

Oracle® Fusion Middleware

Developing Services with Oracle Service Bus

12c (12.1.3)

E60645-04

October 2016

Documentation for developers that describes how to use the Oracle Service Bus Console and Oracle JDeveloper to create and configure proxy and business services, split-joins, and pipelines; perform message transformation with XQuery, XSLT, and MFL; configure transports, work with JCA adapters, and create custom transports; configure security using WS-Security; use the Service Bus API; and create global JNDI resources.

Oracle Fusion Middleware Developing Services with Oracle Service Bus, 12c (12.1.3)

E60645-04

Copyright © 2008, 2016, Oracle and/or its affiliates. All rights reserved.

Primary Author: Oracle Corporation

Contributing Authors:

Contributors:

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

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

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

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

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

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

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

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

Contents

Preface	xlvi
Documentation Accessibility	xlvi
Conventions.....	xlvi
What's New in This Guide	xlvii
Significant Documentation Changes for 12c (12.1.3).....	xlvii
New and Changed Features for 12c (12.1.3)	xlvii
Part I Introduction to Oracle Service Bus	
1 Learning About Oracle Service Bus	
Differences Between Using this Component in the Cloud and On-Premises Environments	1-1
Oracle Service Bus Overview	1-1
Functional Areas.....	1-2
Adaptive Messaging	1-3
Service Security.....	1-3
Service Virtualization.....	1-4
Configuration Framework	1-4
Service Management.....	1-5
Service Bus Architectural Concepts	1-5
Message Processing.....	1-5
Proxy Service Role in Message Processing	1-6
Transport Layer (Inbound)	1-6
Binding Layer.....	1-7
Pipeline Role in Message Processing.....	1-7
Transport Layer (Outbound)	1-7
Business Service Role in Message Processing	1-7
Service Bus Components	1-7
Service Components	1-7
Message Flows	1-8
Transports, Adapters, and Bindings.....	1-10

Transformation Resources	1-12
Transport and Adapter Related Resources.....	1-13
Schema and Document Resources	1-15
Security Resources.....	1-16
Alert Destinations.....	1-17
Throttling Group Resources	1-17
System Resources	1-18
Service Bus Messaging	1-19
Messaging Models.....	1-19
Message Formats	1-19
Message Context.....	1-20
Content Types	1-20
Using Work Managers with Service Bus.....	1-21
Service Bus Security.....	1-21
Service Bus Security Features	1-21
Service Bus Service Security Model.....	1-21
Oracle Web Services Manager	1-22
Oracle Platform Security Services	1-23
WS-Policies	1-23
Types of Security	1-23
Custom Security Credentials	1-25
Approaches for Designing Service Bus Services.....	1-25
Service Bus Top-Down Roadmap	1-25
Service Bus Bottom-Up Roadmap.....	1-26
Naming Guidelines for Service Bus Components	1-27
Viewing Service Bus Resources in a Web Browser.....	1-27
WSDL Documents	1-28
WS Policies	1-28
Message Format Language (MFL) Resources	1-28
Schema Resources	1-28
Notes About Viewing Service Bus Resources in a Web Browser.....	1-28
Accessibility Options.....	1-29
Setting Accessibility Options in JDeveloper.....	1-29
Notes on Screen Reader Mode	1-29
How to Set Accessibility Options in the Oracle Service Bus Console	1-29
Additional Resources	1-30

2 Getting Started with the Oracle Service Bus Console

Overview of the Oracle Service Bus Console.....	2-1
Service Bus Sessions.....	2-1
Oracle Service Bus Console Layout	2-2
Service Bus Projects and Folders.....	2-3
Service Bus Resources.....	2-4

Oracle Service Bus Console Editors	2-4
Getting Started	2-5
How to Access the Oracle Service Bus Console	2-5
How to Exit the Oracle Service Bus Console	2-6
Working with Sessions.....	2-6
How to Create a Session	2-6
How to Activate a Session.....	2-7
How to Exit a Session.....	2-7
Working with Projects, Folders, and Resources in Oracle Service Bus Console	2-8
How to Locate Services.....	2-8
Working with the Project and Folder Definition Editors	2-9
Create New Projects and Folders for Resources	2-12
How to Clone Projects, Folders, and Resources.....	2-13
How to Rename Projects, Folders, and Resources.....	2-14
How to Move Projects, Folders, and Resources.....	2-15
How to Delete Projects, Folders, and Resources.....	2-15
Viewing and Resolving Conflicts	2-16
How to View Conflicts and Errors.....	2-17
How to Resolve Conflicts and Errors	2-18
Viewing Historical Data.....	2-18
How to View the Changes in the Current Session	2-19
How to View the Existing Sessions.....	2-19
How to View the Changes in an Activated Session	2-20
How to Purge Activated Sessions	2-20
Undoing Changes and Activations	2-21
How to Undo Specific Changes in the Current Session	2-21
How to Undo a Session Activation.....	2-22
Viewing References	2-23
Viewing Resource References.....	2-23
Customizing the Appearance of the Oracle Service Bus Console	2-23
How to Customize Table Views.....	2-24

3 Getting Started with Oracle Service Bus in JDeveloper

JDeveloper Concepts for Service Bus.....	3-1
Application Navigator.....	3-2
Service Bus Overview Editor	3-3
Resource Editors.....	3-3
Components Window.....	3-4
Resources Window.....	3-5
Properties Window	3-5
Structure View	3-6
Log Window.....	3-6
Managing Service Bus Components in JDeveloper	3-6

Creating Service Bus Applications and Projects in JDeveloper	3-6
Guidelines for Creating Applications and Projects	3-6
How to Create a Service Bus Application and Project	3-7
Refactoring Service Bus Projects, Folders, and Resources	3-10
How to Rename a Service Bus Folder or Resource in JDeveloper	3-10
How to Move a Service Bus Folder or Resource in JDeveloper	3-11
How to Delete a Project or Resource:	3-12
How to Clone a Project or Folder:.....	3-12
4 Setting up the Development Environment for JDeveloper	
Creating Server Connections in JDeveloper	4-1
How to Create an Application Server Connection	4-1
How to Create a SOA-MDS Connection	4-1
How to Change the MDS Repository Location.....	4-2
Creating Connection Factories for Oracle JCA Adapters	4-3
Disabling the JMS Reporting Provider	4-4
5 Developing Oracle Service Bus Applications in JDeveloper	
Introduction to the Service Bus Overview Editor	5-1
Service Bus Overview Editor Components	5-1
Transports, Adapters, and Bindings.....	5-4
Project and Overview Diagram Synchronization.....	5-10
Adding Service Bus Components.....	5-10
How to Launch the Service Bus Overview Editor.....	5-11
How to Add a Pipeline	5-11
How to Add a Split-Join	5-12
How to Create a Proxy Service.....	5-12
How to Reuse Existing Proxy Services in the Overview	5-14
How to Create a Business Service.....	5-15
How to Reuse Existing Business Services in the Overview	5-16
How to Invoke Deployed Service Bus and SOA Applications.....	5-16
What You May Need to Know About Adding Components.....	5-17
Modifying and Deleting Components in the Service Bus Overview Editor.....	5-18
How to Edit Components from the Service Bus Overview Editor	5-18
How to Rename Components in the Service Bus Overview Editor	5-19
How to Delete Components in the Service Bus Overview Editor.....	5-19
Synchronizing the Overview Diagram.....	5-19
Wiring Service Bus Components.....	5-20
How to Wire Service Bus Components.....	5-20
How to Delete Wires Between Services	5-21
Attaching Security Policies to Service Bus Components	5-21
Testing Service Bus Components in the Overview Editor.....	5-22
How to Test a Service Bus Component.....	5-22

How to Debug a Service Bus Component	5-22
Deploying a Service Bus Application	5-23

Part II Working with Oracle Service Bus Resources

6 Creating and Configuring Project Resources

Introduction to Service Bus Project Resources	6-2
Project Resources and Sessions in the Oracle Service Bus Console	6-2
Working with Service Accounts	6-2
Service Account Authentication Types	6-2
How to Create Service Accounts	6-4
How to Edit Service Accounts	6-6
How to Delete Service Accounts	6-6
Working with Service Key Providers	6-7
How to Create Service Key Providers	6-8
How to Edit Service Key Providers	6-9
How to Delete Service Key Providers	6-9
Working with Alert Destinations	6-10
Alert Destination Types	6-10
How To Create Alert Destinations	6-11
How to Define Email Recipients for an Alert Destination	6-12
How to Define JMS Recipients for an Alert Destination	6-13
How to Edit Alert Destinations	6-14
How to Delete Alert Destinations	6-14
Working with SNMP	6-15
Working with XML Schemas	6-16
How to Create XML Schemas	6-16
How to Edit XML Schemas	6-17
How to Delete XML Schemas	6-17
Working with XML Documents	6-18
How to Create XML Documents	6-18
How to Edit XML Documents	6-19
How to Delete XML Documents	6-19
Working with JAR Files	6-20
How to Add JAR Files	6-20
How to Update a JAR File	6-21
How to Modify JAR File Dependencies	6-22
How to Delete a JAR File	6-22

7 Creating and Configuring System Resources

Working with JNDI Provider Resources	7-1
Classpath Requirements for JBoss Application Server	7-1
About JBoss Initial Context Factory Environment Properties	7-2

