 11-8

Overview of the Resequencer

A Resequencer is used to rearrange a stream of related but out-of-sequence messages back into order. The messages are rearranged on an identifier that is a part of the message. The Resequencer should be aware of the identifier sequence for rearranging the messages. While rearranging the messages, the Resequencer typically uses the identifier sequence to decide the position of a message. This section describes the following:

- Resequencing Order on page 11-1
- Resequencer Thread Pool on page 11-3

Caution: The Resequencer is available in `preview` mode and is disabled by default. When disabled, it does not affect the existing product functionality. If you want to use the Resequencer, then you must enable it by setting the value of the ESB configuration parameter `EnableResequencer` to `true`. For more information, refer to Development on page 11-3.

Resequencing Order

ESB can resequence the incoming messages in a user-specified order. This implementation allows you to specify the three types of orders:

- **First In First Out (FIFO)**

  The FIFO Oracle ESB Resequencer supports the FIFO pattern, which is relevant for applications that need sequencing based on the time of arrival of the message to the ESB Routing Service configured as a Resequencer. The FIFO Resequencer receives a stream of messages that are in order, and uses an external data store to store the messages in order until they are processed asynchronously.

When the messages are processed, they are processed in sequence based on the time of arrival to the routing service. It is important to note that the messages to outbound services of the FIFO Resequencer routing service are guaranteed to arrive in order based on the time of arrival. The FIFO Resequencer does not modify the message contents; it just delivers them in FIFO order.

Note: The ESB FIFO Resequencer delivers messages in the same order in which messages are received by the ESB routing service. However, in a flow, where ESB receives the input from a Database adapter or a File adapter, the processing may not be in a FIFO fashion because the Database adapter and the File adapter do not process messages in a FIFO fashion. Refer to Configuring Database Adapters and File Adapters for FIFO for workarounds that need to be applied on the File adapter and the Database adapter to make them process messages in a FIFO fashion.

- **Standard Resequencer**

  The Standard Oracle ESB Resequencer supports a standard Resequencer pattern, which is relevant for applications that use identifiers from a simple numeric identifier sequence in their messages. The Standard Resequencer receives a stream of messages that may not arrive in order. This uses an internal data store to the messages until a complete sequence based on the `SequenceId` is obtained. The messages are processed as soon as they are in order.

  The in-sequence messages, based on `SequenceId`, are then processed asynchronously in their own transaction one by one. It is important to note that the messages to outbound services of the Standard Resequencer routing service are guaranteed to arrive in sequence. The Standard Resequencer does not modify the message contents; it just orders them.

- **BestEffort Resequencer**

  The Oracle ESB Resequencer supports a `BestEffort` pattern, which is relevant for applications that produce a large number of messages in a short span of time and cannot inform the Oracle ESB Resequencer about the sequence identifier sequence. This way, ESB makes the best effort to ensure that the messages are delivered in sequence. The Oracle ESB `BestEffort` Resequencer can receive a stream of messages that are in order or just a little bit out of order. The messages are processed as soon as they are in order. This Resequencer can also reorder messages based on no knowledge of `SequenceId`. This means that you do not define the sequence such as with the Standard Resequencer.

  The BestEffort Resequencer uses an internal data store to store the messages until they are processed asynchronously. When the messages are processed, they are processed in sequence based on the specified `SequenceId` and the messages that have arrived at that point in time, no matter if a true sequence cannot be obtained. It is important to note that the messages to outbound services of the `BestEffort` routing service are not guaranteed to arrive in order based on `SequenceId`. This does not modify the message contents; it just tries to deliver them in sequence.

  You can use Date and Time data to sequence the messages, when the resequencing order is `BestEffort` Resequencer. For this, you need to add the endpoint property `ResequencerSequenceIdDataType`, which specifies that the data type of `ResequencerSequenceIdXPath` is `xsd:dateTime`. The Resequencer then interprets `ResequencerSequenceIdXPath` based on the `xsd:dateTime` type and sequences the messages.

Resequencer Thread Pool

Oracle Resequencer adds three new thread pools to the ESB run-time, which are used to deliver queued messages for resequencing:

- Locker Thread

  The locker threads are used to populate a work queue. The locker threads periodically search the database tables to find group IDs that can be processed by the worker threads. The locker threads put these group IDs in a work queue.

- Worker Thread

  The worker threads pick up the group IDs from the work queue and run the resequencing logic.

- HeartBeat Thread

  The heartbeat thread is used for unlocking groups that were locked by a crashed container. There will be only one instance of this thread.

Note: For Oracle ESB samples using Resequencer, visit the following URL

`http://www.oracle.com/technology/products/integration/esb/index.html`

Using the Resequencer

This section describes how you can use the resequencing feature. You will use the resequencing feature at the application design and development phase, when developing an ESB-based application and also while administering the deployed application. This section includes the following topics:

- Development on page 11-3
- Configuring the Dequeue System on page 11-5
- Configuring the Heartbeat Infrastructure on page 11-5
- Administration on page 11-6

Development

In the application design and development phase, you can enable resequencing support for the routing service by performing the following steps:

1. **Enable resequencing support in ESB** - by setting the value of the ESB configuration parameter `EnableResequencer` (in the `esb_config.ini` file or using the `-DEnableResequencer` option when ESB starts) to `true`. You must set the value of this ESB configuration parameter to provide sequencing for a routing service also.
2. **Enable resequencing support for a routing service** - by setting the `ResequencerType` endpoint property to `Standard`, `BestEffort`, or `FIFO`. You also need to set the value of the ESB configuration parameter `EnableResequencer` (in `esb_config.ini` or using the `-DEnableResequencer` option when ESB starts) to `true`.
3. **Specify the Group ID field** - by setting the `ResequencerGroupXPath` endpoint property expression that specifies the group identifier field in the incoming

messages. All the messages with the same value for the group ID field will form the set of incoming messages on which resequencing (of ResequencerType) will be done. If `ResequencerGroupXPath` is not specified, then all the messages received by the routing service will belong to one group.

The group ID XPath and sequence ID XPath use the following format:

```
{<<expression>>};{namespace ns1=<namespace url> namespace ns2=<namespace
url>...}.
```

The format is the same as that used in the subscription filters. An example of XPath is as follows:

```
{//ns1:owner};{namespace
ns1=http://www.w3schools.com}
```

You will have to set additional endpoint properties to configure the desired resequencing. These additional properties will vary with the desired resequencing and are mentioned in the table in step 4.

4. **Configure Standard Resequencing** - by specifying the endpoint properties mentioned in the following table:

Sl. no	Endpoint Property Name	Description
1.	`ResequencerSequenceIdXPath`	XPath that points to the field in the incoming message on which resequencing will be done.
2.	`ResequencerTimeout`	The time period in milliseconds to wait for an expected message. The Resequencer will lock the group as errored out if timeout happens. The group will be unlocked when a new message for the locked group arrives. The default value for timeout is `Long.MAX_VALUE`. **See Also**: Refer to "Resequencer Timeout Feature" for limitations of this endpoint property.
3.	`ResequencerSequenceStart` and `ResequencerSequenceIncrement`	The identifier sequence will be a natural number sequence, and the user will have to specify the `SEQUENCE_START` and `SEQUENCE_INCREMENT` values. The default value of both of these properties is 1.

5. **Configure BestEffort Resequencing** - by specifying the endpoint property mentioned in the following table:

Sl. no	Endpoint Property Name	Description
1.	`ResequencerSequenceIdXPath`	XPath that points to the field in the incoming message on which resequencing will be done.
2.	`ResequencerSequenceIdDataType`	Set its value to `dateTime` to pass an input of type `xsd:dateTime`. (*http://www.w3schools.com/Schema/schema_dtypes_date.asp*). By default, the data type is assumed to be a number.

6. **Configure FIFO Resequencing** - by specifying the endpoint property mentioned in the following table:

Sl. no	Endpoint Property Name	Description
1.	`ResequencerSequenceIdXPath`	XPath that points to the field in the incoming message on which resequencing will be done.

Configuring the Dequeue System

For a Resequencer-enabled routing service, the ESB enqueues the incoming messages to the database. The dequeue system of the Resequencer consists of a locker thread and a worker thread pool. The locker thread and the worker thread pool are tied to a system. Each system has its own locker thread and worker thread pool. The locker thread locks the groups to avoid concurrent access and then passes the locked groups to the worker thread pool. The worker thread pool retrieves the messages in bulk for a locked group and processes the messages. The following table describes the parameters to configure the dequeue system:

Name	Type	Description	Default
`ResequencerMaxGroupsLocked`	ECP	Number of groups locked by the dequeue system for processing.	Number of all available groups.
`ResequencerMaxRowsRetrieved`	EPP	Number of in-order messages that Resequencer should pick from the data store at a time.	1 for `BestEffort` and 5 for Standard and FIFO.
`ResequencerWorkerThreadPoolSize`	ECP	Number of threads per system for processing messages.	1
`systemName_ResequencerWorkerThreadPoolSize`	ECP	Number of threads per system for processing messages. The value of this parameter overrides the value of the `ResequencerWorkerThreadPoolSize` parameter for the particular system.	1
`ResequencerLockerThreadSleep`	ECP	Time in milliseconds for which the dequeue system thread sleeps, if it does not find a group for processing.	2000 milliseconds

where EPP is End Point Property and ECP is ESB Configuration Parameter.

Configuring the Heartbeat Infrastructure

The Resequencer's dequeue system locks a group and then processes the messages of the group. The dequeue system uses an instance identifier (ID) to lock a group. The instance ID is created and maintained by the heartbeat infrastructure. There is a single instance ID for every ESB instance. The heartbeat infrastructure creates this unique instance ID for the ESB instance. It inserts the instance ID, along with the current time into a table called `ESB_CONTAINERID_LEASE`.

The heartbeat infrastructure has one heartbeat thread. The heartbeat thread periodically updates the time associated with the instance ID to announce its presence to other ESB instances. The ESB configuration parameter `ResequencerContainerIdLeaseRefresh`, specified in minutes, is used for this purpose. The default value of this parameter is 10 minutes.

The heartbeat thread also looks for instance IDs that have not been updated within a configurable time period. The ESB configuration parameter `ResequencerContainerIdLeaseTimeout`, specified in minutes, is used for this purpose. The default value of this parameter is 15 minutes. The thread releases the

locks held by these instance IDs. Once the groups are unlocked, the dequeue system of other ESB instances can pick up the groups for processing.

Administration

The administrative tasks deal with the management of Resequencer support in ESB. In this release, the following administrative tasks are supported:

- Resubmitting an error-generating message.

  The message that generates an error is moved to the error hospital. The group is moved to an error state and is not picked up by the dequeue system of any ESB instance. The user can resubmit the error-generating message using the error hospital. On resubmission of the message, the group is moved to a normal state, so that it can be picked up by the dequeue system of any ESB instance.

- Managing timed out groups.

  A timed out group is not picked for processing. The `resequencer_list_timedout_groups.sql` SQL script can be used to list the routing service GUID, the group ID, and the group name of timed out groups. You can then pass this information to the `resequencer_restart_processing_group.sql` SQL script, so that a group is reopened for processing.

- Purging the message store.

  Resequencer does not delete any message that it receives. So, you must manually run a purge script to remove the processed messages that are older than a given time stamp. The purge script is a PL/SQL procedure called `resequencer_purge_messages` and is available in the `resequencer_purge_message_store.sql` file.

  > **Caution:** Use this script with caution to avoid unintended deletion of messages.

  For example, use the following command to delete all instances older than a given time stamp:

  ```
  resequencer_purge_messages(to_timestamp('2006-05-11 00:00:00','YYYY-MM-DD
  HH24:MI:SS'))
  ```

  Again, use the following command to delete all instances that are older than 120 days:

  ```
  resequencer_purge_messages(SYSDATE - 120)
  ```

- Managing Locked Groups.

  When a project is undeployed, the messages for a Resequencer enabled routing service are rejected. If the messages, already existing in the data store, are picked up for processing, then the Resequencer dequeue system is unable to process the messages and locks the groups because metadata for the routing service is not available. Such locked groups may be listed by using the PL/SQL procedure in the file `resequencer_list_erroredout_groups.sql`. This procedure lists the group identifier and routing service GUID for the groups that have reached an error state because the project was undeployed. You can pass the group ID and routing service GUID to the procedure `resequencer_restart_processing_group.sql` to reopen a group for processing.

Limitations in the Resequencer

The following limitations of Resequencer have been noted in this release:

- Change in Resequencer Type Not Supported on page 11-7
- Resequencer Timeout Feature on page 11-7
- Resequencer Endpoints Need to Be on the Same System on page 11-7
- Resequencer Fails If XSD File Contains Multibyte Characters That the Server Locale Encoding Does Not Support on page 11-7

Change in Resequencer Type Not Supported

In the 10.1.3.4 release, ESB provides three types of resequencing. The type of resequencing required by a routing service is specified at design time by setting the value of the `ResequencerType` endpoint property to any of the following: `Standard`, `BestEffort`, or `FIFO`. The Resequencer run-time maintains information about the state of the message sequence that is used to pick the next message for processing.

In the 10.1.3.4 release, the Resequencer does not support a change in the Resequencer type at design time, because the change in the Resequencer type will corrupt the state information about the message sequence. This may lead to undesirable effects such as messages not picked up for processing.

This issue is tracked with Oracle bug 7117296.

Resequencer Timeout Feature

For the standard Resequencer, an endpoint property called `ResequencerTimeout` can be used to set the time that the Resequencer should wait for a group's next message. If the next message is not received within the `ResequencerTimeout` period, then the Resequencer will mark the group in the timed out state and the group will not be picked up for processing.

In the 10.1.3.4 release, the timeout feature is not available. This is because of the absence of a user-friendly way to open the groups in the timed out state for processing. Currently, the timeout feature requires the use of SQL to open the groups in the timed out state. In this release, the default value of the `ResequencerTimeout` endpoint property is set to `Long.MAX_VALUE` (*http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Long.html#MAX_VALUE*).

This issue is tracked with Oracle bug 7117346.

Resequencer Endpoints Need to Be on the Same System

When, Resequencer is enabled on the routing service, then all routing rules in that routing service should connect to the endpoints that are on the same system.

This way, for routing rule targets belonging to different systems, such targets should be faced with Resequencer-enabled routing service.

This issue is tracked with Oracle bug 7143449.

Resequencer Fails If XSD File Contains Multibyte Characters That the Server Locale Encoding Does Not Support

If the XSD file contains multibyte characters that the server locale encoding does not support, then the resquencer execution fails after triggering the project flow. ESB control also shows failure.

This issue is tracked with Oracle bug 7218685.

Configuring Database Adapters and File Adapters for FIFO

This section describes how to make the Database adapters and the File adapters to work in FIFO fashion.

File Adapter

Perform the following for a File adapter:

1. Set the number of processor threads to 1 by performing the following steps:
   - Rename the `$ORACLE_HOME\integration\esb\config\pc.properties.esb` file to `$ORACLE_HOME\integration\esb\config\pc.properties` and set the value of the `oracle.tip.adapter.file.numProcessorThreads` parameter to 1.
   - Edit the `$ORACLE_HOME\bpel\system\service\config\pc.properties` file and set the value of the `oracle.tip.adapter.file.numProcessorThreads` parameter to 1. The default value of this parameter is 4.

     Note: For SOA installations, the settings specified in the BPEL file `$ORACLE_HOME\bpel\system\service\config\pc.properties` takes precedence over the corresponding ESB settings. So, you must edit this BPEL file to have the correct configuration settings for ESB.

2. Configure the inbound adapter by performing the following steps:
   1. In JDeveloper, edit the WSDL file of the inbound file adapter to add `Sorter="oracle.tip.adapter.file.sorter.TimestampSorterAscending"` to `ActivationSpec` in the following way:

      ```
      <jca:operation

      PhysicalDirectory="/home/Xiaocheng/oracle/product/10.1.3.1soa/resequencer/r
      esequencer/F2Fdata/in"
           ActivationSpec="oracle.tip.adapter.file.inbound.FileActivationSpec"
           Sorter="oracle.tip.adapter.file.sorter.TimestampSorterAscending"
           DeleteFile="true"
           IncludeFiles=".*\.xml"
           PollingFrequency="3"
           MinimumAge="2"
           OpaqueSchema="false" >
      </jca:operation>
      ```

      Note: The changes made to this file will not take effect at this point.

   2. Save the file.
   3. In JDeveloper, open the `.esbsvc` file of the inbound file adapter.
   4. Click **Configure tech adapter service wsdl** and follow the wizard without making any modification to the WSDL file.
   5. Save the file.

6. Open the WSDL file of the inbound file adapter again and check whether or not the settings you made in Step 1 are intact and the file looks like the following:

```
<jca:operation

PhysicalDirectory="/home/Xiaocheng/oracle/product/10.1.3.1soa/resequencer/r
esequencer/F2Fdata/in"
     ActivationSpec="oracle.tip.adapter.file.inbound.FileActivationSpec"
     Sorter="oracle.tip.adapter.file.sorter.TimestampSorterAscending"
     DeleteFile="true"
     IncludeFiles=".*\.xml"
PollingFrequency="3"
MinimumAge="2"
OpaqueSchema="false" >
</jca:operation>
```

3. Register the project with ESB again.

   You will receive a notification regarding successful registration.