How to View JNDI Provider Resources in the Oracle Service Bus Console.....	7-3
How to Create a JNDI Provider Resource	7-4
How to Edit JNDI Provider Resources.....	7-5
How to Delete JNDI Provider Resources.....	7-6
Working with SMTP Server Resources.....	7-6
How to View SMTP Server Resources in the Oracle Service Bus Console	7-6
How to Create SMTP Server Resources	7-7
How to Configure a Default SMTP Server	7-8
How to Edit SMTP Server Resources	7-8
How to Delete SMTP Server Resources	7-8
Working with Proxy Server Resources.....	7-9
Using Proxy Servers with SSL	7-9
How to View Proxy Server Resources in Oracle Service Bus Console	7-10
How to Create Proxy Server Resources	7-10
How to Edit Proxy Server Resources.....	7-11
How to Delete Proxy Server Resources.....	7-12

8 Creating and Configuring Proxy Services

Introduction to Proxy Services.....	8-1
Proxy Service Definitions	8-2
Service Types and Protocols for Proxy Services	8-2
When to Use SOAP or Any XML Service Types.....	8-3
When to Use the Messaging Service Type.....	8-3
Binding Definitions and Runtime Variables for Proxy Service Types	8-3
Proxy Service Transport Protocol Configuration	8-5
Securing Proxy Services.....	8-5
Service Level Agreement Alert Rules	8-6
Web Services Interoperability Compliance.....	8-6
Creating Proxy Services	8-6
How to Create a Proxy Service.....	8-7
How to Generate a Proxy Service from a JCA Binding Resource	8-8
How to Generate a Proxy Service from an Existing Service in JDeveloper	8-10
How to Generate a Proxy Service from a WSDL Document in JDeveloper	8-10
Configuring Proxy Services.....	8-11
How to Configure General Information for a Proxy Service	8-11
How to Configure a Proxy Service Transport.....	8-13
How to Configure Proxy Service Message Handling	8-14
How to Configure Security for a Proxy Service.....	8-15
How to Configure Service Level Agreement Alerts for a Proxy Service	8-15
Deleting Proxy Services	8-16
How to Delete a Proxy Service.....	8-16
Consuming Proxy Services in JDeveloper with WSIL	8-16
How to Consume Service Bus Proxy Services in JDeveloper with WSIL	8-16

9 Creating and Configuring Business Services

Introduction to Business Services.....	9-1
Business Service Definitions	9-2
Service Types and Protocols for Business Services	9-2
Binding Definitions and Runtime Variables for Business Service Types.....	9-3
Business Service Transport Protocol Configuration	9-3
Message Handling for Business Services.....	9-5
Using Proxy Servers	9-6
Service Level Agreement Alert Rules	9-6
Security and Security Policies for Business Services	9-6
Creating Business Services	9-6
How to Create a Business Service	9-7
How to Generate a Business Service from a JCA Binding Resource	9-8
How to Generate a Business Service from a Proxy Service in JDeveloper	9-10
How to Generate a Business Service from a WSDL Document in JDeveloper	9-10
Configuring Business Services.....	9-11
How to Configure General Information for a Business Service	9-11
How to Configure a Business Service Transport	9-12
How to Configure Business Service Message Handling	9-13
How to Configure Performance for a Business Service.....	9-14
How to Configure Security for a Business Service.....	9-14
How to Configure Service Level Agreement Alerts for a Business Service	9-14
Deleting Business Services	9-14
How to Delete a Business Service	9-14
Improving Performance by Caching Business Service Results.....	9-15
How Result Caching Works.....	9-15
Result Caching Best Practices	9-16
Result Cache Metadata	9-16
Testing Result Caching	9-18
How to Configure a Business Service for Result Caching.....	9-18
Result Caching Advanced Configuration.....	9-20

10 Improving Service Performance with Split-Join

Introduction to Split-Joins	10-1
Static Split-Joins	10-2
Dynamic Split-Join	10-3
Split-Join Operations.....	10-4
Using Split-Join with Content in SOAP Headers	10-6
Transaction Support.....	10-7
Security with Split-Joins	10-7
Split-Join Resource Type and Environment Variable	10-7
Service Level Agreement Alert Rules	10-8

Working with Split-Joins in JDeveloper	10-8
How to Create a Split-Join in JDeveloper	10-8
How to Generate a Split-Join from a WSDL Document in JDeveloper	10-9
How to Display the Components Window and Properties Windows	10-9
How to Configure the Start Node.....	10-10
How to View External Services	10-10
How to Configure Global and Local Variables.....	10-10
How to Configure the Receive Operation	10-11
Adding Communication Operations in JDeveloper.....	10-12
How to Invoke a Service.....	10-12
How to Configure a Reply	10-13
Adding Flow Control Operations in JDeveloper	10-14
How to Create a Container Node.....	10-14
How to Iterate Through a Variable Number of Requests	10-15
How to Process a Fixed Number of Requests in Parallel	10-16
How to Define If-Else Conditional Logic.....	10-16
How to Create Error Handlers	10-18
How to Raise an Error	10-18
How to Re-Raise an Error.....	10-19
How to Repeat an Operation Until it Evaluates to True	10-19
How to Repeat an Operation Until it Evaluates to False.....	10-20
How to Insert a Pause in Processing	10-20
Adding Assign Operations in JDeveloper	10-21
About Transformations and Expressions in Assign Operations	10-21
Assign Operation Expression Resolution	10-22
How to Assign a Value to a Variable.....	10-22
How to Copy a Value from a Source to a Destination Document	10-23
How to Delete a Set of Nodes.....	10-24
How to Insert the Result of an XQuery Expression	10-25
How to Invoke a Java Method in a Split-Join.....	10-26
How to Log Split-Join Data.....	10-27
How to Replace a Node or Its Contents.....	10-27
Working with Split-Joins in the Oracle Service Bus Console	10-28
How to Import a Split-Join into the Console.....	10-28
How to Configure Split-Joins in the Console	10-29
How to Define Service Level Agreement Rules for a Split-Join	10-29
Static and Dynamic Split-Join Samples	10-29
Designing a Static Split-Join.....	10-29
Designing a Dynamic Split-Join	10-32

11 Working with WSDL Documents

WSDL Overview	11-1
WSDL Types.....	11-2

WSDL Messages	11-2
WSDL Port Types	11-3
WSDL Bindings	11-3
WSDL Services and Ports	11-4
WSDL Documents in Service Bus	11-4
Web Service Types	11-4
About Effective WSDL Documents and Generated WSDL Documents	11-9
Services Based on WSDL Ports and on WSDL Bindings	11-10
Effective WSDL Documents for Proxy Services	11-10
Effective WSDL Files for Non-Transport-Type Business Services	11-11
Effective WSDL Files for Transport-Type Business Services	11-12
Examples of Proxy Services Based on a Port and on a Binding	11-12
Importing and Exporting WSDL Resources	11-14
Working with WSDL Documents in JDeveloper	11-14
How to Create a WSDL Resource in JDeveloper	11-14
How to Generate a WSDL File from a Service in JDeveloper	11-16
How to Edit a WSDL Document in JDeveloper	11-17
How to Delete a WSDL Document in JDeveloper	11-17
Working with WSDL Documents in the Oracle Service Bus Console	11-17
How to Create a WSDL Resource in the Console	11-17
How to Export a WSDL File in the Console	11-18
How to Generate a WSDL File from a Service in the Console	11-19
How to Edit a WSDL Document in the Console	11-20
How to Delete a WSDL Document in the Console	11-20
Viewing Effective WSDL Documents	11-20

Part III Working with Oracle Service Bus Pipelines

12 Modeling Message Flow in Oracle Service Bus

Pipeline Components	12-2
Building a Message Flow	12-3
Message Execution	12-3
Branching in Pipelines	12-4
Operational Branching	12-4
Conditional Branching	12-4
Configuring Actions in Stages and Route Nodes	12-5
Communication Actions	12-5
Flow Control Actions	12-6
Message Processing Actions	12-7
Reporting Actions	12-8
Configuring Transport Headers in Pipelines	12-9
Performing Transformations in Pipelines	12-13
Transformations and Publish Actions	12-13

Transformations and Route Nodes.....	12-14
Constructing Service Callout Messages	12-14
SOAP Document Style Services	12-14
SOAP RPC Style Services	12-16
XML Services.....	12-18
Messaging Services	12-18
Using Attachments with Service Callout Messages	12-19
Example of Using Attachments with SOAP-Document Style Services.....	12-19
Example of Using Attachments with SOAP RPC Style Service	12-23
MTOM/XOP Support.....	12-26
Page Attachments to Disk	12-26
Handling Errors as the Result of a Service Callout.....	12-26
Transport Errors	12-27
SOAP Faults	12-28
Unexpected Responses	12-29
Handling Errors in Pipelines.....	12-30
Generating the Error Message, Reporting, and Replying	12-31
Different Behavior of Security Fault Handling in Service Bus 11g and 12c	12-31
Example of Action Configuration in Error Handlers.....	12-32
Using Dynamic Routing	12-33
Implementing Dynamic Routing	12-34
Accessing Databases Using XQuery	12-37
Understanding Message Context	12-38
Message Context Components.....	12-39
Guidelines for Viewing and Altering Message Context	12-41
Copying JMS Properties From Inbound to Outbound	12-42
Working with Variable Structures	12-42
Using the Inline XQuery Expression Editor.....	12-42
Using Variable Structures	12-44
Quality of Service.....	12-45
Delivery Guarantees	12-46
Outbound Message Retries	12-50
Using Work Managers with Service Bus.....	12-51
Content Types, JMS Type, and Encoding.....	12-52
Throttling Pattern	12-53
WS-I Compliance	12-53
WS-I Compliance Checks	12-54
Converting Between SOAP 1.1 and SOAP 1.2.....	12-56

13 Working with Pipelines in Oracle Service Bus Console

Introduction to the Oracle Service Bus Console Pipeline Designer	13-1
Edit Message Flow Page on the Console	13-1
Edit Stage Configuration Page on the Console.....	13-4

Viewing and Editing Pipelines in the Console.....	13-4
How to View and Edit Pipelines in the Console.....	13-4
How to Add Shared Variables to Pipelines in the Console.....	13-5
How to Add Pipeline Pairs to Pipelines.....	13-6
How to Add Conditional Branches to Pipelines in the Console	13-6
How to Add Operational Branches to Pipelines in the Console	13-8
How to Add Stages to Pipelines in the Console	13-9
How to Add Route Nodes to Pipelines in the Console.....	13-9
Cutting, Copying, and Pasting Stages and Route Nodes	13-10
Configuring the Resequencer in the Console	13-10
How to Configure Resequencing in a Pipeline in the Console.....	13-11
How to Select the Resequence Level in the Console	13-12
How to Configure the Resequencing Mode in the Console.....	13-13
Creating Variable Structure Mappings.....	13-15
Sample WSDL Document.....	13-15
Creating the Resources You Need for the Examples	13-16
Example 1: Selecting a Predefined Variable Structure.....	13-18
Example 2: Mapping a Variable to a Type.....	13-19
Example 3: Mapping a Variable to an Element.....	13-21
Example 4: Mapping a Variable to a Child Element	13-22
Example 5: Mapping a Variable to a Business Service.....	13-22
Example 6: Mapping a Child Element to Another Child Element.....	13-24

14 Working with Pipeline Actions in Oracle Service Bus Console

Adding and Editing Pipeline Actions in the Console	14-2
Adding Publish Actions in the Console	14-7
Adding Publish Table Actions in the Console.....	14-7
Adding Dynamic Publish Actions in the Console	14-8
Adding Routing Options Actions in the Console	14-9
Adding Service Callout Actions in the Console	14-11
Adding Transport Header Actions in the Console	14-13
Setting Cookies in Outbound HTTP Transport Headers	14-15
Adding Dynamic Routing to Route Nodes in the Console	14-16
Adding Routing Actions to Route Nodes in the Console.....	14-17
Adding Routing Tables to Route Nodes in the Console.....	14-18
Adding For-Each Actions in the Console.....	14-19
Adding If-Then Actions in the Console.....	14-20
Adding Raise Error Actions in the Console.....	14-20
Transactions	14-21
Adding Reply Actions in the Console	14-21
Adding Resume Actions in the Console.....	14-21
Adding Skip Actions in the Console.....	14-22
Adding Assign Actions in the Console	14-22

Adding Delete Actions in the Console	14-22
Adding Insert Actions	14-23
Adding Java Callout Actions in the Console	14-24
Adding MFL Translate Actions in the Console	14-25
Adding nXSD Translate Actions	14-26
Adding Rename Actions in the Console	14-27
Adding Replace Actions in the Console	14-28
Adding Validate Actions in the Console	14-29
Adding Alert Actions in the Console	14-30
Adding Log Actions in the Console	14-31
Adding Report Actions in the Console	14-32
Adding Error Handlers in the Console	14-33
Adding Pipeline Error Handlers in the Console	14-33
Adding Stage Error Handlers in the Console	14-34
Adding Route Node Error Handlers in the Console	14-35
Editing Error Handlers in the Console	14-35
Disabling an Action or a Stage in the Console	14-36
Disabling an Action on the Pipeline	14-36
Re-Enabling an Action in the Pipeline	14-37
Disabling a Stage in the Pipeline	14-37
Re-Enabling a Stage in the Pipeline	14-37

15 Working With Expression Editors in Oracle Service Bus Console

Creating and Editing Inline XQuery and XPath Expressions	15-1
Understanding XQuery Editor Layouts and Tasks	15-3
Palettes	15-3
Workspace	15-4
Property Inspector	15-4
Building Expressions in the Editor Workspace Text Fields	15-4
Creating Namespaces to Use in Inline Expressions	15-7
Creating Variable Structures in the XQuery Editors	15-8
Creating Custom XPath Functions in the XQuery Editors	15-13
Binding External XQuery Resources to Inline XQueries	15-13
Binding External XSLT Resources to Inline XQueries	15-14
Binding Dynamic XQuery Expressions to Inline XQueries	15-15
Binding Dynamic XSLT Expressions to Inline XQueries	15-16
Entering XQuery Comparison Expressions Using the Builder Option	15-17
Entering Unary Expressions Using the Builder Option	15-18

16 Working with Pipelines in Oracle JDeveloper

Adding a Pipeline Component in JDeveloper	16-1
How to Add a Pipeline in JDeveloper	16-1
Viewing and Editing Pipelines in JDeveloper	16-2

How to View and Edit a Pipeline in JDeveloper	16-2
Adding Shared Variables to Pipelines in JDeveloper.....	16-4
How to Add a Shared Variable to a Pipeline in JDeveloper	16-5
Adding Pipeline Pair Nodes to Pipelines in JDeveloper	16-5
How to Add a Pipeline Pair Node to a Pipeline in JDeveloper	16-5
Adding Conditional Branches to Pipelines in JDeveloper.....	16-6
How to Add a Conditional Branch to a Pipeline in JDeveloper	16-6
Adding Operational Branches to Pipelines in JDeveloper	16-7
How to Add an Operational Branch to a Pipeline in JDeveloper	16-8
Adding Stages to Pipelines in JDeveloper	16-9
How to Add a Stage to a Pipeline in JDeveloper.....	16-9
Adding Route Nodes to Pipelines in JDeveloper.....	16-10
How to Add a Route Node to a Pipeline in JDeveloper	16-10
Cutting, Copying, and Pasting Stages and Route Nodes in JDeveloper	16-10
Configuring the Resequencer in JDeveloper	16-11
How to Configure Resequencing in a Pipeline in JDeveloper.....	16-11
Selecting the Resequence Level in JDeveloper.....	16-13
How to Configure the Resequencing Mode in JDeveloper.....	16-13

17 Working with Pipeline Actions in Oracle JDeveloper

Adding and Editing Actions in Pipelines in JDeveloper	17-2
Adding Publish Actions in JDeveloper	17-7
Adding Publish Table Actions in JDeveloper.....	17-8
Adding Dynamic Publish Actions in JDeveloper	17-9
Adding Routing Options Actions in JDeveloper	17-10
Adding Service Callout Actions in JDeveloper	17-12
Adding Transport Header Actions in JDeveloper	17-13
Adding Dynamic Routing to Route Nodes in JDeveloper	17-14
Adding Routing Actions to Route Nodes in JDeveloper	17-15
Adding Routing Tables to Route Nodes in JDeveloper	17-16
Adding For Each Actions in JDeveloper	17-17
Adding If Then Actions in JDeveloper	17-18
Adding Raise Error Actions in JDeveloper	17-19
Adding Reply Actions in JDeveloper	17-20
Adding Resume Actions in JDeveloper.....	17-20
Adding Skip Actions in JDeveloper	17-21
Adding Assign Actions in JDeveloper.....	17-22
Adding Delete Actions in JDeveloper	17-23
Adding Insert Actions in JDeveloper.....	17-23
Adding Java Callout Actions in JDeveloper	17-25
Adding MFL Translate Actions in JDeveloper	17-26
Adding nXSD Translate Actions in JDeveloper	17-27
Adding Rename Actions in JDeveloper	17-28

Adding Replace Actions in JDeveloper	17-29
Adding Validate Actions in JDeveloper	17-30
Adding Alert Actions in JDeveloper	17-32
Adding Log Actions in JDeveloper	17-33
Adding Report Actions in JDeveloper	17-34
Adding Error Handlers in JDeveloper	17-35
How to Add Error Handlers in Pipelines in JDeveloper	17-36
Disabling an Action or a Stage in JDeveloper	17-37
Disabling an Action or Stage	17-37
Re-Enable an Action or Stage	17-37

18 Working with Pipeline Templates

Adding a Pipeline Template	18-1
How to Add a Pipeline Template	18-1
Editing a Pipeline Template	18-2
How to Edit a Pipeline Template	18-2
Adding Placeholder Blocks to a Pipeline Template Message Flow	18-5
Locking an Action in a Pipeline Template	18-5
How to Lock an Action in a Pipeline Template	18-5
Creating a Concrete Pipeline from a Pipeline Template	18-6
How to Create a Concrete Pipeline	18-6
Editing the Message Flow for a Concrete Pipeline	18-7
How to Edit the Message Flow for a Concrete Pipeline	18-7
Converting a Concrete Pipeline in to a Regular Pipeline	18-8
How to Break a Template Link for a Concrete Pipeline	18-9