Note:

- After you complete the settings, the file adapter guarantees to send the files to the ESB Routing Service based on the time stamp of the files. However, if you check the time stamp of the files by using the `ls --full-time` command, then you will see the time to be like '2008-06-25 09:36:45.000000000 +0800', that is, the granularity is on second level. So, if files are copied into the folder of the adapter too fast, then they will have the same time stamp. In such a situation, this workaround cannot guarantee that the order is FIFO because the messages will have the same time of arrival.
- If you make the thread, copying the files, sleep for 1000 milliseconds, then you are guaranteed to have s different time stamp for each file.

Database Adapter

Perform the following for a Database adapter:

1. Set the number of Database threads to 1 by performing the following steps:
   1. In JDeveloper, open the WSDL file of the inbound database adapter.
   2. Set the value of the `NumberOfThreads` parameter to 1 in the following way:

```
<jca:operation
                ActivationSpec="oracle.tip.adapter.db.DBActivationSpec"
                DescriptorName="GetTransactions.OrderTransactionsFifo1"
                QueryName="GetTransactions"
                PollingStrategyName="DeletePollingStrategy"
                SequencingFieldName="TS"
                MaxRaiseSize="1"
                MaxTransactionSize="unlimited"
                PollingInterval="5"
                NumberOfThreads="1"
                UseBatchDestroy="false"
                ReturnSingleResultSet="false"
                MappingsMetaDataURL="GetTransactions_toplink_mappings.xml"
```

```
/>
```

3. Save the file.

2. Add a column to store the time stamp when a row is inserted into the table so that you can sort by this column while configuring the Database adapter. Perform the following steps to achieve this:

   1. In JDeveloper, open the `.esbsvc` file of the inbound file adapter.
   2. Click **Configure tech adapter service wsdl** and follow the wizard.
   3. On the Select Table page, import the table of the Database adapter again to add the newly added timestamp column.
   4. If the table does not have a primary key, then specify the time stamp column as the virtual primary key.
   5. On the Polling Options page, specify time stamp as the value of the **Order By** field.
   6. Save the file.
   7. Open the WSDL file again to check whether or not `SequencingFieldName` is set as the column name of the time stamp column (TS). The WSDL file should look like the following:

```
<jca:operation
                ActivationSpec="oracle.tip.adapter.db.DBActivationSpec"
                DescriptorName="GetTransactions.OrderTransactionsFifo1"
                QueryName="GetTransactions"
                PollingStrategyName="DeletePollingStrategy"
                SequencingFieldName="TS"
                MaxRaiseSize="1"
                MaxTransactionSize="unlimited"
                PollingInterval="5"
                NumberOfThreads="1"
                UseBatchDestroy="false"
                ReturnSingleResultSet="false"
                MappingsMetaDataURL="GetTransactions_toplink_mappings.xml"
/>
```

3. Register the project with ESB again.

   You will receive a notification regarding successful registration.

12

Error Handling

This chapter describes how to interpret error conditions and how to handle them in Oracle Enterprise Service Bus. The Instances view in the Oracle ESB Control enables you to view and manage error conditions that occur across the enterprise service bus.

This chapter contains the following topics:

- Overview of Error Handling on page 12-1
- Managing Error Conditions on page 12-2
- ESB Error Handling Enhancement on page 12-5

You can also view the log files with Oracle Enterprise Manager to check for details on error conditions. See "Checking Log Files" on page 9-11.

Overview of Error Handling

When an error occurs in Oracle Enterprise Service Bus processing, the error is noted by visual cues, such as icon and color changes, in the Oracle ESB Control. See Figure 10–4 on page 10-3, Figure 10–5 on page 10-4, and Figure 10–6 on page 10-5. Also, you can also set up notifications by email, fax, or phone when errors occur. See "Setting Up Notification Channels" on page 9-4.

Error handling in Oracle Enterprise Service Bus involves several types of errors that can occur in transaction processing:

- Application or Business Faults
- Retryable Exceptions: A temporary loss of a service impacting the routing of the message, but is usually resolved in a relatively short time period
- Fatal Exceptions: A disabled or deleted service or system, out of memory condition, or other serious problem that results in a serious error requiring attention by a system administrator

Error handling is processed differently whether asynchronous and synchronous execution is specified for a routing service. Errors during synchronous execution are rolled back and cannot be retried with Oracle ESB Control; errors during asynchronous execution can be resubmitted. For more information about asynchronous and synchronous execution, see "Specifying Synchronous or Asynchronous Execution" on page 5-27.

- For synchronous execution, the transaction is rolled back and an error notification is returned to the adapter that initiated the processing. The calling adapter is expected to handle the error and resubmission. See "Inbound Adapter Error Handling" on page 12-2.

- For asynchronous execution, the user can resubmit the transaction after the error condition has been resolved. See "User Error Handling" on page 12-2.

Managing Error Conditions

This section discuses how to manage errors conditions that occur with Oracle Enterprise Service Bus.

The topics contained in this section are:

- Inbound Adapter Error Handling on page 12-2
- User Error Handling on page 12-2
- Resubmitting Messages on Errors on page 12-4

Inbound Adapter Error Handling

An inbound adapter handles exceptions and faults using the default error handling process of the adapter.

- By default, an adapter retries the message three times at five second intervals for an error condition. The retry count and interval can be specified in the endpoint properties of the adapter service. For information about endpoint properties, see "Using Endpoint Properties" on page 4-11.
- If an inbound adapter fails to invoke a routing service for a certain number of consecutive times, it marks itself broken and disables itself. The Oracle ESB Control displays this event source with a special icon to visually represent its disabled state. You can enable the adapter service.
- If the next service that the inbound adapter invokes does not exist, perhaps because it has been deleted or is not enabled, then the inbound adapter processor disables itself and marks itself broken.
- If a subscription fails a certain number of consecutive times, the service notifies the repository to mark it as in a broken state. The dispatcher does not dispatch this subscription after it is marked as broken.

The administrator is notified of any disabled services.

For more information about adapters, including a discussion of managing errors, see *Oracle Application Server Adapter Concepts*.

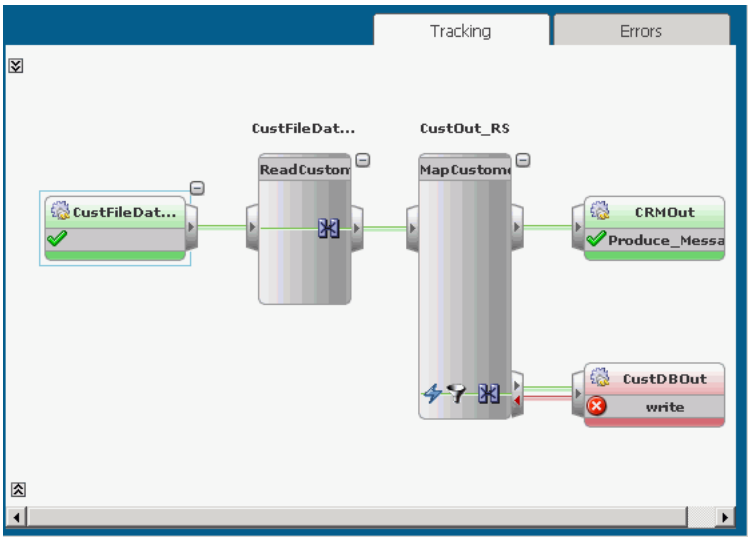
See Also: Inbound Adapter Error Handling on page 12-2

User Error Handling

When an error occurs during process a message instance, it is indicated by a red service icon in the Tracking tab of the Oracle ESB Control Instances view.

An error condition in the Tracking tab appears similar to Figure 12–1.

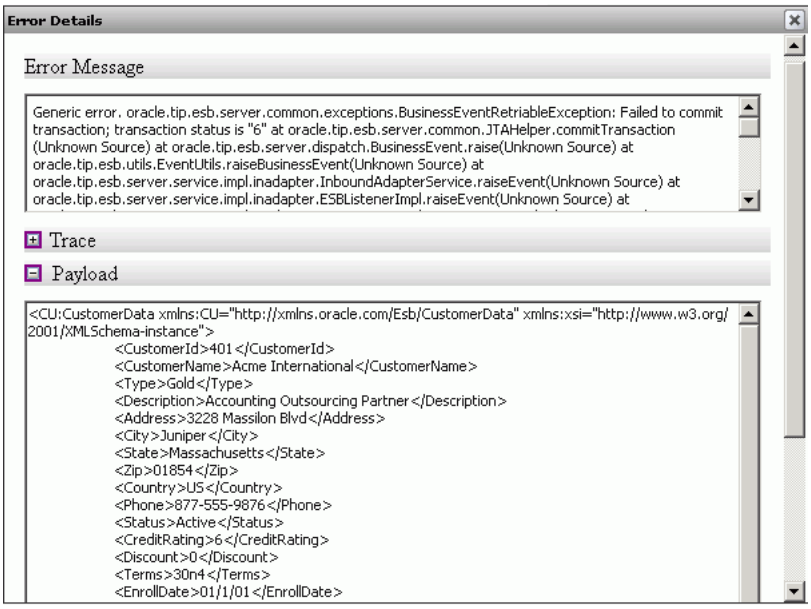
Figure 12–1 Instances View – Tracking Tab



You can view the error message, trace, and payload details by clicking on the **Error Details** icon in the Message column on the Errors tab, shown in Figure 12–3.

Figure 12–2 is an example of the Error Details dialog.

Figure 12–2 Error Details Dialog



After you review the error details and fix the error condition, you can resubmit the message to the invoking service.

For information about enabling validation of the payload at runtime, see Table 3–6 on page 3-14 in "Viewing Service Definitions" on page 3-12.

Resubmitting Messages on Errors

In some situations, a message instance can be resubmitted. If a routing rule has been set to asynchronous execution, you can usually resubmit a message instance after fixing the error condition.

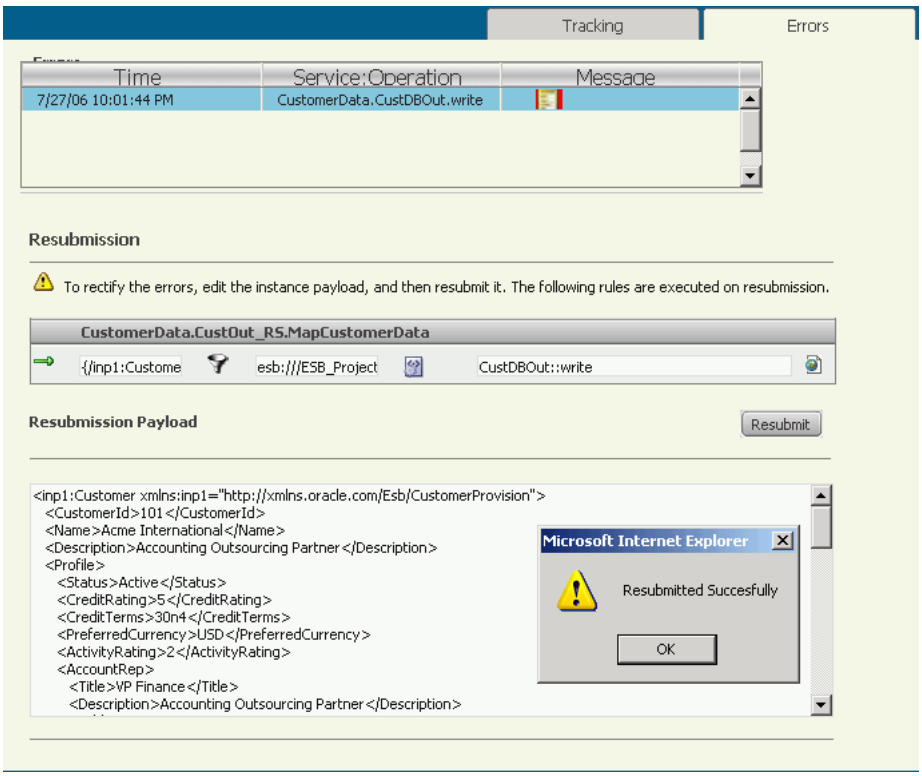
To resubmit a message on error, follow these steps:

1. At the top of the Oracle ESB Control, click the **Instances** icon to display the message instance processing.
2. In the Instances panel of the Instances view, click the message instance where the error occurred.
3. Click the Error tab to display the error information.
4. Click the **Error Details** icon under the Message column to view error message, trace, and payload details about the error condition in the Errors Detail dialog.
5. Click **OK** to close the Errors Detail dialog after reviewing the error message details.
6. Correct the error condition, then click **Resubmit** in the Error tab.

   For example, edit the message payload in the Resubmission Payload window if is incorrect and then click **Resubmit**.

Figure 12–3 is an example of the resubmitted message instance.

Figure 12–3 Instances View - Errors Tab



ESB Error Handling Enhancement

Oracle Application Server release 10.1.3.4 provides additional support for error handling in ESB. The error behavior in ESB can be explained with the following scenario:

```
Third-party Adapter --> Routing Service --> AQ Outbound Adapter
```

The error handling in ESB has the following enhancements in 10.1.3.4:

- Custom Rejection Handler for Third-Party Adapters on page 12-5
- Shutdown of the Inbound Adapter on page 12-5
- Enhance the Information in ERROR_TOPIC on page 12-6
- Reverse Control on page 12-6
- Primary Key on page 12-6
- Removal of Message from the Queue on Rejection on page 12-7

Custom Rejection Handler for Third-Party Adapters

Third-party adapters such as the SAP adapter do not have support for the custom rejection handler. But, in the 10.1.3.4 release, you can invoke the custom rejection handler from ESB. You need to set up the `SupportCustomRejectionForThirdPartyAdaptor` endpoint property to `true`. ESB identifies this property as a third-party adapter and calls the `onReject` function.

The property name and description used for this feature are as follows:

Property Name: `SupportCustomRejectionForThirdPartyAdaptor`

Description: True refers to invoking custom rejection handler for third-party.

Shutdown of the Inbound Adapter

Whenever ESB fails to post the message on `ERROR_TOPIC`, you might want to shut down the inbound adapter to avoid any further loss of error messages. You can achieve this by setting the `shutdownInboundAdaptorOnError` endpoint property to `true`. This property can be used only for the synchronous ESB service and when the messages on the `ERROR_TOPIC` AQJMS topic are posted from the inbound service. By default, the messages on the `ERROR_TOPIC` AQJMS topic are enqueued at the service where the error has happened. To enable the messages on `ERROR_TOPIC` to be enqueued from the inbound service, you need to set the `EnqueueErrorFromInbound` to `true` in the `ESB_config.ini` file.

Table 12–1 summarizes the properties that need to be configured for this feature.

Table 12–1 Properties to Be Set for Shutting Down the Inbound Adapter

Property Name	Description
`shutdownInboundAdaptorOnError`	If the value of this property is `true`, then it indicates a shutdown of the adapter on the CURRENT node only. If `EnqueueErrorFromInbound= false`, then it will not shut down the adapter.
`EnqueueErrorFromInbound`	If the value of this property is `true`, then error should be enqueued on `ERROR_TOPIC` from the inbound service and only then `INBOUND_ADAPTOR_ID` can be set on `ERROR_TOPIC` and eventually the inbound adapter shutdown can happen.

See Also: Reverse Control on page 12-6

Enhance the Information in ERROR_TOPIC

The `ERROR_TOPIC` AQJMS topic contains additional information such as inbound adapter ID (`INBOUND_ADAPTOR_ID`) and outbound adapter ID (`OUTBOUND_ADAPTOR_ID`).

Reverse Control

The reverse control mechanism helps in a scenario where you want all the inbound adapters in the cluster to be shut down and also the status to be updated on the console. You can use the reverse control mechanism, where the design-time is notified by the run time. The run-time updates the status of the service and notifies all the run time services about the change in state. Then, all the run-time services shut down the inbound adapter. This shutdown happens only when the `disable_service_on_all_nodes` endpoint property is set to `true` in the `ESB_config.ini` file.

The reverse control mechanism is used in conjunction with `shutdownInboundAdaptorOnError`. This means that when posting a message on `ERROR_TOPIC` fails, the inbound adapter is shut down and then, if the system is configured such that all the other nodes need to be shut down, then reverse control kicks in.

Property Name: `disable_service_on_all_nodes`

Description: The preceding use case is supported when `disable_service_on_all_nodes=true`. If it is set to `false`, then only the adapter on the current node will be shut down and the status on the console will not be updated.

Primary Key

All the technology adapters provide a unique key to identify the message. ESB uses this unique key as the primary key to determine how many times the message has been retried before it can be rejected. Third-party adapters do not provide this unique key, which results in the message never getting rejected and the message is always sent to the adapter for a retry. ESB provides the `PrimaryKey` endpoint property, which is an XPATH from the payload. This enables a message to be related using this XPATH, which is then retried and rejected eventually.

A sample to define a primary key is as follows:

Sample payload XML:

```
<ns1:userDetail xmlns:ns1="http://www.example.org">
            <ns1:key>41</ns1:key>
            <ns1:value>11</ns1:value>
            </ns1:userDetail>
PrimaryKey={/imp1:userDetail/imp1:key};{namespace imp1=http://www.example.org}
```

Property Name: `PrimaryKey`

Description: If the `PrimaryKey` endpoint property is not set for third-party adapter, it will always result in a `RetriableException` error, requesting the adapter to retry the same message. This way, the custom rejection handler will never be called. When the primary key with an XPATH returns NULL, the message will not be retried and will be rejected after the first failure in processing.