Part IV Transforming Data

19 Transforming Data with XQuery

Introduction to XQuery Transformations	19-1
XQuery Editors and Mappers	19-2
JDeveloper Editors and Mappers	19-2
Oracle Service Bus Console Editors	19-2
Creating XQuery Maps in JDeveloper	19-2
How to Create XQuery Mappings in JDeveloper	19-2
Testing Service Bus Projects Converted from XQuery 2004 to XQuery 1.0 in JDeveloper	19-3
Working with XQuery Resources in the Oracle Service Bus Console	19-3
How to Create an XQuery Resource in the Console	19-4
How to Edit an XQuery Resource in the Console	19-4
How to Delete an XQuery Resource in the Console	19-5
How to Upgrade Your XQuery Resources to use XQuery 1.0	19-5
Service Bus XQuery Functions	19-6
Supported Function Extensions from Oracle	19-6

Function Extensions from Service Bus	19-7
Creating and Using Custom XPath Functions	19-11
20 Transforming Data with XSLT	
Introduction to XSLT	20-1
XSLT Editors and Mappers	20-1
JDeveloper Editors and Mappers.....	20-1
Oracle Service Bus Console Editors	20-2
Creating XSLT Mappings in JDeveloper	20-2
How to Create XSLT Mappings in JDeveloper	20-2
Working with XSLT Resources in the Oracle Service Bus Console.....	20-3
How to Create XSLT Resources in the Console	20-3
How to Edit XSLT Resources in the Console	20-4
How to Delete an XSLT Resource.....	20-4
21 Mapping Data with Cross-References	
Introduction to Cross References	21-1
Cross Reference Database Tables.....	21-1
Cross Reference Functions	21-2
Managing Cross Reference Data at Runtime	21-2
Creating Cross Reference Tables in JDeveloper	21-2
How to Create Cross Reference Tables in JDeveloper	21-2
Working with Cross Reference Resources in the Oracle Service Bus Console	21-3
How to Create Cross Reference (XRef) Resources in the Console	21-3
How to Edit Cross Reference Resources in the Console	21-4
How to Create a Custom Database Table in the Console.....	21-4
Deleting a Cross Reference Resource	21-5
How to Delete a Cross Reference Resource.....	21-5
Populating Cross Reference Tables in Oracle Service Bus.....	21-5
22 Mapping Data with Domain Value Maps	
Introduction to Domain Value Maps.....	22-1
Domain Value Map Functions.....	22-1
Creating Domain Value Maps in JDeveloper	22-2
How to Create a Domain Value Map in JDeveloper	22-2
Working with DVM Resources in the Oracle Service Bus Console.....	22-3
How to Create DVM Resources in the Console	22-3
How to Add Domains to a Domain Value Map	22-4
How to Add Domain Values to a Domain Value Map.....	22-4
How to Edit a Domain Value Map in the Console	22-5
Deleting a Domain Value Map	22-5
How to Delete a Domain Value Map	22-5
Using Domain Value Maps in Expressions and Conditions	22-6

23 Defining Data Structures with Message Format Language

Introduction to the Format Builder	23-1
About MFL Files	23-1
Valid Names for Formats, Fields, and Groups	23-2
Supported Character Delimiters	23-2
Working with MFL Resources in the Oracle Service Bus Console	23-3
How to Create MFL Resources in the Console	23-4
How to Edit MFL Resources in the Console.....	23-4
Creating the MFL Message Structure	23-5
Using Drag and Drop in the Format Builder	23-5
How to Create an MFL File in JDeveloper.....	23-5
How to Create a Group	23-6
How to Create a Field	23-6
How to Reference Groups or Fields.....	23-6
How to Add a Comment	23-7
Configuring the MFL Message Structure.....	23-8
How to Make a Node Recurring	23-8
How to Define Delimiters	23-8
Importing and Converting Metadata.....	23-9
How to Convert a Guideline XML File	23-10
How to Convert an XML Schema	23-10
How to Convert a COBOL Copybook.....	23-10
How to Convert C Structures	23-11
How to Convert an FML Field Table Class	23-12
Deleting MFL Resources.....	23-14
How to Delete an MFL Resource	23-14
Testing Format Definitions.....	23-14
How to Start Format Tester.....	23-14
How to Test Using the Non-XML Window.....	23-15
How to Test Using the XML Window	23-15
How to Test Using the Debug Window	23-15
How to Debug Format Definitions	23-16
Format Tester Command Reference	23-18
Using the Palette	23-20
How to Display the Palette Window	23-20
How to Add Items to the Palette.....	23-20
How to Add Palette Items to a Message Format	23-20
Format Builder Supported Data Types.....	23-21
MFL Data Types	23-21
COBOL Copybook Importer Data Types	23-25
Unsupported C Language Features.....	23-27
Format Builder Field Reference	23-28

Format Builder Window	23-29
Format Builder Tool Bar	23-29
Format Builder Tree Pane	23-30
Field Configuration Window	23-31
Group Configuration Window	23-35
Format Builder Reference Configuration Window	23-37

24 Using Java Callouts and POJOs

Introduction to Java Callouts	24-1
Java Callout Usage Guidelines	24-1
Java Callouts or EJBs	24-2
Working with Streaming Content	24-2
Passing Streaming Content to a Java Callout	24-2
Streaming Content Results from a Java Callout	24-3
Best Practices for Java Callouts and POJOs	24-3

Part V Working with JCA Adapters, Transports, and Bindings

25 Using the JCA Transport and JCA Adapters

Introduction to the JCA Transport	25-1
Supported JCA Adapters	25-2
Oracle JCA Adapter Limitations	25-5
JCA Adapter Framework	25-5
JCA Transport Messaging	25-6
Security for JCA Transports	25-6
Logging	25-7
JCA Transport Error Handling	25-7
URI Rewriting with JCA Transports	25-7
JCA Transport Message Encoding	25-7
Rejected Messages	25-8
JCA Adapter Configuration Recommendations for Service Bus	25-8
Configuring the JCA Adapter Connections	25-8
Configuring JCA Adapters that Poll a Database	25-8
Configuring the Oracle JCA Adapter for Database	25-8
Configuring the Oracle JCA Adapter for AQ	25-9
Configuring the Oracle JCA Adapter for Coherence	25-9
Configuring the Salesforce Cloud Adapter	25-9
Working with JCA Binding Resources	25-9
How to Create a JCA Adapter in JDeveloper	25-10
How to Import JCA Adapters in the Oracle Service Bus Console	25-10
How to Create a JCA Binding Resource in the Oracle Service Bus Console	25-10
How to Edit JCA Binding Resources in the Console	25-11
How to Delete JCA Binding Resources	25-12

Using Custom JCA Adapters.....	25-12
Working with JavaScript Resources.....	25-14
How to Create JavaScript Resources	25-14
How to Edit JavaScript Resources	25-15
How to Delete JavaScript Resources	25-15
JCA Transport Configuration Reference	25-15
JCA Transport Endpoint URIs.....	25-16
JCA Transport Headers and Normalized Message Properties.....	25-16
JCA Transport Endpoint Properties	25-17
JCA Transport Environment Variables	25-20
Configuring Proxy and Business Services to Use the JCA Transport.....	25-20
Proxy Service Operation Configuration.....	25-23

26 Creating REST Services with Oracle Service Bus

Oracle Service Bus and REST	26-1
REST Features in Service Bus	26-1
REST Implementation in Service Bus	26-2
REST Security.....	26-2
WADL Documents for REST Services in Service Bus	26-2
WADL Documents in the Design Time and Runtime	26-3
Query Operations with WADL	26-3
WADL Restrictions	26-3
Effective WADL Documents.....	26-3
Creating WADL Documents	26-4
How to Create a WADL Resource in the Oracle Service Bus Console	26-4
Modifying WADL Documents.....	26-5
How to Edit a WADL Document	26-5
How to Delete a WADL Document	26-5
Creating REST Services.....	26-6
How to Create REST Services for Service Bus.....	26-6
How to Create or Configure a REST Operation.....	26-8
How to Expose an HTTP Proxy or Business Service as REST	26-9
What you May Need to Know About Configuring URI Parameters for REST	26-10
Accessing WADL Documents in a Web Browser	26-11
Viewing WADL Documents in XML Format.....	26-11
Viewing WADL Documents in a Readable Format	26-11

27 Using the DSP Transport

Introduction to the DSP Transport.....	27-1
Enabling Data Services for Service Bus	27-1
Using the DSP Transport	27-1
Generate the WSDL File in Oracle Data Service Integrator	27-2
Create the Service Bus Project	27-2

DSP Transport Configuration Reference.....	27-6
DSP Transport Endpoint URIs	27-6
Configuring Business Services to Use the DSP Transport	27-6
28 Using the EJB Transport	
Introduction to the EJB Transport	28-1
Prerequisites for Creating Services that Invoke EJBs	28-2
Registering a JNDI Provider Resource	28-2
Registering an EJB Client or Converter JAR Resource.....	28-3
Invoking EJB Business Services	28-3
Exposing EJBs as Web Services.....	28-4
Advanced EJB Transport Topics.....	28-4
EJB Transport Transactions.....	28-4
EJB Transport Retries and Failover.....	28-5
EJB Transport Error Handling.....	28-6
Supported Types and Converter Classes.....	28-6
Business Exception Classes	28-7
Troubleshooting EJB Transports.....	28-7
Temp Directories	28-8
Deployed Application.....	28-8
EJB Transport Errors	28-8
EJB Transport Configuration Reference	28-8
EJB Endpoint URI Format	28-9
Configuring Business Services to Use the EJB Transport.....	28-9
29 Using HTTP and Poller Transports	
Introduction to Poller Transports.....	29-1
Using the HTTP Transport	29-1
HTTP Session Stickiness.....	29-2
Retrieving the HTTP Authorization Header in a Proxy Service	29-2
HTTP Transport Configuration Reference	29-2
REST Support.....	29-8
Response Codes and Error Handling for HTTP Business Services	29-11
Using the Email Transport	29-12
Email Transport Configuration Reference.....	29-12
Using the File Transport	29-15
File Transport Configuration Reference.....	29-16
Using the FTP Transport.....	29-18
FTP Transport Configuration Reference.....	29-18
Using the SFTP Transport.....	29-22
SFTP Transport Features	29-22
General Principles of SFTP Authentication	29-23
SFTP Transport Runtime Behavior	29-23

Enabling SFTP Authentication	29-24
Handling SFTP Transport Communication Errors	29-26
Troubleshooting the SFTP Transport	29-26
Importing SFTP Transport Services.....	29-27
SFTP Transport Configuration Reference.....	29-28

30 Using the JEJB Transport

Introduction to the JEJB Transport.....	30-1
Differences Between the JEJB Transport and the EJB Transport.....	30-1
JEJB Transport WSDL Generation	30-2
JEJB Transport Error Handling	30-2
Prerequisites for Creating JEJB Services.....	30-4
Creating and Packaging Your Client EJB JAR File	30-4
Registering a JNDI Provider Resource (Business Services)	30-4
Use Cases	30-5
EJB Invoking an External Service.....	30-5
Non-EJB Client Invoking an EJB	30-6
EJB Invoking EJB	30-7
UDDI Integration.....	30-7
UDDI Publish.....	30-7
UDDI Import.....	30-8
JEJB Transport Configuration Reference.....	30-8
JEJB Transport Endpoint URI.....	30-8
Configuring Proxy Services to Use the JEJB Transport	30-10
Configuring Business Services	30-11
JEJB Transport Environment Values	30-13

31 Using the JMS Transport

Introduction to the JMS Transport.....	31-1
JMS Content Type for Services.....	31-1
JMS Transport Security.....	31-1
Asynchronous Request-Response Messaging.....	31-2
Sending and Receiving Java Objects in Messages	31-2
Required JMS Resources	31-3
Platform Interoperability.....	31-3
Using SOAP Over JMS Transport	31-4
Interoperating with WebLogic Server.....	31-4
Configuring the Response Queues for Cross-Domain JMS Calls	31-5
Naming Guidelines for Domains, Servers, and URIs.....	31-5
JMS Server Names.....	31-5
JNDI Names and Service Bus	31-6
JMS Client ID in Proxy Services.....	31-6
About the Client ID and Subscriber Name.....	31-6

Recommended Usage	31-7
JMS Transport Error Handling	31-7
Application Errors	31-7
Communication Errors	31-7
Pipeline Exceptions with Java Objects	31-8
WSDL-Defined SOAP Fault Messages	31-8
Adding a Fault in a SOAP Message if the Fault is Constructed from inside a Service Bus Pipeline	31-9
Message ID and Correlation ID Patterns for JMS Request/Response	31-10
Overview of JMS Request-Response and Design Patterns.....	31-10
JMS Message ID Pattern	31-12
JMS Correlation ID Pattern	31-13
Comparison of Message ID and Correlation ID Patterns.....	31-14
Interoperating with JAX-RPC Over JMS.....	31-15
JMS Message ID Pattern Examples	31-17
JMS Transport Configuration Reference	31-18
JMS Transport Endpoint URIs.....	31-19
Configuring Proxy Services to Use the JMS Transport.....	31-19
JMS Transport Headers	31-23
Configuring Business Services to Use the JMS Transport.....	31-25

32 Using the Local Transport

Introduction to the Local Transport.....	32-1
Features and Characteristics of Local Transport Proxy Services	32-1
Using Local Transport Proxy Services.....	32-2
Changes from Previous Usage	32-3
Propagating SOAP Faults Between Proxy Services.....	32-4
Using OWSM Security with Local Proxy Services.....	32-4

33 Using the MQ Transport

Introduction to the MQ Transport	33-1
MQ Transport Features	33-1
MQ Transport Advantages	33-2
Messaging Patterns	33-2
MQ Connection Resources.....	33-3
Quality of Service	33-3
MQ Clusters and the MQ Transport.....	33-3
Limitations of the MQ Transport	33-3
Setting Up the Environment for the MQ Transport	33-4
How to Add MQ Client Libraries to Your Environment	33-4
How to Configure Environment Variables.....	33-4
Working with MQ Connections	33-4
How to Create MQ Connections	33-4

How to Edit MQ Connections	33-6
How to Delete MQ Connections	33-7
MQ Transport Error Handling	33-7
Using the WebSphere JMS MQ Interface	33-7
Using the WebSphere MQ JMS Interface.....	33-8
MQ Messaging Types	33-8
Tuning WebSphere MQ.....	33-9
MQ Transport Configuration Reference	33-10
MQ Transport Endpoint URIs	33-10
Configuring Proxy Services to Use the MQ Transport.....	33-10
Configuring Business Services to Use the MQ Transport	33-13
MQ Transport Environment Values	33-16
MQ Transport Headers	33-17
Configuring Transport Headers.....	33-22
About RFH2 Headers.....	33-23

34 Using the Oracle BPEL Process Manager Transport

Introduction to the BPEL Transport.....	34-1
SOAP Support with the BPEL Transport.....	34-2
Transaction Propagation in the BPEL Transport.....	34-2
SSL Support in the BPEL Transport.....	34-2
BPEL Transport Environment Values	34-2
BPEL Transport Simple Use Cases (Synchronous)	34-3
Synchronous: Invoking Processes in Oracle BPEL Process Manager.....	34-3
Synchronous: Calling External Services from Oracle BPEL Process Manager.....	34-4
Associating Messages with the Correct Conversation.....	34-5
Advanced Use Cases (Asynchronous).....	34-5
Asynchronous: Invoking Processes in Oracle BPEL Process Manager	34-5
Asynchronous: Calling Service Providers from Oracle BPEL Process Manager	34-6
BPEL Transport Security.....	34-8
Using SSL from Oracle Service Bus to Oracle Servers	34-8
BPEL Transport Error Handling.....	34-8
Application Errors.....	34-8
Connection Errors	34-9
Other Errors.....	34-9
WS-Addressing Reference	34-9
ReplyTo.....	34-9
MessageID / RelatesTo	34-10
Examples of XML Messaging with the BPEL Transport.....	34-10
Conversation ID Examples	34-10
Asynchronous BPEL to BPEL Through Service Bus Example.....	34-15
BPEL Transport Configuration Reference.....	34-17
BPEL Transport Endpoint URI.....	34-18

Configuring Business Services to Use the BPEL Transport	34-18
35 Using the SB Transport	
Introduction to the SB Transport.....	35-1
SB Transport Features.....	35-1
SB Transport Error Handling.....	35-2
UDDI and the SB Transport	35-3
Publishing a Service	35-3
Importing a Service	35-3
SB Transport Configuration Reference.....	35-4
SB Transport Environment Values	35-4
Configuring Proxy Services to Use the SB Transport	35-4
Configuring Business Services to Use the SB Transport	35-5
36 Using the SOA-DIRECT Transport	
Introduction to the SOA-DIRECT Transport.....	36-1
SOA-DIRECT Transport Features.....	36-1
Service Binding Types	36-2
WS-Addressing for the SOA-DIRECT Transport	36-2
SOA-DIRECT Transport Security	36-3
SOA-DIRECT Transport Error Handling	36-3
Using SOA Suite Services with Service Bus.....	36-3
Simple Use Cases – Synchronous.....	36-4
Advanced Use Cases – Asynchronous.....	36-6
SOA-DIRECT Transport Configuration Reference.....	36-10
SOA-DIRECT Endpoint URIs.....	36-11
Configuring Business Services to Use the SOA-DIRECT Transport.....	36-12
SOA-DIRECT Transport Environment Values	36-14
WS-Addressing Reference.....	36-15
ReplyTo Header.....	36-15
MessageID / RelatesTo Headers.....	36-16
XML Messaging Examples	36-16
Conversation ID Examples	36-16
Asynchronous Composite to Composite Communication Through Service Bus.....	36-21
37 Using the Tuxedo Transport	
Introduction to the Tuxedo Transport.....	37-1
Capabilities of the Tuxedo Transport.....	37-2
Configuring Oracle Tuxedo Connector	37-3
Before You Begin	37-3
Configuring Oracle Tuxedo Connector.....	37-4
Using Tuxedo Services from Service Bus.....	37-4
Configuring a Tuxedo-Based Business Service.....	37-4

Load Balancing and Failover for Tuxedo-Based Business Services	37-6
Error Handling for Tuxedo-Based Business Services	37-6
Testing Your Configuration	37-7
Using Service Bus from Tuxedo	37-7
Configuring a Tuxedo-Based Proxy Service	37-7
Testing Your Configuration	37-7
Tuxedo Transport Buffer Transformation.....	37-8
Buffer Transformation with the Any XML Service Type	37-8
Buffer Transformation with the Messaging Service Type.....	37-9
Tuxedo Transport Transaction Processing.....	37-9
Inbound Tuxedo Service Transaction Processing.....	37-10
Outbound Tuxedo Service Transaction Processing	37-10
Tuxedo Transport Configuration Reference	37-10
Configuring Proxy Services to Use the Tuxedo Transport	37-10
Configuring Business Services to Use the Tuxedo Transport.....	37-12