Removal of Message from the Queue on Rejection

The AQ adapter, which is a technology adapter, removes the message from the main queue and places it in its own error queue when it receives a `RejectionException` error. This way, the messages are not sent back to ESB again. On the other hand, third-party adapters such as the SAP adapter do not have this built-in mechanism. The message remains in the main queue forever. Also, the SAP adapter does not participate in the global transaction. When you set the `NotifyThirdPartyAdaptor` endpoint property to `false`, ESB rolls back its local transaction, which invokes the rejection handler but never returns the exception to the SAP adapter. The SAP adapter then commits its transaction, and the message is removed from the queue.

Property Name: `NotifyThirdPartyAdaptor`

Description: `False` value states that do *not* bubble the exception back to the third-party adapter. ESB will not throw the exception back to the SAP adapter, and the value will be picked from the SAP adapter. For third-party adapters, if the `PrimaryKey` endpoint property is not set, then it will result in ESB always sending the notification to the adapter for a retry.

Table 12–2 describes the property relationship table and the detailed description of the actions that would occur when you set the properties in the order specified in the table.

Table 12–2 Property Relationship Table

SL.No.	EnqueueErrorFromInbound	disable_service_on_all_nodes	shutdownInboundAdaptorOnError	SupportCustomRejectionForThirdPartyAdaptor	NotifyThirdPartyAdaptor	PrimaryKey	Description
1.	true	true	true	true	false	XPATH resulting in NOT NULL value	1. If ESB infra is up, inbound adapter id set on ERROR_TOPIC. Message placed on ERROR_TOPIC. 2. If ESB infra is down, inbound adapter is disabled on all nodes and status updated on the console. 3 Else message is retried based on the inbound and outbound based on parameters set. 4. Custom Rejection Handler is called after the number of inbound retries are exhausted. 5. Message picked from SAP adapter.
2.	true	true	true	true	false	End point property set but XPATH resulting in NULL value	1. If ESB infra is up, inbound adapter id set on ERROR_TOPIC. Message placed on ERROR_TOPIC. 2. If ESB infra is down, inbound adapter is disabled on all nodes and status updated on the console. 3 Else message NOT tried again at the inbound service layer and `RetryCount` is considered only at the outbound layer. 4. Custom Rejection Handler is called. 5. Message picked from SAP adapter.
3.	true	true	true	true	false	end point property not set	1. If ESB infra is up, inbound adapter id set on ERROR_TOPIC. Message placed on ERROR_TOPIC. 2. If ESB infra is down, inbound adapter is disabled on all nodes and status updated on the console. 3 Else because the primary key is not set and also it is NULL for SAP adapter, ESB will continue to retry. 4. Custom Rejection Handler is NOT called. 5. Message is NOT picked from SAP adapter.

Table 12–2 (Cont.) Property Relationship Table

SL.No.	EnqueueErrorFromInbound	disable_service_on_all_nodes	shutdownInboundAdaptorOnError	SupportCustomRejectionForThirdPartyAdaptor	NotifyThirdPartyAdaptor	PrimaryKey	Description
4.	false	true	true	true	false	XPATH resulting in NOT NULL value	1. If ESB infra is up, inbound adapter id set on ERROR_TOPIC. Message placed on ERROR_TOPIC. 2. If ESB infra is down, inbound adapter is NOT disabled on the current node as well as all the other nodes. 3 Else message is retried on the inbound and outbound based on parameters set. 4. Custom Rejection Handler is called after the number of inbound retries are exhausted. 5. Message picked from SAP adapter.
5.	true	false	true	true	false	XPATH resulting in NOT NULL value	1. If ESB infra is up, inbound adapter id set on ERROR_TOPIC. Message placed on ERROR_TOPIC. 2. If ESB infra is down, inbound Adapter is disabled on the current but not on all other running nodes. Status is not updated on the console. 3. Else message is retried on the inbound and outbound based on parameters set. 4. Custom Rejection Handler is called after the number of inbound retries are exhausted. 5. Message picked from SAP adapter.

Table 12–2 (Cont.) Property Relationship Table

SL.No.	EnqueueErrorFromInbound	disable_service_on_all_nodes	shutdownInboundAdaptorOnError	SupportCustomRejectionForThirdPartyAdaptor	NotifyThirdPartyAdaptor	PrimaryKey	Description
6.	true	true	false	true	false	XPATH resulting in NOT NULL value	1. If ESB infra is up, inbound adapter id set on ERROR_TOPIC. Message placed on ERROR_TOPIC. 2. If ESB infra is down, inbound Adapter is NOT disabled on the current node as well as all the other nodes. 3 Else message is retried on the inbound and outbound based on parameters set. 4. Custom Rejection Handler is called after the number of inbound retries are exhausted. 5. Message picked from SAP adapter.
7.	true	true	true	false	false	XPATH resulting in NOT NULL value	1.If ESB infra is up, inbound adapter id set on ERROR_TOPIC. Message placed on ERROR_TOPIC. 2. If ESB infra is down, inbound adapter disabled on all nodes and status updated on the console. 3 Else message is retried on the inbound and outbound based on parameters set. 4. Custom Rejection Handler is NOT called for third-party adapters after the number of inbound retries are exhausted. 5. Message picked from SAP adapter.
8.	true	true	true	true	true	XPATH resulting in NOT NULL value	1. If ESB infra is up, inbound adapter id set on ERROR_TOPIC. Message placed on ERROR_TOPIC. 2. If ESB infra is down, inbound adapter disabled on all nodes and status updated on the console. 3 Else message is retried on the inbound and outbound based on parameters set. 4. Custom Rejection Handler is called after the number of inbound retries are exhausted. 5. Message is NOT picked from SAP adapter as the exception is bubbled back to the adapter.

13

Enterprise Service Bus Ant Deployment Feature

This chapter describes support for the ESB Ant deployment feature in the Oracle Application Server release 10.1.3.4.

This chapter contains the following topics:

- Overview of Ant Deployment Feature on page 13-1
- Ant Installation on page 13-1
- Deployment of Ant Tasks on page 13-2
- ESB Ant Deployment Custom Tasks on page 13-4

Overview of Ant Deployment Feature

The Ant deployment feature includes a set of custom Ant tasks that are used for ESB service deployment automation. ESB Ant deployment feature supports working with ESB metadata and enables you to deploy or undeploy the metadata to an ESB metadata (design time) server using custom Ant tasks, with or without property substitution taking place.

> **See Also:** *http://ant.apache.org/* for more information about the Apache Ant tool

Ant Installation

You need to download and install Ant from this location

`http://ant.apache.org/bindownload.cgi`

Perform the following steps:

1. Install Ant and the dependent JAR files.
2. Edit the `ESBMetadataMigrationTaskdefs.xml` file to point to your JAR files.
3. In JDeveloper, create a simple ESB project called `AntTest` with a routing service and a fileout adapter.
4. Create an `AntTestBuild.xml` file, as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<project name="AntTest" default="usage">
    <property name="esbProjectLocation" value="C:\mywork\ESBTests\AntTest"/>
    <property name="deploymentPlanFile" value=".\AntTestDeploymentPlan.xml"/>
  <import file="${basedir}/ESBMetadataMigrationTaskdefs.xml"/>
```

```
  <target name="AntTest">
    <delete file="${deploymentPlanFile}"/>
    <extractESBDeploymentPlan
        sourceDir= "${esbProjectLocation}"
        deploymentPlanFile="${deploymentPlanFile}"/>
  </target>
    <target name="AntTestDeploy">
<deployESBProjects esbMetadataServerHostname="localhost"
          esbMetadataServerPort="8888"
                            userName="oc4jadmin"
                            password="welcome1">
   <esbProject directory="${esbProjectLocation}"/>
</deployESBProjects>
    </target>
<target name="AntTestDeployPlan">
    <deployESBSuitcase sourceDirectory="${esbProjectLocation}"
           deploymentPlanFilename="${deploymentPlanFile}"
                       esbMetadataServerHostname="localhost"
                       esbMetadataServerPort="8888"
                       forcedDeployment="true"/>
</target>
   <target name="usage">
<exec executable="ant" dir="${basedir}" vmlauncher="false">
    <arg value="-projecthelp"/>
</exec>
    </target>
</project>
```

5. Set the path to access JRE and Ant binaries.
6. Deploy the project using JDeveloper. This has to be done only once in the JDeveloper environment.
7. Run the extracted Ant task using the following command: `"ant -f AntTestBuild.xml AntTest."`
8. Optionally, edit the newly created `AntTestDeploymentPlan.xml` file.
9. Delete the project from the ESB console.
10. Deploy the project with the plan `"ant -f AntTestBuild.xml AntTestDeployPlan."` To deploy directly without a plan, you can run the deploy project Ant task `"ant -f AntTestBuild.xml AntTestDeploy."`
11. Test using the embedded EM test page.

Deployment of Ant Tasks

You can use the Ant scripts to register an ESB project from the command line. To do this, you need to download the ESB deployment package from the following location:

`OracleAS_HOME\integration\esb\deployment\documentation.zip`

which contains the following custom tasks:

- The `deployESBProjects` task enables one to many ESB projects in JDeveloper to be deployed to a given ESB metadata server. The ESB project artifacts found in the identified projects will be deployed as is, that is, without any property substitution taking place during the deployment process. This task is the automated equivalent of manually visiting each ESB project within JDeveloper, to exercise the **Register with ESB** function in that context.

- The `undeployESBEntities` task enables one to many ESB systems, service groups, or services to be deleted (or undeployed) from the identified ESB metadata server, where each entity to be undeployed is uniquely identified by its `guid`. This task is the automated equivalent of manually locating each ESB system, service group, or service within the ESB console and, then clicking **Delete**.
- The `extractESBDeploymentPlan` task supports the extraction of all properties that are allowed to vary for all of the ESB systems, service groups, and services found within a single ESB project that is owned and maintained by JDeveloper. The extracted collection of properties is known as a deployment plan and is placed into an XML document on disk.
- The `deployESBSuitcase` task supports deploying the ESB project artifacts found in the identified JDeveloper ESB project directory (and subdirectories) to the identified ESB metadata server, while applying the properties found in the specified deployment plan to those artifacts. Both the ESB project artifacts and the deployment plan are read-only.

Deployment Ant Tasks Sample File

Example 13–1 describes the `ESBMetadataMigrationTaskdefs.xml` sample file.

Example 13–1 The ESBMetadataMigrationTaskdefs.xml Sample File

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<project name="ESBMetadataMigrationTaskdefs">
<!-- PROPERTIES, unique  to the installation environment -->
 <property name="commons.httpclient.home" value="C:/src/esbDeploymentAutomation"/>
 <property name="jaxb.v2.0.2.home"
 value="C:/src/esbDeploymentAutomation/jaxb-ri-20070122"/>
 <property name="soa.suite.home" value="C:/product/10.1.3.1/OraESB_1"/>
<!--- Custom ant task definitions, do not change below this line -->
    <dirname property="imported.basedir"
 file="${ant.file.ESBMetadataMigrationTaskdefs}"/>
    <taskdef resource="oracle/tip/esb/client/anttasks/antlib.xml">
        <classpath>
            <pathelement location="${imported.basedir}/ESBMetadataMigration.jar"/>
            <pathelement
 location="${commons.httpclient.home}/commons-httpclient-3.0.1.jar"/>
            <pathelement location="${soa.suite.home}/lib/xmlparserv2.jar"/>
            <pathelement
 location="${soa.suite.home}/integration/esb/lib/commons-logging.jar"/>
            <pathelement
 location="${soa.suite.home}/integration/esb/lib/commons-codec-1.3.jar"/>
            <pathelement
 location="${soa.suite.home}/integration/esb/lib/oraesb.jar"/>
            <pathelement location="${jaxb.v2.0.2.home}/lib/activation.jar"/>
            <pathelement location="${jaxb.v2.0.2.home}/lib/jaxb-api.jar"/>
            <pathelement location="${jaxb.v2.0.2.home}/lib/jsr173_1.0_api.jar"/>
            <pathelement location="${jaxb.v2.0.2.home}/lib/jaxb-impl.jar"/>
        </classpath>
    </taskdef>
</project>
```

Example 13–2 describes the `AntTestBuild.xml` sample file.

Example 13–2 The AntTestBuild.xml Sample File

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<project name="AntTest" default="usage">
```

```
    <property name="esbProjectLocation" value="C:\mywork\ESBTests\AntTest"/>
    <property name="deploymentPlanFile" value=".\AntTestDeploymentPlan.xml"/>
  <import file="${basedir}/ESBMetadataMigrationTaskdefs.xml"/>
  <target name="AntTest">
    <delete file="${deploymentPlanFile}"/>
    <extractESBDeploymentPlan
        sourceDir= "${esbProjectLocation}"
        deploymentPlanFile="${deploymentPlanFile}"/>
  </target>
    <target name="AntTestDeploy">
<deployESBProjects esbMetadataServerHostname="localhost"
 esbMetadataServerPort="8888"
userName="oc4jadmin" password="welcome1">
      <esbProject directory="${esbProjectLocation}"/>
</deployESBProjects>
    </target>
<target name="AntTestDeployPlan">
    <deployESBSuitcase sourceDirectory="${esbProjectLocation}"
deploymentPlanFilename="${deploymentPlanFile}"
esbMetadataServerHostname="localhost" esbMetadataServerPort="8888"
forcedDeployment="true"/>
</target>
   <target name="usage">
<exec executable="ant" dir="${basedir}" vmlauncher="false">
    <arg value="-projecthelp"/>
</exec>
    </target>
</project>
```

ESB Ant Deployment Custom Tasks

This section describes the following ESB Ant deployment custom tasks:

- deployESBProjects on page 13-4
- undeployESBEntities on page 13-6
- extractESBDeploymentPlan on page 13-7
- deployESBSuitcase on page 13-9

deployESBProjects

deployESBProjects is a custom Ant task that can be used to deploy one to many ESB projects, which are created and maintained by JDeveloper, to a given ESB metadata server. This task provides identical behavior to that of JDeveloper's **Register with ESB** function, with the additional functionality of Automatic refresh or resynchronization of the JDeveloper-owned and maintained ESB project artifacts. If JDeveloper is used to manually deploy one or more of those given ESB projects using the **Register with ESB** function, then the ESB project metadata first needs to be refreshed from the ESB metadata server.

Parameters

The attribute and description of the parameters are described in the following table:

Attribute	Description	Required
esbMetadataServerHostname	The host name of the computer that is supporting the ESB Metadata Server of interest.	Yes
esbMetadataServerPort	The port number of the ESB Metadata Server.	Yes
userName	The login name that has permission to deploy or undeploy ESB systems, service groups, and services using the ESB console, typically, `oc4jadmin`.	Yes
password	The password for the `userName` account.	Yes

Note: The `esbMetadataServerPort` for a given ESB metadata server instance can be identified by the port number used to interact with the ESB control/console for the given machine. The URL used to interact with the ESB control/console for the desired ESB instance will use that same port number.

Parameter Specified as Nested Elements

The `deployESBProjects` custom task supports one to many nested `esbProject` tasks. An `esbProject` task cannot be nested within another `esbProject` task. The parameters for an `esbProject` task are documented in the following table:

Attribute	Description	Required
directory	The top-level directory of an ESB project created/maintained by JDeveloper.	Yes

System Response

The custom Ant task will intimate you with the semantic validation errors in the case where the task has not been defined properly, as well as the output, informing you of its progress. On completion of the task, open the ESB console to ensure that your ESB systems, service groups, and services have been deployed to the ESB metadata server of interest.

Examples

Example 13–3 requests that the five ESB projects be deployed to the ESB metadata server that is running on the local host and is to be contacted on the local host at port 8889. The ESB projects will be deployed in the following order: `CustomerData`, `CustomerData5`, `CustomerData8`, `CustomerData9`, and `CustomerData10`. It is important to note that the directories specified are the top-level directory for each of those five projects. Within each directory (or one of its subdirectories), ESB project artifacts (files with the following file name extensions: `.esb`, `.esbgrp`, `.esbsvc`, `.esbsys`, `.xml`, `.xsd`, `.xsl`, `.xslt`, and `.wsdl`) can be found.

Example 13–3 Example for deployESBProjects

```
<target name="test.DeployESBProjects">
    <deployESBProjects esbMetadataServerHostname="localhost"
     esbMetadataServerPort="8889"
                        userName="oc4jadmin"
                        password="oc4jadmin">
                        protocol="https">
                       <truststore="C:\JDev_10134\jdk\jre\lib\security\cacerts">
```

```
        <esbProject
directory="C:/Oracle/product/JDeveloper/v10.1.3.1/jdev/mywork/ESBSamples/CustomerD
ata"/>
        <esbProject
directory="C:/Oracle/product/JDeveloper/v10.1.3.1/jdev/mywork/ESBSamples/CustomerD
ata5"/>
        <esbProject
directory="C:/Oracle/product/JDeveloper/v10.1.3.1/jdev/mywork/ESBSamples/CustomerD
ata8"/>
        <esbProject
directory="C:/Oracle/product/JDeveloper/v10.1.3.1/jdev/mywork/ESBSamples/CustomerD
ata9"/>
        <esbProject
directory="C:/Oracle/product/JDeveloper/v10.1.3.1/jdev/mywork/ESBSamples/CustomerD
ata10"/>
    </deployESBProjects>
</target>
```

undeployESBEntities

`undeployESBEntities` is a custom Ant task that can be used to undeploy one to many ESB entities (systems, service groups, and services) that have been previously deployed to the identified ESB metadata server. For simplified dependency management and error handling, all identified services are undeployed first, followed by all identified service groups, followed by all identified Systems. The task does require that at least one system, service group, or service be specified to be undeployed. ESB systems, service groups, and services identified for undeployment can be specified in any order.

Parameters

The attribute and description of the parameters are described in the following table:

Attribute	Description	Required
esbMetadataServerHostname	The host name of the computer that is supporting the ESB Metadata Server of interest.	Yes
esbMetadataServerPort	The port number of the ESB Metadata Server.	Yes
userName	The login name that has permission to delete ESB Systems, Service Groups, and Services using the ESB console, typically, `oc4jadmin`.	Yes
password	The password for the `userName` account.	Yes

Note: The `esbMetadataServerPort` for a given ESB metadata server instance can be identified by the port number used to interact with the ESB control/console for the given machine. The URL used to interact with the ESB control/console for the desired ESB instance will use that same port number.

Parameter Specified as Nested Elements

The `undeployESBEntities` custom task supports one to many nested system, service group, and service tasks. These nested tasks cannot be nested within each other, as each task must be a peer of the others. The parameters for the system, service group, and service task are identical and are documented in the following table:

Attribute	Description	Required
guid	This value uniquely identifies the ESB entity that is to be undeployed. The guid of any ESB entity can be found in the guid attribute of the System, Service Group, or Service element of the corresponding .esbsys, .esbgrp, or .esbsvc file, respectively, that JDeveloper maintains. The guid for each entity can also be found in the guid attribute of the elements in the metadata.xml file that is produced using ESB's export command-line utility. Note: The `metadata.xml` file is housed in the ZIP file that the export command-line utility produces.	Yes

System Response

The custom Ant task will intimate you with the semantic validation errors in the case where the task has not been defined properly, as well as the output, informing you of its progress. On completion of the task, open the ESB console to see that your ESB systems, service groups, and services have been undeployed from the ESB metadata server of interest.

Examples

Example 13–4 requests that the five ESB systems, two service groups, and two services identified be undeployed from the ESB metadata (design time) server that is running on the local host, and is to be contacted on the local host at port 8889. Again, for simplified dependency management and error handling, the two identified Services are undeployed first, followed by the two identified service groups, followed by the five identified systems.

Example 13–4 Example for undeployESBEntities

```
<target name="test.UndeployESBEntities">
    <undeployESBEntities esbMetadataServerHostname="localhost"
                         esbMetadataServerPort="8889"
                         userName="oc4jadmin"
                         password="oc4jadmin">
                         protocol="https">
        <system guid="8D61C3F0871111DB8F2675C60E6C31C6"/>
        <serviceGroup guid="0EB5F380896111DBBFBC9530C01627AC"/>
        <service guid="0547F370841611DBBFCF2D9BF80323FA"/>
        <service guid="05458270841611DBBFCF2D9BF80323FA"/>
        <system guid="86443990871611DB8F2675C60E6C31C6"/>
        <serviceGroup guid="B90E6B70895F11DBAF1483EEF470B835"/>
        <system guid="A62C91C1841511DBBFCF2D9BF80323FA"/>
        <system guid="D9F290E1896011DBBFBC9530C01627AC"/>
        <system guid="A9B213C1895F11DBAF1483EEF470B835"/>
    </undeployESBEntities>
</target>
```

extractESBDeploymentPlan

`extractESBDeploymentPlan` is a custom Ant task that supports extracting the deployment plan from a JDeveloper ESB project source directory into the identified `deploymentPlanFile`.

Preconditions

The identified ESB project must have been deployed at least once using JDeveloper's **Register with ESB** function. This is due to the fact that the underlying ESB artifacts

will be modified as a result of the **Register with ESB** function, and this task expects the artifacts to be in that fully specified state.

Additionally, the deployment plan file to be created is protected from being inadvertently overwritten by the execution of this task. If the deployment plan file to be created already exists when this task is executed, then the task will be terminated and inform you of this condition.

Parameters

The attribute and description of the parameters are described in the following table:

Attribute	Description	Required
sourceDir	The absolute or relative top level directory used by JDeveloper to house the ESB project, must point to a directory that both exists and can be read, and must contain at least one ESB artifact.	Yes
deploymentPlanFile	The absolute or relative file name of the file that is to be created as a result of this task and must point to a file that can be written.	Yes

Parameter Specified as Nested Elements

The `extractESBDeploymentPlan` custom task supports no nested elements.

System Response

The custom Ant task will intimate you with the semantic validation errors in the case where the task has not been defined properly, as well as the status of its ability to create the deployment plan. On success, you will be informed of the full path of the created deployment plan file.

Examples

Example 13–5 describes the task to extract the deployment plan information that is currently held in the ESB project artifacts found in the `C:/Oracle/product/JDeveloper/v10.1.3.1/jdev/mywork/ESBSamples/CustomerData8` directory, into the file identified as `../CustomerData8DeploymentPlan.xml`.

Example 13–5 Example for extractESBDeploymentPlan

```
<target name="test.ExtractESBDeploymentPlan">
    <extractESBDeploymentPlan
sourceDir="C:/Oracle/product/JDeveloper/v10.1.3.1/jdev/mywork/ESBSamples/CustomerD
ata8"
        deploymentPlanFile="../CustomerData8DeploymentPlan.xml"/>
</target>
```

Example 13–6 is identical to the previous with the exception that the deployment plan will be deleted off the disk before the task is executed.

Example 13–6 Example for extractESBDeploymentPlan

```
<target name="test.ExtractESBDeploymentPlan">
    <property name="deploymentPlan" value="../CustomerData8DeploymentPlan.xml"/>
    <delete file="${deploymentPlan}"/>
    <extractESBDeploymentPlan
sourceDir="C:/Oracle/product/JDeveloper/v10.1.3.1/jdev/mywork/ESBSamples/
CustomerData8" deploymentPlanFile="${deploymentPlan}"/>
</target>
```

deployESBSuitcase

`deployESBSuitcase` is a custom Ant task that can be used to deploy one ESB project that has been created and is otherwise maintained by JDeveloper, after the contents of the designated deployment plan have been applied. The task provides identical behavior to that of JDeveloper's **Register with ESB** function, with the following differences:

- Application of the deployment plan to the found ESB artifacts is not done in JDeveloper's **Register with ESB** function, but is done with this task.
- Automatic refresh/resync of the JDeveloper artifacts is done in JDeveloper's **Register with ESB** function, but is not done with this task.

If this task is used to deploy one to many ESB projects and if JDeveloper is later desired to be used to manually deploy one or more of those given ESB projects using the **Register with ESB** function, then the ESB project metadata will first need to be refreshed from the ESB metadata server.

Parameters

The attribute and description of the parameters are described in the following table:

Attribute	Description	Required
sourceDir	This is the top level directory that houses all of the ESB project-related artifacts maintained by JDeveloper, that is to say, all .esb, .esbgrp, .esbsvc, .esbsys, .xml, .xsd, .xsl, .xslt, and .wsdl files maintained as part of the project	Yes
deploymentPlanFile	This indicates a file that houses the deployment plan that exists and can be read.	Yes
esbMetadataServerHostname	The host name of the computer that is supporting the ESB Metadata Server of interest.	Yes
esbMetadataServerPort	The port number of the ESB Metadata Server.	Yes
forcedDeployment	Whether or not to force the deployment and ignore conflicts or version and/or time stamp mismatches. This attribute has a default value of true. False is also accepted.	No

Note: The `esbMetadataServerPort` for a given ESB metadata server instance can be identified by the port number used to interact with the ESB control/console for the given machine. The URL used to interact with the ESB control/console for the desired ESB instance will use that same port number.

Parameter Specified as Nested Elements

The `deployESBSuitcase` custom task supports no nested elements.

System Response

The custom Ant task will intimate you with the semantic validation errors in the case where the task has not been defined properly, as well as output informing you of its progress. On completion of the task, open the ESB console of the appropriate ESB metadata server to verify that the ESB systems, service groups, and services have been

appropriately deployed. Additionally, you can inspect the aspects of each deployed entity to verify that the properties held in the specified deployment plan were those that were used at deployment time and are what the metadata server is in fact using.

In case the conflicts do exist between the artifacts held by the ESB metadata server and those hold in the JDeveloper-owned and managed project directory, you will see output from the Ant target that leverages this task that is similar to the following:

```
[deployESBSuitcase] Deployment Attempt Response :
[deployESBSuitcase] <deploymentSummary>
[deployESBSuitcase]    <message>Entity Deployment Failed</message>
[deployESBSuitcase]    <conflicts>
[deployESBSuitcase]       <conflict qname="AdapterServices"
 guid="A3CC9DE1AEF811DBBFF7559FF9492F22">
[deployESBSuitcase]          <lastModifiedTime>Tue Feb 27 15:59:03 MST
 2007</lastModifiedTime>
[deployESBSuitcase]          <lastModifiedUser>JDev_temp_ESBSamples
_AdapterServices_localhost</lastModifiedUser>
[deployESBSuitcase]       </conflict>
[deployESBSuitcase]       <conflict qname="AdapterServices.databaseAdapter"
 guid="CC826930AEF911DBBFF7559FF9492F22">
[deployESBSuitcase]          <lastModifiedTime>Tue Feb 27 15:59:03 MST
 2007</lastModifiedTime>
[deployESBSuitcase]          <lastModifiedUser>JDev_temp_ESBSamples
_AdapterServices_localhost</lastModifiedUser>
[deployESBSuitcase]       </conflict>
[deployESBSuitcase]    </conflicts>
[deployESBSuitcase] </deploymentSummary>
```

In this case, you can opt to use forced deployment, by specifying the `forcedDeployment` attribute to have a value of `true`, in which case this task should then succeed on the subsequent attempt.

Examples

Example 13–7 describes the custom Ant task to read the ESB project and all of its artifacts from the `C:/Oracle/product/JDeveloper/v10.1.3.1/jdev/mywork/ESBSamples/CustomerData8` directory and all of its subdirectories, apply the deployment plan found in `../CustomerData8DeploymentPlan.xml` to those artifacts (without changing the artifacts), and deploying the resulting artifacts to the ESB Metadata Server that is running on the localhost machine that is listening on port 8889. The resulting deployed artifacts are not retained as they will be regenerated each time the task is executed. Because the `forcedDeployment` attribute is given a value of `false`, if any conflicts are detected at deployment time, then the deployment attempt will be aborted.

Example 13–7 Example for deployESBSuitcase

```
<target name="test.DeployESBSuitcase">
    <deployESBSuitcase
sourceDir="C:/Oracle/product/JDeveloper/v10.1.3.1/jdev/mywork/ESBSamples/CustomerD
ata8"
                       deploymentPlanFile="../CustomerData8DeploymentPlan.xml"
                       esbMetadataServerHostname="localhost"
                       esbMetadataServerPort="8889"
                       forcedDeployment="false"/>
</target>
```

A

XPath Extension Functions

Oracle provides XPath extension functions that use built-in ESB capabilities and XPath standards. This chapter describes the XPath extension functions, along with their descriptions, signature, argument descriptions, and property ID information

This appendix contains the following topics:

- add-dayTimeDuration-to-dateTime on page A-2
- compare on page A-2
- compare-ignore-case on page A-3
- create-delimited-string on page A-3
- current-date on page A-4
- current-dateTime on page A-4
- current-time on page A-4
- day-from-dateTime on page A-5
- doc on page A-5
- ends-with on page A-5
- format-dateTime on page A-6
- format-string on page A-6
- generate-guid on page A-6
- get-content-as-string on page A-7
- get-localized-string on page A-7
- getInboundResponseHeader on page A-8
- getRequestHeader on page A-8
- hours-from-dateTime on page A-8
- implicit-timezone on page A-9
- index-within-string on page A-9
- last-index-within-string on page A-9
- left-trim on page A-10
- lookup-dvm on page A-10
- lookup-table on page A-11
- lookup-xml on page A-12

- lower-case on page A-12
- minutes-from-dateTime on page A-12
- month-from-dateTime on page A-13
- query-database on page A-13
- right-trim on page A-14
- seconds-from-dateTime on page A-14
- sequence-next-val on page A-14
- setOutboundHeader on page A-15
- setResponseHeader on page A-15
- square-root on page A-16
- subtract-dayTimeDuration-from-dateTime on page A-16
- timezone-from-dateTime on page A-16
- upper-case on page A-17
- year-from-dateTime on page A-17

add-dayTimeDuration-to-dateTime

This function returns a new date time value adding `dateTime` to the given duration.

If the duration value is negative, then the resulting value precedes `dateTime`.

Signature:

```
xp20:add-dayTimeDuration-from-dateTime(dateTime as string,
duration as string)
```

Arguments:

- `dateTime as string` – The `dateTime` to which the function adds the duration, in string format.
- `duration as string` – The duration to add to the `dateTime`, or subtract if the duration is negative, in string format.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

compare

This function returns the lexicographical difference between `inputString` and `compareString` comparing the unicode value of each character of both the strings.

This function returns `-1` if `inputString` lexicographically precedes the `compareString`.

This function returns `0` if both `inputString` and `compareString` are equal.

This function returns `1` if `inputString` lexicographically follows the `compareString`.

Example:

`xp20:compare('Audi', 'BMW')` returns `-1`

Signature:

`xp20:compare(inputString as string, compareString as string)`

Arguments:

- `variableName` – The source variable for the data
- `propertyName` – The qualified name (QName) of the property

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

compare-ignore-case

This function returns the lexicographical difference between `inputString` and `compareString` while ignoring case and comparing the unicode value of each character of both the strings.

This function returns `-1` if `inputString` lexicographically precedes the `compareString`.

This function returns `0` if both `inputString` and `compareString` are equal.

This function returns `1` if `inputString` lexicographically follows the `compareString`.

Example:

`xp20:compare-ignore-case('Audi','bmw')` returns `-1`

Signature:

`orcl:compare-ignore-case(inputString as string, compareString as string)`

Arguments:

- `inputString as string` – The input string
- CompareString as string – The string to compare against the input string

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

create-delimited-string

This function returns a delimited string created from `nodeSet` delimited by delimiter.

Signature:

`orcl:create-delimited-string(nodeSet as node-set, delimiter as string)`

Arguments:

- `nodeSet` – The node set to be converted into a deliminated string
- `delimiter` – The character that separates the items in the output string; for example, a comma or a semicolon.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: orcl`

current-date

This function returns the current date in ISO format `YYYY-MM-DD`.

Signature:

```
xp20:current-date(object)
```

Arguments:

- `object` - The time in standard format

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix: xp20`

current-dateTime

This function returns the current datetime-value in ISO format `CCYY-MM-DDThh:mm:ssTZD`.

Signature:

```
xp20:current-dateTime(object)
```

Arguments:

- `object` – The time in standard format

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix: xp20`

current-time

This function returns the current time in ISO format. The format is `hh:mm:ssTZD`.

Signature:

```
xp20:current-time(object)
```

Arguments:

- `object` – The time in standard format

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

day-from-dateTime

This function returns the day from `dateTime`. The default day is `1`.

Signature:

```
xp20:day-from-dateTime(object)
```

Arguments:

- `object` – The time in standard format

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

doc

This function returns the content of an XML file.

Signature:

```
ora:doc('fileName','xpath'?)
```

Arguments:

- `fileName` – The name of the XML file
- `xpath` – path to the XML file

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

ends-with

This function returns true if `inputString` ends with `searchString`.

Example:

`xp20:ends-with('XSL Map','Map')` returns `true`

Signature:

```
xp20:ends-with(inputString as string, searchString as string)
```

Arguments:

- `inputString` – The string of data to be searched
- `searchString` – The string for which the function searches

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

format-dateTime

This function returns the formatted string of `dateTime` using the format provided.

Signature:

```
xp20:format-dateTime(dateTime as string, format as string)
```

Arguments:

- `dateTime` – The `dateTime` to be formatted
- `format` – The format for the output

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

format-string

This function returns the message formatted with the arguments passed. At least one argument is required and supports up to a maximum of 10 arguments.

Example:

`orcl:format-string('{0} + {1} = {2}','2','2','4')` returns `'2 + 2 = 4'`

Signature:

```
orcl:format-string(string,string,string...)
```

Arguments:

- `string` – One of the strings to be used in the formatted output

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

generate-guid

This function generates a unique GUID.

Signature:

```
orcl:generate-guid()
```

Arguments:

- none

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: orcl`

get-content-as-string

This function returns the XML representation of the input element.

Signature:

```
orcl:get-content-as-string(element as node-set)
```

Arguments:

- `element as node-set` – The input element that the function returns as an XML representation

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: orcl`

get-localized-string

This function returns the locale-specific string for key. This function uses language, country, variant, and resource bundle to identify the correct resource bundle.

The resource bundle in obtained by resolving `resourceLocation` against the `resourceBaseURL`. The URL is assumed to be a directory only if it ends with `/`.

Usage:

```
orcl:get-localized-string(resourceBaseURL as string,
resourceLocation as string, resource bundle as string, language
as string, country as string, variant as string, key as string)
```

Example:
`orcl:get-localized-string('file:/c:/','','MyResourceBundle','en','US','','MSG_KEY')` returns a locale-specific string from a resource bundle `'MyResourceBundle'` in the `C:\` directory

Signature:

```
orcl:get-localized-string(resourceURL,resourceLocation,resourceBundleName,language,country,variant,messageKey)
```

Arguments:

- `resourceURL` – The URL of the resource
- `resourceLocation` – The subdirectory location of the resource
- `resourceBundleName` – The name of the zip file containing the resource bundle
- `language` – The language of the localized output
- `country` – The country of the localized output

- `variant` – The language variant of the localized output
- `messageKey` – The message key in the resource bundle

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: orcl`

getInboundResponseHeader

This function gets a string value from inbound response header.

Signature:

```
ehdr:getInboundResponseHeader(headerXPath,
namespaceDeclarations)
```

Arguments:

- `headerXPath` is the xpath into the header document
- `namespaceDeclarations` used in the headerXPath as `prefix=Namespace-URI`

Property IDs:

- `http://www.oracle.com/XSL/Transform/java/oracle.tip.esb.server.headers.ESBHeaderFunctions`
- `namespace-prefix: ehdr`

getRequestHeader

This function gets a string value from request message header.

Signature:

```
ehdr:getRequestHeader(headerXPath, namespaceDeclarations)
```

Arguments:

- `headerXPath` is the xpath into the header document
- `namespaceDeclarations` used in the headerXPath as `prefix=Namespace-URI`

Property IDs:

- `http://www.oracle.com/XSL/Transform/java/oracle.tip.esb.server.headers.ESBHeaderFunctions`
- `namespace-prefix: ehdr`

hours-from-dateTime

This function returns the hour from `dateTime`. The default hour is `0`.

Signature:

```
xp20:hours-from-dateTime(dateTime as string)
```

Arguments:

- `dateTime as string` – The `dateTime`

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

implicit-timezone

This function returns the current time zone in ISO format `+/- hh:mm`, indicating a deviation from UTC (Coordinated Universal Timezone).

Signature:

`xp20:implicit-timezone(object)`

Arguments:

- `object` – The time in standard format

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

index-within-string

This function returns the zero-based index of the first occurrence of `searchString` within the `inputString`.

This function returns `-1` if `searchString` is not found.

Example:

`orcl:index-within-string('ABCABC, 'B')` returns `1`

Signature:

`orcl:index-within-string(inputString as string, searchString as string)`

Arguments:

- `inputString` – The string to be searched
- searchString – The string for which the function searches in the `inputString`

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

last-index-within-string

This function returns the zero-based index of the last occurrence of `searchString` within `inputString`.

This function returns `-1` if `searchString` is not found.

Example:

`orcl:last-index-within-string('ABCABC', 'B')` returns `4`

Signature:

```
orcl:last-index-within-string(inputString as string,
searchString as string)
```

Arguments:

- `inputString` – The string to be searched
- searchString – The string for which the function searches in the `inputString`

Property IDs:

- `namespace-uri:` `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

left-trim

This function returns the value of `inputString` after removing all the leading white spaces.

Example:

`orcl:left-trim('  account  ') returns 'account  '`

Signature:

`orcl:left-trim(inputString)`

Arguments:

- `inputString` – The string to be left-trimmed

Property IDs:

- `namespace-uri:` `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

lookup-dvm

This function returns a string by looking up the value for target column in the DVM where the value for source column is equal to the source value. The source value is an xpath expression bound to the source document of the XSLT transformation. The expression will be evaluated during the transformation and the result value will be passed as the source value for lookup. This function takes 5 parameters.

Example:

```
orcl:lookup-dvm
('StateCodes','Long',/inp1:Customer/Profile/AccountRep/Address/State,'Short',"")
```

Signature:

```
orcl:lookup-dvm (dvmName as string, sourceColumn as string,
sourceValue as xpath expression, targetColumnName as string,
defaultValue as string)
```

Arguments:

- dvmName
- sourceColumnName
- `sourceValue`
- targetColumnName
- `defaultValue`

Property IDs:

- `namespace-uri:` `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

lookup-table

This function returns a string based on the SQL query generated from the parameters.

The string is obtained by executing:

```
SELECT outputColumn FROM table WHERE inputColumn = key
```

against the data source that can be either a JDBC connect string (`jdbc:oracle:thin:`*`username`*`/`*`password`*`@`*`host`*`:`*`port`*`:`*`sid`*) or a data source JNDI identifier. Only Oracle Thin Driver is supported if the JDBC connect string is used.

Example:

```
orcl:lookup-table('employee','id','1234','last_
name','jdbc:oracle:thin:scott/tiger@localhost:1521:ORCL')
```

Signature:

```
orcl:lookup-table(table, inputColumn, key, outputColumn,
datasource)
```

Arguments:

- `table` – The table from which to draw the data
- `inputColumn` – The column within the table
- `key` – The key
- `outputColumn` – The column to output the data
- `datasource` – The source of the data

Property IDs:

- `namespace-uri:` `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

lookup-xml

This function returns the string value of an element defined by `lookupXPath` in an XML file (`docURL`) given its parent XPath (`parentXPath`), the key XPath (`keyXPath`), and the value of the key (`key`).

Example:

`orcl:lookup-xml('file:/d:/country_data.xml', '/Countries/Country', 'Abbreviation', 'FullName', 'UK')` returns the value of the element `FullName` child of `/Countries/Country` where `Abbreviation = 'UK'` is in the file `D:\country_data.xml`.

Signature:

`orcl:lookup-xml(docURL, parentXPath, keyXPath, lookupXPath, key)`

Arguments:

- `docURL` – The XML file
- `parentXPath` – The parent XPath
- `keyXPath` – The key XPath
- `lookupXPath` – The lookup XPath
- `key` – The key value

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: orcl`

lower-case

This function returns the value of `inputString` after translating every character to its lower-case correspondent.

Example:

`xp20:lower-case('ABc!D')` returns `'abc!d'`

Signature:

`xp20:lower-case(inputString)`

Arguments:

- `inputString` – The input string

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix: xp20`

minutes-from-dateTime

This function returns the minute from `dateTime`. The default minute is `0`.

Signature:

`xp20:minutes-from-dateTime(dateTime)`

Arguments:

- `dateTime` – The `dateTime`

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

month-from-dateTime

This function returns the month from `dateTime`. The default month is `1` (January).

Signature:

`xp20:month-from-dateTime(dateTime)`

Arguments:

- `dateTime` – The `dateTime` to be formatted

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

query-database

This function returns a node-set by executing the SQL query against the specified database.

Signature:

`orcl:query-database(sqlquery as string, rowset as boolean, row as boolean, datasource as string)`

Arguments:

- `sqlquery` – The SQL query to perform
- `rowset` – Indicates if the rows should be enclosed in a `<rowset>` element
- `row` – Indicates if each row should be enclosed in a `<row>` element
- `datasource` – Either a JDBC connect string (`jdbc:oracle:thin:`*`username/password@host:port:sid`*) or a JNDI name for the database

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

right-trim

This function returns the value `inputString` after removing all the trailing white spaces.

Example:

`orcl:right-trim('  account  ')` returns `'  account'`

Signature:

```
orcl:right-trim(inputString as string)
```

Arguments:

- `inputString` – The input string to be right-trimmed

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: orcl`

seconds-from-dateTime

This function returns the second from `dateTime`. The default second is `0`.

Signature:

```
xp20:seconds-from-dateTime(dateTime as string)
```

Arguments:

- `dateTime` – The `dateTime` as a string

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix: xp20`

sequence-next-val

Return the next value of an Oracle sequence.

The next value is obtained by executing

```
SELECT sequence.nextval FROM dual
```

against data source that can be either a JDBC connect string (`jdbc:oracle:thin:`*`username`*`/`*`password`*`@`*`host`*`:`*`port`*`:`*`sid`*) or a data source JNDI identifier. Only Oracle Thin Driver is supported if a JDBC connect string is used.

Example:

```
orcl:sequence-next-val('employee_id_
sequence','jdbc:oracle:thin:scott/tiger@localhost:1521:ORCL')
```

Signature:

```
orcl:sequence-next-val(sequence as string, datasource as string)
```

Arguments:

- `sequence` – name of the sequence in the database
- `datasource` – a JDBC connect string or data source JNDI identifier

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.`
- `namespace-prefix: orcl`

setOutboundHeader

This function sets an outbound message header value.

Signature:

```
ehdr:set-outbound-header(headerXPath, value,
namespaceDeclarations)
```

Arguments:

- `headerXPath` is the xpath into the outbound header document
- `value` is the value to set on outbound message header
- `namespaceDeclarations` used in the headerXPath as `prefix=Namespace-URI`

Property IDs:

- `http://www.oracle.com/XSL/Transform/java/oracle.tip.esb.server.headers.ESBHeaderFunctions`
- `namespace-prefix: ehdr`

setResponseHeader

This function sets a response message header value.

Signature:

```
ehdr:set-response-header(headerXPath, value,
namespaceDeclarations)
```

Arguments:

- `headerXPath` is the xpath into the response header document
- `value` is the value to set on response message header
- `namespaceDeclarations` used in the headerXPath as `prefix=Namespace-URI`

Property IDs:

- `http://www.oracle.com/XSL/Transform/java/oracle.tip.esb.server.headers.ESBHeaderFunctions`
- `namespace-prefix: ehdr`

square-root

This function returns the square root of `inputNumber`.

Example:

`orcl:square-root(25)` returns `5`

Signature:

```
orcl:square-root(inputNumber as number)
```

Arguments:

- `inputNumber` – The input number for which the function calculates the square root

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: orcl`

subtract-dayTimeDuration-from-dateTime

This function returns a new `dateTime` value after subtracting duration from `dateTime`.

If the duration value is negative, then the resultant `dateTime` value follows `input-dateTime` value.

Signature:

```
xp20:subtract-dayTimeDuration-from-dateTime(dateTime as string,
duration as string)
```

Arguments:

- `dateTime as string` – The `dateTime` from which the function subtracts the duration, in string format.
- `duration as string` – The duration to subtract to the `dateTime`, or add if the duration is negative, in string format.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix: xp20`

timezone-from-dateTime

This function returns the timezone from `dateTime`. The default timezone is `GMT+00:00`.

Signature:

```
xp20:timezone-from-dateTime(dateTime as string)
```

Arguments:

- `dateTime as string` – The `dateTime` for which this function returns a time zone

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

upper-case

This function returns the value of `inputString` after translating every character to its upper-case correspondent.

Example:

`xp20:upper-case('abCd0')` returns `'ABCD0'`

Signature:

`xp20:upper-case(inputString as string)`

Arguments:

- `inputString` – The input string

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

year-from-dateTime

This function returns the year from `dateTime`.

Signature:

`xp20:year-from-dateTime(dateTime as string)`

Arguments:

- `dateTime` – The `dateTime`

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

B

Oracle Enterprise Service Bus API

This appendix describes the classes and interfaces that can be used with Oracle ESB Control. It contains the following topics:

- ConsoleClientFactory Class on page B-1
- ConsoleClient Interface on page B-1
- XML Schema File on page B-8

ConsoleClientFactory Class

Provides the ConsoleClient object.

```
package oracle.tip.esb.client;
import oracle.tip.esb.client.impl.ConsoleClientImpl;
public class ConsoleClientFactory
  {
     public static ConsoleClient getConsoleClient(String host, int port, String
     userName, String password) throws ClientException
      {
        return new ConsoleClientImpl(host,port,userName,password);
      }
  }
```

ConsoleClient Interface

Provides the interface to interact with the Oracle ESB Control to perform actions on the metadata and instance data.

```
package oracle.tip.esb.client;
import java.util.Map;
public interface ConsoleClient
{
public String perform(String action, Map<String,String> requestParameters) throws
ClientException;
}
```

Perform Function

The `perform` function of the `ConsoleClient` interface invokes the console servlet to perform a specified action. The `perform` function returns the data produced by an action. If the action specified in the action parameter does not generate any data, then the `perform` function throws `ClientException` exception. The `action` parameter can contain one of the following values:

- GetInstances
- GetFailedInstances
- GetErrorInstance
- GetTrackingDetails
- ResubmitInstancesByIds
- ResubmitInstancesById
- ResubmitInstance
- GetTrackingConfig
- UpdateTrackingConfig

Using Perform Function

You can use the perform function to perform the following tasks:

- Get the list of instances
- Get the list of errored instances
- Get all the errors occurred in a flow id
- Get the XML to draw the instance diagram for a flow id
- Resubmit a list of instances by Ids
- Resubmit an instance by Id
- Resubmit an instance with modified/unmodified payload
- Get the current status of the instance tracking
- Enable/disable instance tracking

Get the list of instances

To get a list of all instances, specify GetInstances as value of action parameter. The perform function returns a list of instances matching the given filter. If an error occurs, an exception is returned to the caller.

Input XSD Element

The input XML should correspond to the instanceFilter element of the XSD file described in "XML Schema File" on page B-8.

Output XSD

The output XML corresponds to the instances element of the XSD file described in "XML Schema File" on page B-8.

Example

The following example returns all the instances created in last one day.

```
String filter = "<instanceFilter timeZone=\"GMT+05:30\">" +
        "   <startTime>86400000</startTime>" +
        "</instanceFilter>";

HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("filter", URLEncoder.encode(filter,"UTF-8"));
```

```
ConsoleClient client = ConsoleClientFactory.
        getConsoleClient(HOST,PORT,USER_NAME,PASSWORD);
String data = client.perform("GetInstances", requestProps);
System.out.println("Instances ==> "+data);
```

Get the list of errored instances

To get a list of errored instances, specify `GetFailedInstances` as value of `action` parameter. The `perform` function returns a list of errored instances matching the given filter. If an error occurs, an exception is returned to the caller.

Input XSD Element

The input XML should correspond to the `failedInstanceFilter` element of the XSD file described in "XML Schema File" on page B-8.

Output XSD

The output XML corresponds to the `failedInstances` element of the XSD file described in "XML Schema File" on page B-8.

Example

The following example returns all the errored (Retryable and non-retryable) instances created in last one day:

```
String filter = "<failedInstanceFilter timeZone=\"GMT+05:30\">" +
        "   <startTime>86400000</startTime>" +
        "</failedIinstanceFilter>";

HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("filter", URLEncoder.encode(filter,"UTF-8"));

ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);

String data = client.perform("GetFailedInstances", requestProps);

System.out.println("Failed Instances ==> "+data);
```

Get all the errors occurred in a flow id

To get a list of all errors occurred in a flow id, specify `GetErrorInstance` as value of `action` parameter. The `perform` function returns all the errors that occurred in the given flow id. If the error can be retried, the corresponding error element contains the list of routing rules that would be executed upon resubmission and the payload that the user can modify before resubmission. If an error occurs, an exception is returned to the caller.

Output XSD

The output XML corresponds to the `errorInstances` element of the XSD file described in "XML Schema File" on page B-8.

Example

The following example returns all the errored instances for a given instance id:

```
HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("instanceID",URLEncoder.encode("102@1164094022","UTF-8"));
```

```
ConsoleClient client = ConsoleClientFactory.
getConsoleClient(HOST,PORT,USER_NAME,PASSWORD);

String data = client.perform("GetErrorInstance", requestProps);

System.out.println("Error Instances ==> "+data);
```

Get the XML to draw the instance diagram for a flow id

To get XML containing the flow details, specify `GetTrackingDetails` as value of `action` parameter. The `perform` function returns the XML containing the flow details. You can use this XML is used to draw the instance diagram on the console. If an error occurs, an exception is returned to the caller.

Output XSD

The output XML corresponds to the `relationship` element of the XSD file described in "XML Schema File" on page B-8.

Example

The following example returns the XML required to draw the instance diagram for the given flow id.

```
HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("flowId", URLEncoder.encode("102@1164094022","UTF-8"));

ConsoleClient client = ConsoleClientFactory.
getConsoleClient(HOST,PORT,USER_NAME,PASSWORD);

String data = client.perform("GetTrackingDetails", requestProps);
System.out.println("Instance Diagram XML ==> "+data);
```

Resubmit a list of instances by Ids

To resubmit multiple instances that failed in multiple systems and flows, specify `ResubmitInstancesByIds` as value of `action` parameter. The input for this action is a list of ids. A unique instance id is created by using the `FlowId` and `SystemId`. During resubmit, the console back end retrieves the actual payloads for these ids from instance store.

Input XSD Element

The input XML should correspond to the `resubmitInstanceIds` element of the XSD file described in "XML Schema File" on page B-8.

Output XSD

The output XML corresponds to the `resubmissionFailureReport` element of the XSD file described in "XML Schema File" on page B-8.

Example

```
HashMap<String,String> requestProps = new HashMap<String,String>();
String ids = "<resubmitInstanceIds>" +
             "<resubmitInstanceId flowId=\"102@1164094022\"
systemId=\"96DD76C0971311DABF1A87858E4395A7\"/>" +
         "<resubmitInstanceId flowId=\"101@1164094022\"
systemId=\"96DD76C0971311DABF1A87858E4395A7\"/>" +
```

```
        "</resubmitInstanceIds"; requestProps.put("resubmitIds",
URLEncoder.encode(ids,"UTF-8"));

ConsoleClient client = ConsoleClientFactory.
getConsoleClient(HOST,PORT,USER_NAME,PASSWORD);

String data = client.perform("ResubmitInstancesByIds",requestProps);

System.out.println("Resubmit Instances By Ids Status ==> "+data);
```

Resubmit an instance by Id

To resubmit multiple instances that failed in a system, specify `ResubmitInstanceById` as value of `action` parameter. The input for this action is an ID. A unique instance id is created by using the `FlowId` and `SystemId`. During resubmit, the console back end retrieves the actual payload for the given id from instance store.

Example

```
HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("flowId", flowId);
requestProps.put("systemId", systemId);
ConsoleClient client = ConsoleClientFactory.
                getConsoleClient(HOST,PORT,USER_NAME,PASSWORD);
client.perform("ResubmitInstanceById",requestProps);
```

Resubmit an instance with modified/unmodified payload

To resubmit a single instance that failed in a system and flow, specify `ResubmitInstance` as value of `action` parameter. The input for this action is an XML document containing the modified or unmodified payload.

The actions mentioned in "Get the list of instances" on page B-2 and "Get the list of errored instances" on page B-3 can be used to obtain the list of errored instances with payload. You need to call these actions in a loop to submit each error instance.

Input XSD Element

The input XML should correspond to the `messageInstance` element of the XSD file described in "XML Schema File" on page B-8.

The following example shows a sample XML file based on the `messageInstance` element.