38 Using the WS Transport

Introduction to the WS Transport	38-1
Web Services Reliable Messaging	38-1
WS Transport Features	38-1
Messaging Patterns	38-2
WS-Policies in the WS Transport	38-3
Streaming Content for Large Messages	38-3
Web Services Interoperability.....	38-3
Authentication and Authorization of Services	38-4
Proxy Service Authentication	38-4
Proxy Service Authorization.....	38-4
Business Service Authentication	38-4
Using the WS Transport.....	38-5
Importing the WSDL Document into the Oracle Service Bus Console	38-5
Configuring WS Policies.....	38-5
Attaching WS Policies to a Service.....	38-6
Configuring an Error Queue.....	38-6
Routing the WS Transport Through an HTTP Proxy Server	38-6
WS Transport Error Handling	38-6
Importing and Exporting Resources.....	38-6
Importing and Publishing Services Using UDDI Registries	38-7
WS Transport Configuration Reference	38-7
Endpoint URIs for the WS Transport	38-7
Configuring Business Services to Use the WS Transport	38-7
Configuring Proxy Services to Use the WS Transport.....	38-9

Part VI Creating Custom Transport Providers

39 Learning About Custom Transport Providers

Introduction to Transport Providers.....	39-1
Introduction to the Transport SDK	39-2
Transport SDK Features	39-2
Transport Provider Modes.....	39-3
Related Features	39-3
Determining Whether to Develop a Custom Transport Provider.....	39-4
When to Use the Transport SDK.....	39-4
When Alternative Approaches are Recommended.....	39-5
Transport Provider Components	39-5
Design-Time Component	39-6
Runtime Component.....	39-8
The Transaction Model	39-9
Overview of Transport Endpoint Properties.....	39-9
Support for Synchronous Transactions	39-10
Transport SDK Security Model	39-12
Inbound Request Authentication	39-12
Outbound Request Authentication	39-13
Link-Level or Connection-Level Credentials	39-14
Uniform Access Control to Proxy Services	39-15
Identity Propagation and Credential Mapping	39-15
Transport SDK and the Threading Model	39-15
Inbound Request Message Thread.....	39-16
Outbound Response Message Thread.....	39-16
Support for Asynchrony.....	39-17
Publish and Service Callout Threading.....	39-17
Designing for Message Content	39-18
Sources and Transformers.....	39-18
Sources and the MessageContext Object	39-19
Built-In Transformations	39-20

40 Developing Custom Transport Providers

Development Road Map.....	40-1
Planning	40-1
Developing	40-2
Packaging and Deploying	40-2
Before You Begin.....	40-2
Basic Development Steps.....	40-3
Step1. Review the Transport Framework Components.....	40-3
Step 2. Create a Directory Structure for Your Transport Project.....	40-4

Step 3. Create an XML Schema File for Transport-Specific Artifacts	40-4
Step 4. Define Transport-Specific Artifacts	40-4
Step 5. Define the TransportProviderConfiguration XMLBean	40-7
Step 6. Implement the Transport Provider User Interface	40-8
Step 7. Implement the Runtime Interfaces.....	40-9
Step 8. Package and Deploy the Transport Provider	40-10
Important Development Topics.....	40-10
Handling Messages.....	40-11
Transforming Messages	40-13
Working with TransportOptions	40-14
Handling Errors.....	40-16
Defining Custom Environment Value Types.....	40-19
Publishing Proxy Services to a UDDI Registry	40-20
When to Implement TransportWLSArtifactDeployer	40-22
Creating Help for Custom Transports.....	40-22
About Custom Transport Online Help	40-23
How to Provide Custom Transport Help in the Console.....	40-23
How to Provide Custom Transport Help in JDeveloper	40-27
Packaging Help for the Transport Plug-in.....	40-28

41 Developing Custom Transport Providers for JDeveloper

Introduction.....	41-1
Services Runtime and Services Configuration	41-1
Offline Methods.....	41-2
Restrictions when Working Offline	41-4
Working Offline with a Remote Server.....	41-4
Bootstrapping Transports in Offline Mode	41-5
Packaging Transports for JDeveloper.....	41-6
Custom Transport Provider Reference for Offline Tools.....	41-6
Working in Different Modes	41-7
TransportProviderFactory.....	41-8
TransportManagerHelper Methods.....	41-8

42 Packaging and Deploying a Custom Transport Provider

Packaging and Deployment Overview.....	42-1
Custom Transport Provider Components	42-1
Custom Transport Provider Resources	42-2
Packaging the Transport Provider	42-2
Transport JAR File Packaging	42-2
Transport EAR File Packaging	42-3
Transport Plug-in Registration for JDeveloper	42-3
Transport Plug-in Installation.....	42-4
Deploying the Transport Provider.....	42-4

Transport Registration.....	42-4
Undeploying a Transport Provider.....	42-5
Deploying to a Cluster	42-5

43 Creating a Sample Socket Transport Provider

Sample Socket Transport Provider Design	43-1
Concepts Illustrated by the Sample	43-1
Basic Architecture of the Sample	43-2
Configuration Properties.....	43-2
Sample Location and Directory Structure	43-3
Building and Deploying the Sample	43-4
How to Set Up the Environment.....	43-5
How to Build the Sample Transport Provider	43-5
How to Deploy the Sample Transport Provider.....	43-5
Registering the Sample Transport Provider With JDeveloper	43-6
Creating a Socket Transport Sample Project.....	43-6
Creating the Project.....	43-7
Creating the Business Service	43-7
Creating the Proxy Service.....	43-8
Creating the Pipeline.....	43-9
Connecting the Proxy Service and Pipeline.....	43-11
Testing the Socket Transport Provider	43-11
Using the Sample Server and Client for Testing.....	43-11
Using the Test Console	43-12

Part VII Sharing Artifacts and Services

44 Importing and Exporting Resources and Configurations

About Importing and Exporting Resources.....	44-1
About Exporting Resources	44-1
About Importing Resources.....	44-2
Importing and Exporting Resources in JDeveloper.....	44-6
How to Export Resources to a Configuration JAR File in Oracle JDeveloper.....	44-6
How to Export Resources to a Server in Oracle JDeveloper	44-7
How to Import Resources in JDeveloper	44-8
Importing and Exporting Resources in the Oracle Service Bus Console	44-9
How to Export Resources to a Configuration JAR File in the Console	44-10
How to Import Resources from a Configuration JAR File in the Console.....	44-10
How to Import Resources from a ZIP File in the Console	44-12
How to Import Resources from a URL in the Console	44-13
Exporting a Service Bus Configuration Offline.....	44-13
About the Export Process.....	44-14
Preparing to Export a Service Bus Configuration.....	44-15

Exporting a Service Bus Configuration Offline.....	44-16
Export Settings File Format, Samples, and Schema	44-18

45 Sharing Data Using the Metadata Services Repository

Service Bus and the MDS Repository	45-1
Managing the MDS Repository	45-2
Sharing Artifacts Using the MDS Repository	45-2
How to Publish Service Bus Artifacts to the MDS Repository	45-2
Consuming Artifacts Stored in the MDS Repository	45-5
How to Consume MDS Repository Artifacts Using the Resource Browser	45-5
How to Add MDS Repository Artifacts to a Service Bus Project	45-7
How to Create a Business Service from a WSDL File in the MDS Repository	45-7
How to Create a Business Service from a WADL File in the MDS Repository	45-9
How to Expose a WSDL File in the MDS Repository as a REST Service	45-10
Opening the Project Overview File Through a SOA-MDS Connection	45-11

46 Working with UDDI Registries

UDDI, UDDI Registries, and Web Services	46-1
Basic Concepts of the UDDI Specification	46-2
Benefits of Using a UDDI Registry with Service Bus	46-2
Introduction to UDDI Entities	46-3
Service Bus and UDDI.....	46-4
UDDI Registry URLs.....	46-4
UDDI Registry Security Configuration.....	46-5
Authentication Configuration and UDDI Registries	46-5
About Publishing Proxy Services to a UDDI Registry.....	46-5
About Importing Services from a UDDI Registry	46-6
Keeping Services Synchronized.....	46-7
Automatic Publishing for Proxy Services	46-7
Automatic Importing of UDDI Services	46-8
Related References.....	46-9
Working with UDDI Registry Resources	46-9
How to View UDDI Registry Resources in the Oracle Service Bus Console.....	46-10
How to Create UDDI Registry Resources.....	46-10
How to Create a UDDI Registry Resource from a JDeveloper UDDI Connection	46-11
How to Edit a UDDI Registry Resource.....	46-12
How to Specify a Default UDDI Registry Resource.....	46-12
How to Delete a UDDI Registry Resource.....	46-13
Sharing UDDI Registry Services in JDeveloper	46-14
How to Create a UDDI Registry Connection in JDeveloper	46-14
How to Create a Business Service from a UDDI Registry Service	46-15
How to Download a Service From a UDDI Registry	46-16
Sharing UDDI Registry Services in the Oracle Service Bus Console.....	46-16

Publishing Proxy Services to a UDDI Registry	46-17
How to Import Resources from a UDDI Registry	46-18
How to Automatically Synchronize Imported Services	46-19
How to Manually Synchronize an Imported Service	46-20
How to Unlink an Imported Service From the UDDI Registry	46-20
Sample Business Scenarios for Service Bus and UDDI	46-21
Basic Proxy Service Communication with a UDDI Registry	46-21
Cross-Domain Deployment in Service Bus	46-21
Mapping Service Bus Proxy Services to UDDI Entities	46-22
UDDI Mapping Details for a Service Bus Proxy Service	46-24
Transport Attributes	46-26
Service Type Attributes	46-28
Canonical tModels Supporting Service Bus Services	46-29
Mapping Example	46-30

Part VIII Security

47 Understanding Oracle Service Bus Security

Inbound Security.....	47-2
Outbound Security	47-3
Options for Identity Propagation	47-3
Using a Service Account with Business Service when Attaching OWSM Policies.....	47-11
Example: Authentication with a User Name Token	47-12
Administrative Security.....	47-13
Access Control Policies	47-13
Configuring Proxy Service Access Control	47-14
Access Control Policy Management	47-14
Configuring the Oracle WebLogic Security Framework: Main Steps.....	47-16
Context Properties Are Passed to Security Providers	47-19
Context Properties for HTTP Transport-Level Authentication	47-19
ContextHandler Properties for Access Control and Custom Authentication	47-20
Additional Transport-Specific Context Properties	47-21
Administrator-Supplied Context Properties for Message-Level Authentication	47-22
Security Provider Must Have Knowledge of the Property Name	47-22
WebLogic Server Administrative Channel is Supported	47-23
Using Security Providers	47-24
Configuring Authentication Providers	47-24
Using a Custom Authorization Provider to Protect Service Bus Resources.....	47-25
About Errors When Using Security Provider Policies	47-28

48 Oracle Service Bus Security FAQ

How are Service Bus and WebLogic Server Security related?	48-1
What is Transport-Level Security?	48-2

What is Web Services Security?	48-2
What is Web Service Policy?	48-2
What are Web Service Policy assertions?	48-2
Are Access Control Policy and Web Service Policy the same?	48-3
What is Web Services Security Pass-Through?	48-3
What is a Web Services Security Active Intermediary?	48-3
What is outbound Web Services Security?.....	48-3
What is SAML?.....	48-3
Is it possible to customize the format of the subject identity in a SAML assertion?.....	48-4
What is the Certificate Lookup And Validation Framework?	48-4
Does Service Bus support identity propagation in a proxy service?.....	48-4
Is single sign-on supported in Service Bus?.....	48-4
Are security errors monitored?.....	48-5
Can I configure security for MBeans?	48-5

49 Securing Business and Proxy Services

Introduction to Policies	49-1
Security and Security Policies for Business and Proxy Services	49-2
Security Policies in Service Bus	49-2
Policy Overrides	49-2
Security Settings	49-3
Global Policies.....	49-3
Service Accounts in Business Services	49-3
Security-Related Validation for Active Proxy Services.....	49-4
Attaching and Configuring Policies in JDeveloper	49-4
How to Attach Oracle Web Services Manager Policies in JDeveloper	49-5
How to Define Override Values for a Policy in JDeveloper	49-7
How to Configure Custom Authentication for Proxy Services in JDeveloper	49-7
How to Specify a Service Key Provider for a Proxy Service in JDeveloper	49-8
How to Specify Web Services Policy Enforcement in JDeveloper	49-9
Attaching and Configuring Policies in the Oracle Service Bus Console.....	49-9
How to Attach Oracle Web Services Manager Policies in the Console	49-10
How to Define Override Values for a Policy in the Console	49-11
How to Configure Custom Authentication for a Proxy Service in the Console	49-12
How to Specify a Service Key Provider for a Proxy Service in the Console.....	49-13
How to Specify Web Services Policy Enforcement in the Console	49-14
Configuring Service Bus Client Access Security	49-14
How To Configure Transport-Level Access Policies	49-14
How to Configure Message-Level Access Policies.....	49-16
How to Add Policy Conditions	49-17
Hiding Personally Identifiable Information in Messages	49-21
How to Hide Personally Identifiable Information	49-22

50 Configuring Message-Level Security for Web Services

About Message-Level Security	50-2
Sample Sequence of Actions in Message-Level Security	50-2
Message-Level Access Control Policies for Proxy Services	50-3
Configuring Proxy Service Message-Level Security	50-3
Creating an Active Intermediary Proxy Service: Main Steps	50-4
Creating a Pass-Through Proxy Service: Main Steps	50-5
Configuring Business Service Message-Level Security: Main Steps	50-6
Using the Service Identity Certificate Extensions	50-7
Publishing Certificate Identity Extension in a Proxy Service Effective WSDL	50-7
Consuming Certificate Identity Extension in a Business Service	50-8
Examples of Custom WS-Policy Statements	50-8
Example: Encrypting Part of the SOAP Body and Header	50-8
Example: Encryption Policy for a Business Service	50-10
Example: Encrypting a Custom SOAP Header	50-11
Example: Signing the Message Body and Headers	50-12
Example: Signing a SOAP Body with SAML Holder-of-Key	50-13
Example: Authenticating, Signing, and Encrypting with SAML Sender Vouches	50-15
Disabling Outbound WS-Security	50-18

51 Configuring Transport-Level Security

Configuring Transport-Level Security for HTTPS	51-1
HTTPS Authentication Levels	51-2
Configuring Inbound HTTPS Security: Main Steps	51-2
Configuring Outbound HTTPS Security: Main Steps	51-3
Configuring Transport-Level Security for HTTP	51-4
Configuring Inbound HTTP Security: Main Steps	51-4
Configuring Outbound HTTP Security: Main Steps	51-5
Using Custom Authentication for Outbound HTTP Security	51-6
Configuring Transport-Level Security for JMS	51-7
Configuring Inbound JMS Transport-Level Security: Main Steps	51-7
Configuring Outbound JMS Transport-Level Security: Main Steps	51-8
Configuring Transport-Level Security for SFTP Transport	51-8
How Two-Way Authentication is Performed	51-9
Use of the known_hosts File	51-9
SFTP Transport Authentication Process	51-10
Configuring Inbound SFTP Transport-Level Security: Main Steps	51-12
Configuring Outbound SFTP Transport-Level Security: Main Steps	51-13
SFTP Security Attributes Preserved During Import	51-15
SFTP Credential Life Cycle	51-15
Email, FTP, and File Transport-Level Security	51-16
Email and FTP Transport-Level Security	51-16

File Transport Security	51-16
Configuring Transport-Level Security for SB Transport	51-16
Configuring SAML Authentication With Service Bus (SB) Transport	51-17
Configuring Transport-Level Security for WS Transport.....	51-17
Reliable Web Services Messaging Defined	51-18
WS Transport Resources Visible in WLS Console	51-18
Use of WS-Policy Files for Web Service Reliable Messaging Configuration	51-18
RM WS-Policy Required Prior to Activation.....	51-19
Async Responses	51-19
Proxy Service Authentication	51-19
Preserving Security Configuration on Import	51-21
Configuring Inbound and Outbound WS Transport-Level Security.....	51-21
Configuring Transport-Level Security for WebSphere Message Queue Transport	51-21
Configuring Inbound MQ Transport-Level Security: Main Steps	51-22
Configuring Outbound MQ Transport-Level Security: Main Steps	51-22
Transport-Level Security Elements in the Message Context.....	51-23

52 Securing Oracle Service Bus with Oracle Web Services Manager

About Oracle Web Services Manager Integration with Oracle Service Bus	52-1
Security Providers	52-2
Using Oracle Web Services Manager with Oracle Service Bus.....	52-3
Attaching Oracle Web Services Manager Policies to Oracle Service Bus Services	52-3
Configuring SAML.....	52-3
Advertising WSDL Files to Support WS Standards	52-4
Deployment Considerations	52-6
Auditing.....	52-6
Monitoring Statistics	52-6
Predefined Policies and Unsupported Assertions.....	52-6
Custom Assertions	52-10

53 Securing Oracle Service Bus Proxy and Business Services with WS-Policy

About Web Services Policy.....	53-1
Relationship Between WS-Security and WS-Policy	53-2
Abstract and Concrete WS-Policy Statements.....	53-2
Oracle-Proprietary Security Policy Best Practices.....	53-3
Policy Subjects and Effective Policy	53-4

54 Using SAML with Oracle Service Bus

Mapping Identity to a SAML Token	54-1
Configuring SAML Pass-Through Identity Propagation.....	54-2
Authenticating SAML Tokens in Proxy Service Requests.....	54-2
Configuring SAML Authentication with Service Bus (SB) Transport	54-3
Using SAML Identity Switching.....	54-3