```
String resubInstance = "<messageInstance  flowId="eJy_IuGNrBcA2cdYouGUGw.."
operationGUID="776C7C10EF1D11DBBFB64F031236B6A9">
  <instanceLink subFlowId="96DD76C0971311DABF1A87858E4395A7~1" />
  <inPayload>
     <![CDATA[<PurchaseOrder xmlns="http://www.globalcompany.com/ns/sales">
                 <CustID>Fonda Automobiles</CustID>
                 <ID>4P-91S3</ID>
                 <ShipTo>
                   <Name>
                     <First>Prashant</First>
                       <Last>Nema</Last>
                   </Name>
                   <Address>
                     <Street>123, Sultod Ave</Street>
                     <City>Vasetras</City>
```

```
                        <State>Sweden</State>
                        <Zip>SW-30845</Zip>
                        <Country>Sweden</Country>
                      </Address>
                    </ShipTo>
                    <BillTo>
                      <Name>
                        <First>Matti</First>
                        <Last>Korkalop</Last>
                      </Name>
                      <Address>
                        <Street>A5G, TagenStrasse</Street>
                        <City>Ausburg</City>
                        <State>Bravaria</State>
                        <Zip>DE-15982</Zip>
                        <Country>Germany</Country>
                      </Address>
                    </BillTo>
                    <UserContact>
                       <PhoneNumber>01-41-294-2938</PhoneNumber>
                       <EmailAddress>mattik@automart.de</EmailAddress>
                    </UserContact>
                    <OrderItems>
                       <Item>
                         <ProductName>Front Glass Panel</ProductName>
                         <itemType>Automobile</itemType>
                         <partnum>F39-FR3</partnum>
                         <price>400.00</price>
                         <Quantity>1</Quantity>
                       </Item>
                       <Item>
                         <ProductName>Al Wheel Hub</ProductName>
                         <itemType>Assembly</itemType>
                         <partnum>45-JRP4</partnum>
                         <price>900.00</price>
                         <Quantity>3</Quantity>
                       </Item>
                    </OrderItems>
                    <OrderDate>2003-11-12</OrderDate>
                    <OrderPrice>5100.00</OrderPrice>
                    <OrderStatus>unknown</OrderStatus>
             </PurchaseOrder>
                ]]>
    </inPayload>
</messageInstance>";
```

Example

```
HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("resub", URLEncoder.encode(resubInstance,"UTF-8"));
ConsoleClient client = ConsoleClientFactory.
getConsoleClient(HOST,PORT,USER_NAME,PASSWORD);

client.perform("ResubmitInstance",requestProps);
```

Get the current status of the instance tracking

To know if instance tracking is enabled or not, specify `GetTrackingConfig` as value of `action` parameter. The `perform` function returns `true` if the instance tracking is enabled else returns `false`.

Output XSD

The output XML corresponds to the `instanceManage` element of the XSD file described in "XML Schema File" on page B-8.

Example

The following example returns the current status of the instance tracking:

```
ConsoleClient client = ConsoleClientFactory.
getConsoleClient(HOST,PORT,USER_NAME,PASSWORD);

client.perform("GetTrackingConfig",null);
```

Enable/disable instance tracking

To enables or disable the instance tracking, specify `UpdateTrackingConfig` as value of `action` parameter.

Input XSD Element

The input XML should correspond to the `instanceManage` element of the XSD file described in "XML Schema File" on page B-8.

Example

The following example disables the instance tracking:

```
String config = "<instanceManage enable=\"false\" />"
HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("root", config);
ConsoleClient client = ConsoleClientFactory.
getConsoleClient(HOST,PORT,USER_NAME,PASSWORD);
client.perform("UpdateTrackingConfig", requestProps);
```

Purge instances based on time

To purge instances based on time, specify `UpdateTrackingConfig` as value of `action` parameter. You can purge instances based on the following criteria: `all` or `Older than <TIME>`.

Input XSD Element

The input XML should correspond to the `purgeInstance` element of the XSD file described in "XML Schema File" on page B-8.

Example

The following example deletes all the instances that are older than ten minutes:

```
String purge = "<purgeInstance userPurgeTimePeriod=\"600000\"/>";
 HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("root", purge);
ConsoleClient client = ConsoleClientFactory.
getConsoleClient(HOST,PORT,USER_NAME,PASSWORD);
client.perform("UpdateTrackingConfig", requestProps);
```

XML Schema File

The XML Schema file for instance tracking is defined in the following example.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema elementFormDefault="qualified"
 targetNamespace="http://www.oracle.com/esb"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esb="http://www.oracle.com/esb" xmlns="http://www.oracle.com/esb">
<!-- Elements for relationship diagram -->

  <xsd:element name="relationship">
    <xsd:annotation>
      <xsd:documentation>
        Provides the data required to render the relationship diagram
        on the console
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="service">
            <xsd:complexContent>
              <xsd:extension base="RelationshipService">
                <xsd:sequence>
                  <xsd:element name="linkedServices" minOccurs="0">
                    <xsd:complexType>
                      <xsd:sequence>
                        <xsd:element name="service"
                                     type="RelationshipService"
                                     maxOccurs="unbounded"/>
                      </xsd:sequence>
                    </xsd:complexType>
                  </xsd:element>
                </xsd:sequence>
              </xsd:extension>
            </xsd:complexContent>
          </xsd:element>
          <xsd:element name="serviceLinks" minOccurs="0">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="serviceLink" maxOccurs="unbounded">
                  <xsd:complexType>
                    <xsd:sequence>
                      <xsd:element name="operationLink"
                                   maxOccurs="unbounded">
                        <xsd:complexType>
                          <xsd:sequence>
                            <xsd:element ref="property"
                                         maxOccurs="unbounded">
                              <xsd:annotation>
                                <xsd:documentation>
                                  Will have one of the following
                                  properties
                                    executionType - Immediate |
                                                    Deferred
                                    subscriptionType - Local | Remote
                                    phase - The phase value
                                    filter - Subscription filter
```

```
                                        expression
                                mep - One-Way | RequestResponse |
                                      RequestResponseFault |
                                      SolicitResponse |
                                      Notification
                                transformFile - ESB URL to XSL file
                                replyTransformFile - ESB URL to XSL
                                                     file
                                faultTransformFile - ESB URL to XSL
                                                     file

                                replyHandler - Guid of the service,
                                               which the response
                                       is routed to
                                replyHandlerOperation -
                                    Name of the operation,
                            which the response is routed to
                                faultHandler -
                                    Guid of the service, which the
                                    fault is routed to
                                faultHandlerOperation -
                                    Name of the operation, which
                            the fault is routed to

                                Following properties are used ONLY
                                for Instance Diagram

                                status - Completed | Faulted |
                                         Error | In-Flight |
                                         Filtered | NotElgible |
                                         Committed | RolledBack
                                errorType - Transform |
                                            Reply_Transform |
                                            Fault_Transform
                                routedOutputTo - Source | Ignored |
                                                 Handler
                            </xsd:documentation>
                          </xsd:annotation>
                        </xsd:element>
                        <xsd:element name="replyHandlerLink"
                                     type="HandlerLink"
                                     minOccurs="0"/>
                        <xsd:element name="faultHandlerLink"
                                     type="HandlerLink"
                                     minOccurs="0"/>
                      </xsd:sequence>

                      <xsd:attributeGroup ref="RelationshipLink"/>
                    </xsd:complexType>
                  </xsd:element>
                </xsd:sequence>
                <xsd:attributeGroup ref="RelationshipLink"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:complexType name="HandlerLink">
  <xsd:sequence>
    <!-- Contains the following properties
    status - Completed | Error | In-Flight
    transformFile - XSL file
    -->
    <xsd:element ref="property" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:attribute name="serviceGUID" type="xsd:string"
                 use="required"/>
  <xsd:attribute name="operation" type="xsd:string" use="required"/>
</xsd:element>

<xsd:complexType name="RelationshipService">
  <xsd:sequence>
    <xsd:element ref="property" maxOccurs="unbounded">
      <xsd:annotation>
        <xsd:documentation>
          Contains the following properties:
            id - unique id
            system - Name of the system that this service belongs to
            status - ENABLED | DISABLED for Relationship diagram
                     Completed | Faulted | Error | In-Flight |
                     Committed | RolledBack for instance diagram
            depth - To be used to place the service icons at
                    appropriate places on the console

          BPEL process specific properties
            processDomain - Domain name
            processName - The name of the bpel process
            processVersion - version
            host - The host name where bpel service runs
            port - bpel service port
            instanceId - Bpel instance id

          Overlay properties
            latency - Average execution time in milliseconds taken
                      for the service to process messages
            throughput - Average number of messages processed by the
                         service in a minute
            instanceCount -  Total number of messages processed by
                             the service
            failureCount -  Total number of messages that the service
                            could not process sucessfully

            Above-mentioned overlay properties are ONLY for
            relationship diagram

            changeType - Modified | Deleted
            executionTime - Time taken in millieseconds to process a
                            message.
            lastActivityTime - Time when the last activity recorded
                               for service during a message processing

            Above-mentioned overlay properties are ONLY for instance
            diagram

          The overlay properties depend on the instance tracking
          data.
```

```
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <!-- Only for instance diagram -->
   <xsd:element name="trackableFields" type="TrackableFieldValueList"
                minOccurs="0"/>
  </xsd:sequence>
  <!-- contains guid, qname and name attribute -->
  <xsd:attributeGroup ref="IdentifierInfo"/>
  <xsd:attributeGroup ref="ServiceTypeInfo"/>
  <xsd:attribute name="version" type="xsd:long"/>
</xsd:complexType>
<xsd:attributeGroup id="RelationshipLink">
  <xsd:attribute name="source" type="xsd:string">
    <xsd:annotation>
      <xsd:documentation>
         Represents the name of the source that invokes the
         target. If the source service is Inbound Adapter Service,
         this attribute will not be present in the xml
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
  <xsd:attribute name="target" type="xsd:string" use="required">
    <xsd:annotation>
      <xsd:documentation>
         Represents the name of the target that gets invoked
        by the source
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
</xsd:attributeGroup>

<xsd:element name="property">
  <xsd:annotation>
    <xsd:documentation>
      Represents a name-value pair
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:restriction base="xsd:any">
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="value" type="xsd:string"
                       use="required"/>
      </xsd:restriction>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name="instanceStatusMessage" type="xsd:string"
             minOccurs="0">
  <xsd:annotation>
    <xsd:documentation>
      Holds a message providing more information about an activity
      based on the status.
      If the status is error, this element will have the error
      message.
      If the status is faulted, this element will have the fault
      message generated by the target serice.
    </xsd:documentation>
```

```
    </xsd:annotation>
  </xsd:element>


<!-- Elements and types for Instances-->

  <xsd:element name="instances">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="instance" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="instanceCount" type="xsd:string"/>
      <xsd:attribute name="availableMore" type="xsd:boolean"/>
      <xsd:attribute name="generatedTime" type="xsd:dateTime"/>
      <xsd:attribute name="generatedTimeString" type="xsd:string"/>
    </xsd:complexType>
   </xsd:element>
  <xsd:element name="instance">
    <xsd:complexType>
      <xsd:attribute name="flowId" type="xsd:string" use="required"/>
        <!-- Service GUID for Inbound adapter service
             SOAP
             Java -->
      <xsd:attribute name="initiatedBy" type="MessageSource"
                     use="required"/>
      <xsd:attribute name="initiatedAt" type="xsd:dateTime"
                     use="required"/>
      <xsd:attribute name="initiatedAtString" type="xsd:string"
                     use="required"/>
      <xsd:attribute name="status" type="InstanceStatus"/>
      <xsd:attribute name="type" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="MessageSource">
    <xsd:restriction base="xsd:TOKEN">
      <xsd:enumeration value="Service"/>
      <xsd:enumeration value="SOAP"/>
      <xsd:enumeration value="Java"/>
      <xsd:enumeration value="Unknown"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="InstanceStatus">
    <xsd:restriction base="xsd:TOKEN">
      <!-- <xsd:enumeration value="Completed"/> -->
      <xsd:enumeration value="Error"/>
      <xsd:enumeration value="Faulted"/>
      <!-- <xsd:enumeration value="In-Flight"/> -->
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:element name="instanceFilter">
    <xsd:complexType>
      <xsd:sequence>
        <!-- Flow id to uniquely identify a flow
        If flow id is present in the filter, rest of the filters will
        be ignored.
        -->
        <xsd:element name="flowId" type="xsd:string" minOccurs="0"/>
        <!-- Activity in last 'n' seconds | minutes | hours | days.
```

```
      The value should be converted to milliseconds
      -->
      <xsd:element name="startTime" type="xsd:long" minOccurs="0"/>
      <xsd:element name="serviceGUID" type="xsd:string"
                   minOccurs="0"/>
      <xsd:element name="status" type="InstanceStatus"
                   minOccurs="0"/>
      <xsd:element name="trackableFields"
                   type="TrackableFieldValueList"
        minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="timeZone" type="xsd:TOKEN" use="required"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="failedInstanceFilter">
  <xsd:complexType>
    <xsd:complexContent>
       <xsd:extension base="instanceFilter"/>
         <xsd:sequence>
           <xsd:element name="errorMessageContains" type="xsd:string"
                        minOccurs="0"/>
         </xsd:sequence>

         <!-- The starting number of the instance page -->
   <xsd:attribute name="pageStartsFrom" type="xsd:integer"
                        default="1"/>

       <!-- The size of the instance page.
        If omitted,  only 100 (that is default for instance
              search) instances would be retured with
             "availableMore" attribute set to true in
       "FailedInstances" output element -->
      <xsd:attribute name="pageSize" type="xsd:integer"/>

     <!-- Error message, Stacktrace and payload would be included
              only if this flag is set to true -->
         <xsd:attribute name="includeMoreDetails"
                        type="xsd:boolean"/>

      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
<xsd:complexType name="TrackableFieldValueList">
  <xsd:sequence>
    <xsd:element name="trackableField" maxOccurs="unbouunded">
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:extension base="xsd:string">
            <xsd:attribute name="name" type="xsd:string"
                           use="required"/>
            <xsd:attribute name="operator" type="xsd:string"/>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="operator">
      <xsd:simpleType>
```

```
        <xsd:restriction base="xsd:TOKEN">
          <xsd:enumeration value="AND"/>
          <xsd:enumeration value="OR"/>
        </xsd:restriction>
      </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="MessageInstanceType">
  <xsd:sequence>
    <xsd:element name="instanceLink" type="InstanceLinkType"
                 minOccurs="1" maxOccurs="unbounded"/>
    <!-- Request payload -->
    <xsd:element name="inPayload" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="flowId" type="xsd:string" use="required"/>
  <xsd:attribute name="batchId" type="xsd:string"/>
  <!-- QName of the service -->
  <xsd:attribute name="serviceQName" type="xsd:string"/>
  <xsd:attribute name="operationGUID" type="xsd:string"/>
  <!-- name of the operation -->
  <xsd:attribute name="operationQName" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="InstanceLinkType">
  <xsd:sequence>
    <xsd:element name="filterExpression" type="xsd:string"
                 minOccurs="0"/>
    <xsd:element name="xslFileURL" type="xsd:string"  minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="guid" type="xsd:string"/>
  <xsd:attribute name="subFlowId" type="xsd:string" use="required"/>
  <!-- QName of the target service -->
  <xsd:attribute name="serviceQName" type="xsd:string"
                 use="required"/>
  <!-- QName of the target operation -->
  <xsd:attribute name="operationQName" type="xsd:string"
                 use="required"/>
</xsd:complexType>

<xsd:element name="messageInstance" type="MessageInstanceType"/>

<xsd:element name="messageInstances"
  <xsd:complexType>
     <xsd:sequence>
         <xsd:element ref="messageInstance" maxOccurs="unbounded"/>
     </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="errorInstances"
  <xsd:complexType>
     <xsd:sequence>
         <xsd:element ref="errorInstance" maxOccurs="unbounded"/>
     </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="errorInstance">
  <xsd:complexType>
```

```
    <xsd:complexContent>
      <xsd:extension base="MessageInstanceType">
        <xsd:sequence>
          <xsd:element name="failedInstanceLink"
                       type="FailedInstanceLinkType" minOccurs="1"
                       maxOccurs="unbounded"/>
          <!-- This will be present only if the retryable attribute
               value is true -->
          <xsd:element ref="messageInstance"/>
        </xsd:sequence>
        <xsd:attribute name="retryable" type="xsd:boolean"
                       use="required"/>
        <xsd:attribute name="serviceType" type="xsd:string"
                       use="required"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="FailedInstanceLinkType">
  <xsd:complexContent>
    <xsd:extension base="InstanceLinkType">
      <xsd:sequence>
          <xsd:element name="failureTime" type="xsd:dateTime"/>
          <xsd:element name="failureTimeString" type="xsd:string"/>
          <xsd:element name="errorMessage" type="xsd:string"/>
          <xsd:element name="exception" type="xsd:string"
                       minOccurs="0"/>
          <xsd:element name="outPayload" type="xsd:string"
                       minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>


<xsd:element name="failedInstances">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="failedInstance" maxOccurs="unbounded"/>
    </xsd:sequence>
    <!-- Indicates more instances are available matching the given
         filter -->
    <xsd:attribute name="availableMore" type="xsd:boolean"/>
  </xsd:complexType>
</xsd:element>

<!-- Describes the details of an errored instance -->
<xsd:element name="failedInstance">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="failureTime" type="xsd:dateTime"/>
      <xsd:element name="errorMessage" type="xsd:string"/>
      <xsd:element name="exception" type="xsd:string" minOccurs="0"/>
      <xsd:element name="inPayload" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="flowId" type="xsd:string" use="required"/>

    <!-- Id of the system in which the deferred subscription failed
     -->
```

```
    <xsd:attribute name="systemId" type="xsd:string"/>

    <!-- Qname of the failed service -->
    <xsd:attribute name="serviceQName" type="xsd:string"/>

    <!-- Name of the failed operation -->
    <xsd:attribute name="operationName" type="xsd:string"/>

    <!-- Indicates If the instance is retryable -->
    <xsd:attribute name="retryable" type="xsd:boolean"
                   use="required"/>

  </xsd:complexType>
</xsd:element>


<!-- Contains the list of Ids to be used to resubmit the failed instances -->
<xsd:element name="resubmitInstanceIds">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="resubmitInstanceId"
                   type='ResubmitInstanceId' maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="onError" default="SkipAndContinue">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <!-- Indicates that the resubmission should continue, even
            if the resubmission failed for an id. At the end,
            the API would return the details on the failed ids -->
          <xsd:enumeration value="SkipAndContinue"/>
          <!-- Indicates that the resubmission should terminate with
              an exception -->
          <xsd:enumeration value="Abort"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <!-- Applicable only if onError is SkipAndContinue -->
    <xsd:attribute name="abortThreshold" type="xsd:integer"/>
    <!-- Delay to be applied between resubmits to avoid message
         flooding -->
    <xsd:attribute name="delay" type="xsd:long"/>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="ResubmitInstanceId">
  <!-- The id of the flow for which the errored instance should be
       resubmitted -->
  <xsd:attribute name="flowId" type="xsd:string" use="required"/>
  <!-- The id of the system in which the errored instance should be
       processed -->
  <xsd:attribute name="systemId" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:element name="resubmissionFailureReport">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="ResubmitInstanceId">
        <xsd:sequence>
          <xsd:element name="reason" type="xsd:string"/>
          <xsd:element name="exception" type="xsd:string"
```

```
                        minOccurs="0"/>
          </xsd:sequence>
          <xsd:attribute name="aborted" type="xsd:boolean"/>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="instanceManage">
    <xsd:complexType>
        <!-- True if instance tracking is enabled. or false -->
        <xsd:attribute name="enable" type="xsd:boolean"
                       use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="purgeInstance">
    <xsd:complexType>
        <!-- long value indicating the date. The instances, which are
             older than the given date would be deleted.
          If userPurgeTimePeriod=0, all the instances would be deleted
        -->
        <xsd:attribute name="userPurgeTimePeriod" type="xsd:long" use="required"/>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

C

Resubmission APIs

This appendix describes the resubmission APIs that allow you to execute all the actions that can be performed in the ESB control, on the ESB DT server. This appendix covers the following sections:

- Get the List of Instances on page C-1
- Get the List of Instances That Raised an Error on page C-2
- Get All the Errors That Occurred in a Given Flow on page C-2
- Get the XML to Draw the Instance Diagram on page C-3
- Resubmit a List of Instances by IDs on page C-3
- Resubmit an Instance by ID on page C-4
- Resubmit an Instance with Modified or Unmodified Payload on page C-4
- Get the Current Status of Instance Tracking on page C-5
- Enable or Disable Instance Tracking on page C-5
- Purge the Instances Based on Time on page C-6
- Retrieve All Service Metadata on page C-6
- Update Service Status on page C-7

Note:

- If an error occurs while executing the resubmission APIs, then an exception is thrown back to the caller.
- Refer to the following link for more information about resubmission APIs

  `http://www.oracle.com/technology/products/integration/esb/files/clientapi.zipp`

Get the List of Instances

This API returns a list of instances matching the given filter.

Action

GetInstances

XSD

- Input XSD Element: Refer to the element `instanceFilter` in Instance Tracking section of the XML schema definition.
- Output XSD Element: Refer to the element `instances` in Instance Tracking section of the XML schema definition.

Sample

The following code snippet returns all the instances created in the last 24 hours:

```
String filter="<instanceFilter timeZone=\"GMT+05:30\">"+
"<startTime>86400000</startTime>"+"</errorIinstanceFilter>";
 HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("filter", URLEncoder.encode(filter,"UTF-8"));
ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);
String data = client.perform("GetInstances", requestProps);
System.out.println("Instances ==> "+data);
```

Get the List of Instances That Raised an Error

This API returns a list of errors matching the given filter.

Action

GetFailedInstances

XSD

- Input XSD Element: Refer to the element `failedInstanceFilter` in Instance Tracking section of the XML schema definition.
- Output XSD Element: Refer to the element `failedInstances` in Instance Tracking section of the XML schema definition.

Sample

The following code snippet returns all the instances created in the last 24 hours that raised retryable and nonretryable errors:

```
String filter = "<failedInstanceFilter
timeZone=\"GMT+05:30\">"+"<startTime>86400000</startTime>"+
"</failedIinstanceFilter>";
HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("filter", URLEncoder.encode(filter,"UTF-8"));
ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);
String data = client.perform("GetFailedInstances", requestProps);
System.out.println("Failed Instances ==> "+data);
```

Get All the Errors That Occurred in a Given Flow

This API returns all the errors that occurred in a given flow ID. If the error is retryable, then the corresponding error element contains the list of routing rules that are executed upon resubmission, and the payload that the user can modify before resubmission.

Action

GetErrorInstance

XSD

- Output XSD Element: Refer to the element `errorInstances` in Instance Tracking section of the XML schema definition.

Sample

The following code snippet returns all the instances that raised errors for the given instance ID.

```
HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("instanceID",URLEncoder.encode("102@1164094022","UTF-8"));
ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);
String data = client.perform("GetErrorInstance", requestProps);
System.out.println("Error Instances ==> "+data);
```

Get the XML to Draw the Instance Diagram

This API returns the XML data containing the flow details. This XML data is used to draw the instance diagram on the console.

Action

GetTrackingDetails

XSD

- Output XSD Element: Refer to the element `relationship` in Instance Tracking section of the XML schema definition.

Sample

The following code snippet returns the XML data required to draw the instance diagram for the given flow ID.

```
HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("flowId",URLEncoder.encode("102@1164094022","UTF-8"));
ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);
String data = client.perform("GetTrackingDetails", requestProps);
System.out.println("Instance Diagram XML ==> "+data);
```

Resubmit a List of Instances by IDs

This API resubmits multiple instances that failed in multiple systems and flows. The input for this action is a list of IDs. In this case, `FlowId` and `SystemId` form a unique `instanceId`. During the process of resubmitting the instances, the console back end retrieves the actual payloads for these IDs from the instance store.

Action

ResubmitInstancesByIds

XSD

- Input XSD Element: Refer to the element `resubmitInstanceIds` in Instance Tracking section of the XML schema definition.
- Output XSD Element: Refer to the element `resubmissionFailureReport` in Instance Tracking section of the XML schema definition.

Sample

```
HashMap<String,String> requestProps = new HashMap<String,String>();
String ids = "<resubmitInstanceIds>" + "<resubmitInstanceId
flowId=\"102@1164094022\" systemId=\"96DD76C0971311DABF1A87858E4395A7\"/>" +
"<resubmitInstanceId flowId=\"101@1164094022\"
systemId=\"96DD76C0971311DABF1A87858E4395A7\"/>" + "</resubmitInstanceIds";
requestProps.put("resubmitIds", URLEncoder.encode(ids,"UTF-8"));
ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);
String data = client.perform("ResubmitInstancesByIds",requestProps);
System.out.println("Resubmit Instances By Ids Status ==> "+data);
```

Resubmit an Instance by ID

This API resubmits a single instance that failed in the given system that was a part of the given flow. The input for this action is an ID. In this case, `FlowId` and `SystemId` form a unique `instanceId`. During the process of resubmitting the instance, the console back end retrieves the actual payload for the given ID from the instance store.

Action

ResubmitInstanceById

Sample

```
HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("flowId", flowId);
requestProps.put("systemId", systemId);
ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);
client.perform("ResubmitInstanceById",requestProps);
```

Resubmit an Instance with Modified or Unmodified Payload

This API resubmits a single instance that failed in the given system as part of the given flow. The input for this action is an XML document containing the modified/unmodified payload.

The actions mentioned in the Get the List of Instances and Get the List of Instances That Raised an Error sections can also be used to obtain the list of instances with payload that raised errors. In a loop, the user needs to call this action to submit each error instance.

Action

ResubmitInstance

XSD

- Input XSD Element: Refer to the element `messageInstance` in Instance Tracking section of the XML schema definition.

Sample

```
HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("resub", URLEncoder.encode(resubInstance,"UTF-8"));
ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);
client.perform("ResubmitInstance",requestProps);
```

Get the Current Status of Instance Tracking

This API returns `true` if the instance tracking is enabled or `false`.

Action

GetTrackingConfig

XSD

- Output XSD Element: Refer to the element `instanceManage` in Instance Tracking section of the XML schema definition.

Sample

The following code snippet returns current status of instance tracking:

```
ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);
client.perform("GetTrackingConfig",null);
```

Enable or Disable Instance Tracking

This API enables or disables instance tracking.

Action

UpdateTrackingConfig

XSD

- Input XSD Element: Refer to the element `instanceManage` in Instance Tracking section of the XML schema definition.

Sample

The following code snippet disables instance tracking:

```
String config = "<instanceManage enable=\"false\" />"HashMap<String,String>
requestProps = new HashMap<String,String>();
requestProps.put("root", config);
ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);
client.perform("UpdateTrackingConfig", requestProps);
```

Purge the Instances Based on Time

This API purges instances based on the following criteria:

- All
- Older than a specified time

Action

UpdateTrackingConfig

XSD

- Input XSD Element: Refer to the element `purgeInstance` in Instance Tracking section of the XML schema definition.

Sample

The following code snippet deletes all the instances that are older than 10 minutes:

```
String purge = "<purgeInstance userPurgeTimePeriod=\"600000\"/>";
HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("root", purge);
ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);
client.perform("UpdateTrackingConfig", requestProps);
```

Retrieve All Service Metadata

This API lists all service data as shown in the ESB Control Service Metadata view.

Action

ExploreServices

XSD

- Output XSD Element: Not available.

Sample

The following code lists all service metadata:

```
ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);
String data = client.perform("UpdateTrackingConfig", null);
System.out.println("Service XML ==> "+data);

<metadata>
   <system name="DefaultSystem" guid="96DD76C0971311DABF1A87858E4395A7"
qname="DefaultSystem" status="ENABLED">
      <versionInfo>
         <ID>1139232135202</ID>
         <lastModifiedTime>Thu Jun 29 08:30:55 PDT 2006</lastModifiedTime>
         <lastModifiedUser>JDev</lastModifiedUser>
      </versionInfo>
      <service name="GetMovies_SOAP" guid="4ACA9F5083F911DBBFB6F366CB962EB8"
qname="DefaultSystem.GetMovies_SOAP"
status="ENABLED" serviceType="ExternalService" typeDescription="External Service"
isWSDLEditable="false">
         <versionInfo>
```

```
            <ID>1190913146234</ID>
            <lastModifiedTime>Thu Sep 27 10:12:26 PDT 2007</lastModifiedTime>
            <lastModifiedUser>oc4jadmin</lastModifiedUser>
         </versionInfo>
         <parent guid="96DD76C0971311DABF1A87858E4395A7" qname="DefaultSystem"
type="system"/>
         <operations>
            <operationInfo guid="4ADFFC1083F911DBBFB6F366CB962EB8"
qname="DefaultSystem.GetMovies_SOAP.GetTheatersAndMovies"
wsdlOperation="GetTheatersAndMovies" mepDisplayName="Request Response"
mep="RequestResponse"/>
            <operationInfo guid="4ADFFC1183F911DBBFB6F366CB962EB8"
qname="DefaultSystem.GetMovies_SOAP.GetUpcomingMovies"
wsdlOperation="GetUpcomingMovies" mepDisplayName="Request Response"
mep="RequestResponse"/>
         </operations>
         <endpointProperties>
            <property name="RetryInterval" value="0"/>
            <property name="RetryCount" value="0"/>
         </endpointProperties>
      </service>
```

Update Service Status

This API sets the service status to `ENABLED` or `DISABLED`.

Action

UpdateServiceStatus

XSD

- Output XSD Element: No response.

Sample

The following code updates the Service Status:

```
HashMap<String,String> requestProps = new HashMap<String,String>();
requestProps.put("service", guid);
requestProps.put("status", status);
ConsoleClient client = ConsoleClientFactory.getConsoleClient(HOST,PORT,USER_
NAME,PASSWORD);
client.perform("UpdateServiceStatus",requestProps);
```

Index

A

adapter services
  access to data sources and, 1-2
  AQ, 1-3
  configuring, 4-1
  creating, 4-3
  database, 1-3
  deleting, 4-4
  described, 1-2
  error handling with inbound, 12-2
  example creating an inbound file, 4-4
  file, 1-3
  FTP, 1-3
  inbound, 1-3
  JMS, 1-3
  modifying, 4-4
  Native MQSeries, 1-3
  Oracle Applications (OA), 1-3
  outbound, 1-3
  using, 4-2
adapters
  supported by Oracle JDeveloper, 1-6
add-dayTimeDuration-to-dateTime function
  description, A-2
Apache Axis
  connectivity with, 1-2
  SOAP invocation services and, 1-2
Application Navigator tab
  New Gallery dialog, 2-2
  Oracle JDeveloper, 2-1
  Register with ESB, 2-2
applications
  creating, 2-11
  ESB, 2-11
Applications Navigator
  Oracle JDeveloper and, 1-7
AQ adapter services
  defined, 1-3
arrays
  in transformations, 6-16
asynchronous execution
  JMS and, 1-5
auto mapping
  in transformations, 6-17
auto mapping with confirmation
  in transformations, 6-18

B

BPEL partner link
  to an ESB service, 4-12

C

class
  ConsoleClientFactory, B-1
clusters
  Oracle Enterprise Service Bus, 9-2
compare function
  description, A-2
compare-ignore-case function
  description, A-3
Component Palette
  Oracle JDeveloper, 2-2
concrete WSDL
  calling an ESB service, 4-13
concrete WSDL URL
  using to invoke a service, 1-4
conditional processing
  with xsl choose, 6-15
  with xsl if, 6-14
Connection Navigator tab
  Oracle JDeveloper, 2-1
connections
  creating a database, 2-8
  creating an application server, 2-7
  creating an integration server, 2-8
  creating and testing in Oracle JDeveloper, 2-6
  testing in Oracle JDeveloper, 2-10
connectivity
  example of, 1-2
  Oracle Enterprise Service Bus, 1-2
  SOAP invocation services and, 1-2
ConsoleClient Interface, B-1
  perform function, B-1
ConsoleClientFactory class, B-1
constant values
  in transformations, 6-7
Create Routing Service dialog
  opening, 5-13
  Oracle JDeveloper, 2-3

specifying the WSDL file, 5-14
create-delimited-string function
description, A-3
creating
domain-value map, 7-2
cross reference table look up, 8-12
xref
lookupXRef function, 8-12
lookupXRef1M function, 8-14
cross reference tables, 8-1
creating, 8-5
deleting values, 8-15
looking up, 8-12
modifying, 8-5
populating columns, 8-6
xref
lookupXRef function, 8-12
lookupXRef1M function, 8-14
markForDelete function, 8-15
populateXRefRow function, 8-6
populateXRefRow1M function, 8-10
cross references
creating, 8-5
introduction, 8-1
modifying, 8-5
overview, 8-1
schema definition file, 8-18
current-date function
description, A-4
current-dateTime function
description, A-4
current-time function
description, A-4
custom ant tasks
deployESBProjects, 2-15

D

data mapper
document transformation and, 1-4
purpose of, 1-4
database adapter services
defined, 1-3
day-from-dateTime function
description, A-5
Definition tab
services, 3-4
definitions
services in Oracle ESB Control, 3-12
deleting
adapter services, 4-4
ESB projects, 2-16
deleting a routing service, 5-33
deleting cross reference table value, 8-15
xref
markForDelete function, 8-15
deployed services
browsing, 4-10
deployment, 2-15
custom ant tasks, 2-15
Design tab
Oracle JDeveloper, 2-2
details
displaying in Diagram panel, 3-4
Details panel
on Diagram tab, 3-4
Diagram panel
displaying details, 3-4
Diagram tab
Details panel, 3-4
Navigator panel, 3-4
dictionaries
in transformations, 6-20
doc function
description, A-5
document transformation
data mapper and, 1-4
Oracle Enterprise Service Bus, 1-4
reuse of, 1-4
document transformations
routing rules and, 1-5
domain-value maps
creating, 7-2
creating and populating, 7-1
editing, 7-10
importing an existing map, 7-7
importing rows into an existing map, 7-8
overview, 7-1
using in a transformation map, 7-12

E

elements
ignoring in XSLT documents, 6-23
endpoint properties
specifying, 4-11
using, 4-11
ends-with function
description, A-5
error conditions
managing, 12-2
error handling
inbound adapters, 12-2
managing, 12-2
Oracle Enterprise Service Bus, 12-1
overview, 12-1
resubmitting messages, 12-4
user, 12-2
ESB metadata
moving, 9-15
ESB Metadata Server
overview, 1-1
ESB Server
overview, 1-1
registering ESB projects, 2-15
starting, 1-13
starting and stopping on Linux, 1-14
starting and stopping on Windows, 1-14
stopping, 1-13
syncing services, 2-15

execution priority
routing rules and, 1-5
execution types
asynchronous, 1-5
defined, 1-5
synchronous, 1-5
Expression Builder
specifying a trackable field, 3-18

F

file adapter services
defined, 1-3
filter expressions
defined, 1-4
routing rules and, 1-4
format-dateTime function
description, A-6
format-string function
description, A-6
FTP adapter services
defined, 1-3
functions
add-dayTimeDuration-to-dateTime, A-2
chaining in transformations, 6-10
compare, A-2
compare-ignore-case, A-3
create-delimited-string, A-3
current-date, A-4
current-dateTime, A-4
current-time, A-4
day-from-dateTime, A-5
descriptions, 6-8
doc, A-5
editing in transformations, 6-9
editing XPath expressions in transformations, 6-13
ends-with, A-5
format-dateTime, A-6
format-string, A-6
functions prefixed with xp20 or orcl, 6-8
generate-guid, A-6
get-content-as-string, A-7
getInboundResponseHeader, A-8
get-localized-string, A-7
getRequestHeader, A-8
hours-from-dateTime, A-8
implicit-timezone, A-9
in transformations, 6-8
index-within-string, A-9
last-index-within-string, A-9
left-trim, A-10
lookup-dvm, A-10
lookup-table, A-11
lookup-xml, A-12
lower-case, A-12
minutes-from-dateTime, A-12
month-from-dateTime, A-13
prefixed with xp20 or orcl, 6-8
query-database, A-13
right-trim, A-14
seconds-from-dateTime, A-14
sequence-next-val, A-14
setOutboundHeader, A-15
setResponseHeader, A-15
square-root, A-16
subtract-dayTimeDuration-from-dateTime, A-16
timezone-from-dateTime, A-16
upper-case, A-17
user-defined in transformations, 6-11
year-from-dateTime, A-17

G

generate-guid function
description, A-6
get the list of instances, B-2
get-content-as-string function
description, A-7
GetErrorInstance, B-2
GetFailedInstances, B-2
getInboundResponseHeader function
description, A-8
GetInstances, B-2
get-localized-string function
description, A-7
getRequestHeader function
description, A-8
getting started
Oracle JDeveloper, 2-5
GetTrackingConfig, B-2
GetTrackingDetails, B-2

H

header transformation and filtering, 5-27
header-based routing, 5-27
heap size
increasing, 6-27
hours-from-dateTime function
description, A-8

I

icons
used in Oracle ESB Control, 3-8
implicit-timezone function
description, A-9
import
user-defined functions in the XSLT Mapper, 6-11
inbound adapter services
defined, 1-3
index-within-string function
description, A-9
instance tracking
XML schema file, B-8
Instances view
Details tab, 10-6
disabling tracking, 10-6
elements and controls, 10-2
enabling tracking, 10-6

Oracle ESB Control, 3-5, 10-1
Overlay tab, 10-7
processing statistics, 10-7
purging messages, 10-9
searching for messages, 10-7

J

JMS
asynchronous execution type and, 1-5
JMS adapter services
defined, 1-3

L

last-index-within-string function
description, A-9
left-trim function
description, A-10
log files
configuring in Oracle Enterprise Manager, 9-13
viewing in Oracle Enterprise Manager, 9-12
looking up cross reference tables, 8-12
xref
lookupXRef function, 8-12
lookupXRef1M function, 8-14
lookup-dvm function
description, A-10
lookup-table function
description, A-11
lookup-xml function
description, A-12
lower-case function
description, A-12

M

maps
domain-value, 7-1
Maps view
Oracle ESB Control, 3-6
maxOccurs attribute
setting for transformations, 6-27
messages
purging in Instances view, 10-9
resubmitting on error, 12-4
searching in the Instances view, 10-7
Microsoft .NET
SOAP invocation services and, 1-2
Microsoft .Net
connectivity with, 1-2
minOccurs attribute
setting for transformations, 6-27
minutes-from-dateTime function
description, A-12
modes
xref
populateXRefRow function, 8-6
populateXRefRow1M function, 8-10
modifying a routing service, 5-33
month-from-dateTime function
description, A-13
moving ESB metadata, 9-15

N

named templates
creating, 6-11
in functions, 6-11
Native MQSeries adapter services
defined, 1-3
Navigator panel
on Diagram tab, 3-4
notification channels
email, 9-5
mobile, 9-9
phone, 9-9
setting up, 9-4
wireless provider for voice, 9-7

O

Oracle Applications (OA) adapter services
defined, 1-3
Oracle BPEL Process Manager
connectivity with, 1-2
SOAP invocation services and, 1-2
Oracle Database Lite
starting, 1-13
Oracle Enterprise Service Bus
adapter services, 4-1
administrative stages, 9-1
architecture, 1-8
checking log files
log files
Oracle Enterprise Service Bus, 9-11
clusters, 9-2
components, 1-1
configuring InterConnect adapter, 9-16
developing with Oracle JDeveloper, 2-1
error handling, 12-1
features, 1-2
managing error conditions, 12-2
notification channels, 9-4
overview, 1-1
planning resources, 9-2
routing rules, 5-1
routing services, 5-1
Secure Sockets Layer (SSL), 9-4
security, 9-4
service groups, 2-12
starting and stopping components, 1-13
systems, 2-12
testing, 9-9
tools for creating, configuring, and managing, 1-5
WSIF, 1-2
Oracle ESB Control
creating a service group, 3-10
creating a system, 3-8
creating or modifying routing rules, 3-14

creating, viewing, and updating structures, 3-7
defining and updating trackable fields, 3-17
enabling and disabling services, 3-12
icons used in, 3-8
Instances view, 3-5, 10-1
introduction to, 1-7
layout, 3-3
managing routing rules, 3-14
managing systems and service groups, 3-8
managing trackable fields, 3-16
managing tracking fields, 3-16
Maps view, 3-6
opening, 1-13, 1-15
overview, 1-1, 3-1
Services view, 3-3
understanding trackable fields, 3-17
viewing and updating service definitions, 3-12
viewing or modifying a service group, 3-11
viewing or modifying a system, 3-9
viewing routing rules, 3-15
viewing service definitions, 3-12
views, 3-3
Oracle InterConnect
configuring the adapter with ESB, 9-16
Oracle JDeveloper
adapters supported by, 1-6
Application Navigator tab, 2-1
Applications Navigator, 1-7
Component Palette, 2-2
Connection Navigator tab, 2-1
Create Routing Service dialog, 2-3, 5-13
creating a database connection, 2-8
creating an application server connection, 2-7
creating an integration server connection, 2-8
creating and modifying routing services, 5-13
creating and testing connections, 2-6
Design tab, 2-2
getting started, 2-5
introduction to, 1-6
overall process and tasks, 2-5
overview, 1-1, 2-1
routing rules, 5-18
routing services and, 1-6
SOAP services, 1-6
starting, 1-13, 1-14
testing a connection, 2-10
transformations and, 1-6
outbound adapter services
defined, 1-3

P

perform function, B-1
GetErrorInstance, B-2
GetFailedInstances, B-2
GetInstances, B-2
GetTrackingConfig, B-2
GetTrackingDetails, B-2
ResubmitInstance, B-2
ResubmitInstancesById, B-2
ResubmitInstancesByIds, B-2
UpdateTrackingConfig, B-2
using, B-2
populating cross reference tables, 8-6
xref
populateXRefRow function, 8-6
populateXRefRow1M function, 8-10
port numbers
viewing, 2-6
projects
adding content, 2-14
creating, 2-11
deleting, 2-16
ESB, 2-11
importing files into, 2-14
registering with ESB Server, 2-15
Properties tab
Oracle ESB Control, 3-4

Q

query-database function
description, A-13

R

registering ESB projects with ESB Server, 2-15
repeating elements
in transformations, 6-16
reports
correcting memory errors when generating for transformations, 6-27
customizing sample XML generation for transformations, 6-27
generating for transformations, 6-26
ResubmitInstance, B-2
ResubmitInstancesById, B-2
ResubmitInstancesByIds, B-2
resubmitting messages
on error conditions, 12-4
right-trim function
description, A-14
routing
content-based, 1-4
header-based, 1-4, 5-27
XML messages, 1-4
routing rules
accept messages from, 5-13
asynchronous invocation, 5-13
creating, 5-7
creating or modifying in Oracle ESB Control, 3-14
defined, 1-4
deleting, 3-16
document transformations and, 1-5
execution priority and, 1-5
execution type and, 1-5
filter expression, 5-10
filter expressions and, 1-4
introduction, 5-1
invocation type, 5-13

managing in Oracle ESB Control, 3-14
overview, 2-4
priority, 5-13
specifying, 5-7
specifying in Oracle JDeveloper, 5-18
synchronous invocation, 5-13
target service and operation, 5-9
transformation map, 5-12
viewing in Oracle ESB Control, 3-15
Routing Rules tab
Oracle ESB Control, 3-4
routing service
defined, 1-4
routing services
Create Routing Service dialog, 5-13
creating, 5-2
creating and modifying in Oracle JDeveloper, 5-13
deleting, 5-33
header transformation and filtering, 5-27
introduction, 5-1
modifying, 5-33
Oracle JDeveloper and, 1-6
overview, 2-4
specifying the WSDL file, 5-14
specifying WSDL, 5-2

S

scenario
used for tutorial, 1-12
schema definition file
cross references, 8-18
schemas
replacing in the XSLT Mapper, 6-24
seconds-from-dateTime function
description, A-14
sequence-next-val function
description, A-14
service groups
creating, 2-13
creating in Oracle ESB Control, 3-10
ESB, 2-12
managing in Oracle ESB Control, 3-8
Oracle Enterprise Service Bus, 3-8
restrictions on, 2-13
viewing or modifying in Oracle ESB Control, 3-11
service icon
used in Oracle ESB Control, 3-8
services
browsing for deployed, 4-10
calling from an external service, 4-13
creating a BPEL partner link to, 4-12
creation as SOAP services, 1-3
Definition tab, 3-4
enabling and disabling in Oracle ESB Control, 3-12
external, 4-13
managing definitions in Oracle ESB Control, 3-12
managing in Oracle ESB Control, 3-7
overview of definitions, 2-4
syncing from ESB Server, 2-15
viewing definitions in Oracle ESB Control, 3-12
services group icon
used in Oracle ESB Control, 3-8
Services view
Oracle ESB Control, 3-3
setOutboundHeader function
description, A-15
setResponseHeader function
description, A-15
SOAP endpoint property
location, 4-11
SOAP invocation services, 1-2
Apache Axis and, 1-2
Microsoft .NET and, 1-2
Oracle BPEL Process Manager and, 1-2
SOAP services
creating, 4-7
deleting, 4-9
modifying, 4-9
Oracle Enterprise Service Bus services, 1-3
supported by Oracle JDeveloper, 1-6
using, 4-7
square-root function
description, A-16
stacked disks icon
systems and, 3-8
starting and stopping components
Oracle Enterprise Service Bus, 1-13
subtract-dayTimeDuration-from-dateTime function
description, A-16
sync services from ESB Server, 2-15
system icon
used in Oracle ESB Control, 3-8
systems
creating, 2-12
creating in Oracle ESB Control, 3-8
ESB, 2-12
examples of, 2-12, 3-7
managing in Oracle ESB Control, 3-8
Oracle Enterprise Service Bus, 3-7
stacked disks icon and, 3-8
viewing or modifying in Oracle ESB Control, 3-9

T

target services
defined, 1-4
testing
Oracle Enterprise Service Bus, 9-9
testing JDeveloper connections
viewing port numbers, 2-6
timezone-from-dateTime function
description, A-16
trackable fields
defining and updating, 3-17
deleting, 3-20
disabling, 3-19
enabling, 3-19

managing in Oracle ESB Control, 3-16
overview, 3-17
specifying with Expression Builder, 3-18
Trackable Fields tab
Oracle ESB Control, 3-5
tracking fields
managing in Oracle ESB Control, 3-16
transformations
adding XSLT constructs, 6-14
auto mapping, 6-17
auto mapping with confirmation, 6-18
chaining functions, 6-10
correcting memory errors, 6-27
creating an XSL map, 6-3
creating an XSL map from an XSL stylesheet, 6-6
customizing sample XML generation, 6-27
dictionaries, 6-20
editing functions, 6-9
editing XPath expressions, 6-13
error when mapping duplicate elements, 6-3
functions, 6-8
functions prefixed with xp20 or orcl, 6-8
generating optional elements, 6-27
generating reports, 6-26
ignoring elements, 6-23
linking source target nodes, 6-7
named templates in functions, 6-11
Oracle JDeveloper and, 1-6
repeating elements, 6-16
replacing schemas, 6-24
rules, 6-3
setting constant values, 6-7
setting the maximum depth, 6-28
setting the number of repeating elements, 6-27
testing the map file, 6-24
user-defined functions, 6-11
using arrays, 6-16
using the XSLT Mapper, 6-6
viewing unmapped target nodes, 6-19
xsl choose conditional processing, 6-15
xsl if conditional processing, 6-14
tutorial
scenario description, 1-12

U

UpdateTrackingConfig, B-2
upper-case function
description, A-17
user-defined extension functions in expression builder, 5-25
using perform function, B-2
enable/disable instance tracking, B-7
get all the errors occurred in a flow id, B-3
get instance tracking status, B-7
get the list of errored instances, B-3
get the list of instances, B-2
get XML for a flow id, B-4
purge instances based on time, B-7
resubmit a list of instances by ids, B-4
resubmit an instance by id, B-5
resubmit an instance with modified/unmodified payload, B-5
using xref
lookupXRef Function, 8-12
markForDelete function, 8-15
populateXRefRow function, 8-8

W

WSDL
concrete, 4-13
WSDL URL
using to invoke a service, 1-4
WSIF
connectivity, 1-2

X

XML messages
routing, 1-4
XML schema file
instance tracking, B-8
XPath expressions
editing in transformations, 6-13
XPath functions
in transformations, 6-8
xref
lookupXRef function, 8-12
exception reasons, 8-12
parameters, 8-12
using, 8-12
lookupXRef1M function, 8-14
exception reasons, 8-14
parameters, 8-14
markForDelete function, 8-15
exception reasons, 8-15
parameters, 8-15
using, 8-15
populateXRefRow function, 8-6
modes, 8-6
parameters, 8-6
using, 8-8
populateXRefRow1M function, 8-10
modes, 8-10
parameters, 8-10
XSD file
cross references, 8-18
xsl choose
conditional processing, 6-15
xsl if
conditional processing, 6-14
XSL map
creating, 6-3
creating from an XSL stylesheet, 6-6
XSL stylesheet
creating an XSL map, 6-6
XSLT constructs
adding in transformations, 6-14
XSLT Mapper

adding XSLT constructs, 6-14
auto mapping, 6-17
auto mapping with confirmation, 6-18
chaining functions, 6-10
correcting memory errors when generating reports, 6-27
creating a map file, 6-1
creating an XSL map, 6-3
creating an XSL map from an XSL stylesheet, 6-6
customizing sample XML generation for transformations, 6-27
dictionaries, 6-20
editing functions, 6-9
editing XPath expressions, 6-13
error when mapping duplicate elements, 6-3
functions, 6-8
functions prefixed with xp20 or orcl, 6-8
generating optional elements, 6-27
generating reports, 6-26
ignoring elements, 6-23
layout in Oracle JDeveloper, 6-1
linking source and target nodes, 6-7
named templates in functions, 6-11
repeating elements, 6-16
replacing schemas, 6-24
rules, 6-3
setting constant values, 6-7
setting the maximum depth, 6-28
setting the number of repeating elements, 6-27
testing the map file, 6-24
user-defined functions, 6-11
using, 6-6
using arrays, 6-16
viewing unmapped target nodes, 6-19
xsl choose conditional processing, 6-15
xsl if conditional processing, 6-14

Y

year-from-dateTime function
description, A-17