Protecting the Identity-Switching Resource	54-3
Troubleshooting SAML with Oracle Service Bus.....	54-3
55 Configuring Custom Authentication	
Introduction to Custom Authentication in Oracle Service Bus.....	55-1
Understanding Custom Authentication Tokens.....	55-1
Custom Authentication Token Use and Deployment.....	55-2
Understanding Transport-Level Custom Authentication.....	55-2
Understanding Message-Level Custom Authentication	55-3
Propagating the Identity Obtained From Custom Authentication Tokens	55-4
Combining WS-Security with Custom User Name/Password and Tokens.....	55-4
Format of XPath Expressions	55-4
Configuring Identity Assertion Providers for Custom Tokens	55-5
Object Type of Custom Tokens	55-6
Configuring a Custom Token Type in an Identity Assertion Provider.....	55-7
Configuring Custom Authentication Transport-Level Security	55-8
How to Create a Custom Authentication Class for Outbound	55-8
How to Configure Transport-Level Custom Authentication	55-9
Configuring Message-Level Custom Authentication.....	55-10
How to Configure Message-Level Custom Authentication for Proxy Services.....	55-10
56 Defining Message-Level Security with .Net 2.0	
Message-Level Security Between .NET 2.0 and Oracle Service Bus.....	56-1
What is .NET?.....	56-1
Message-Level Security Configuration in .NET.....	56-1
Oracle Service Bus Configuration for Message-Level Security with .NET	56-3
Sample WSDL File.....	56-5
Part IX Completing Oracle Service Bus Services	
57 Debugging Oracle Service Bus Applications	
Introduction to the Debugger	57-1
Debug Servers	57-1
Local and Remote Debugging	57-2
Debugging With Breakpoints	57-2
JDeveloper Debugging Windows	57-3
Configuring the Project and Debugger.....	57-3
How to Create Run Configuration for Remote Debugging	57-3
How to Choose a Run Configuration for Debugging	57-4
Accessing the Debugger	57-4
Debugging a Service Bus Application	57-5
How to Set Breakpoints on Service Bus Components.....	57-5
How to Debug Using Breakpoints.....	57-5

How to Step Through a Debugging Session	57-6
How to End or Detach from Debugging.....	57-7
Working with the Debugger Windows	57-7
How to Edit Breakpoint Options	57-7
How to Create a Breakpoint Group.....	57-8
How to Remove or Disable Breakpoints.....	57-8
How to Enable a Disabled Breakpoint	57-9
How to View and Modify Variable Values at the Current Breakpoint.....	57-9
How to Add a Watch	57-9

58 Using the Test Console

Introduction to the Test Console	58-1
Proxy Service Testing.....	58-2
Pipeline Testing	58-2
Business Service Testing.....	58-3
Recommended Approaches to Testing Services.....	58-4
HTTP Requests	58-5
Accessing the Test Console	58-5
Prerequisites.....	58-5
How to Access the Test Console from the Oracle Service Bus Console	58-6
How to Access the Test Console from Fusion Middleware Control	58-6
How to Access the Test Console from JDeveloper.....	58-7
Testing Proxy Services, Business Services, Pipelines, and Split-Joins	58-8
How to Test Service Bus Services.....	58-8
How to Test Attachments in Services.....	58-10
How To Trace Pipeline Processing	58-11
How to View Service Test Results	58-12
Testing MFL Transformations	58-12
How to Test MFL Transformations in the Test Console.....	58-13
MFL Test Console Example	58-13
Testing XSLT Transformations (Resources).....	58-14
How to Test XSLT Transformations Using the Test Console	58-14
How to Test XSLT Transformations Using the JDeveloper XSLT Mapper.....	58-15
Testing XQuery Transformations (Resources)	58-15
XQuery Transformation Testing Prerequisites and Guidelines	58-16
How to Test XQuery Transformations in the Test Console	58-16
Testing Inline Expressions.....	58-16
How to Test XQuery Expressions	58-17
How to Test XPath Expressions	58-17
Testing Services With OWSM Security.....	58-18
Limitations for Services and Policies.....	58-20
About Security and Transports.....	58-21
Undeploying the Test Console.....	58-21

Untargeting the Test Console Before Domain Creation	58-21
Untargeting the Test Console when the Server is Running	58-22
Untargeting the Test Console when the Server is Not Running	58-22
Test Console Page Reference for Services	58-23
Test Configuration Test Console Properties.....	58-23
Service Operation Test Console Properties	58-23
Request Document Test Console Properties	58-24
Security Test Console Properties.....	58-26
Authentication Test Console Properties	58-26
Transport Test Console Properties.....	58-27
Attachment Test Console Properties	58-30
59 Deploying Oracle Service Bus Services	
Deployment Overview.....	59-1
Before You Deploy.....	59-2
Creating a Service Bus Domain Using the Configuration Wizard.....	59-2
Resolving Conflicts.....	59-2
Configuring JMS Resources	59-2
Configuring Security.....	59-2
Deploying from the Oracle Service Bus Console	59-3
How to Deploy from the Console	59-3
Deploying from JDeveloper	59-3
How to Create a Connection to the WebLogic Server	59-4
How to Create a Deployment Profile	59-5
How to Deploy a Service Bus Project or Application	59-6
How to Deploy a Project or Application Using the Previous Configuration.....	59-6
What Happens When You Deploy Using JDeveloper	59-7
Deploying a Service Bus Configuration JAR File in Fusion Middleware Control.....	59-7
Updating an Online Configuration.....	59-7
What You Need to Know for Successful Online Configuration Updates.....	59-9
Changing an Online Business Service	59-9
Changing an Online Proxy Service.....	59-9
Changing an Online Pipeline.....	59-10
Updating an Online Configuration in a Cluster	59-10
Changing a Business Service in a Cluster	59-10
Installing a New Version of a Proxy Service in a Cluster.....	59-10
60 Using the Oracle Service Bus Development Maven Plug-In	
Introduction to the Oracle Service Bus Maven Plug-In.....	60-1
Maven Lifecycle Phases and Goals	60-1
POM Files and Archetypes	60-2
Installing and Configuring Maven.....	60-2
How to Configure the Oracle Service Bus Development Maven Plug-In.....	60-3

How to Use Maven Online Help.....	60-4
Using the Oracle Service Bus Development Maven Plug-In	60-4
How to Generate a Service Bus Project POM File	60-4
How to Generate a Service Bus Project POM File from an Archetype.....	60-4
How to Generate a Service Bus System Resources POM File from an Archetype	60-6
Parameters for Generating a POM File	60-7
Service Bus Development Maven Plug-In Goals	60-8
package	60-8
deploy.....	60-9
Oracle Service Bus Development Maven Plug-In POM File Samples	60-10

Part X Appendixes

A Message Context

The Message Context Model.....	A-1
Predefined Context Variables	A-1
Message-Related Variables.....	A-2
Inbound and Outbound Variables	A-14
Operation Variable	A-20
Fault Variable	A-21
messageID Variable.....	A-23
Initializing Context Variables	A-23
Performing Operations on Context Variables	A-25
Constructing Messages to Dispatch.....	A-26
Message Context Schema	A-28
Errors Schema.....	A-33

B XPath Extension Functions

Cross-Reference Functions	B-1
Domain Value Map Functions	B-7
Creating Custom XPath Functions.....	B-9

C Oracle Service Bus APIs

Resource Update and Customization	C-1
Management and Monitoring.....	C-2
Deployment	C-2

D Transport SDK Interfaces and Classes

Introduction.....	D-1
Schema-Generated Interfaces.....	D-1
General Classes and Interfaces	D-2
Source and Transformer Classes and Interfaces	D-4
Metadata and Header Representation for Request and Response Messages.....	D-6

User Interface Configuration	D-7
E Transport SDK UML Sequence Diagrams	
Service Bus Runtime Inbound Messages	E-1
Service Bus Runtime Outbound Messages	E-2
Design Time Service Registration.....	E-3
F XQuery-SQL Mapping Reference	
IBM DB2/NT 8	F-1
Microsoft SQL Server	F-2
Oracle8i, 8.1.x	F-3
Oracle 9i and Later.....	F-4
Sybase 12.5.2 (and higher)	F-5
Base (Generic) RDBMS Data Type Mapping	F-7
G Work Managers and Threading	
Key Threading Concepts	G-1
Pipeline Actions	G-2
Work Managers.....	G-2
Designating Work Managers	G-3

List of Tables

1-1	Service Bus Development Roadmap - Top-Down Approach.....	1-26
1-2	Service Bus Development Roadmap - Bottom-Up Approach.....	1-26
3-1	Restrictions on Naming a Service Bus Project.....	3-7
5-1	Service Bus Overview Editor.....	5-3
5-2	Technology Components.....	5-5
5-3	Application Components.....	5-8
5-4	Advanced Components.....	5-8
5-5	Starting Service Bus Editors from the Service Bus Overview Editor.....	5-18
7-1	JBoss Initial Context Factory Environment Parameters.....	7-3
10-1	Split-Join Communication Operations.....	10-4
10-2	Split-Join Flow Control Operations.....	10-4
10-3	Split-Join Assign Operations.....	10-6
11-1	High-level WSDL Elements.....	11-1
12-1	Pipeline Components.....	12-2
12-2	Path of a Request Message During a Message Flow.....	12-3
12-3	Path of a Response Message During a Message Flow.....	12-4
12-4	Communication Actions.....	12-6
12-5	Flow Control Actions.....	12-6
12-6	Message Processing Actions.....	12-7
12-7	Reporting Actions.....	12-9
12-8	Limitations to Transport Header Values You Specify in Transport Header Actions...	12-11
12-9	Scope of Error Handlers.....	12-30
12-10	Predefined Context Variables in Service Bus	12-39
12-11	Delivery Guarantee Types.....	12-46
12-12	Delivery Guarantee Rules.....	12-48
12-13	Service Bus WS-I Compliance Checks.....	12-54
13-1	Edit Message Flow Page Icons and Options.....	13-2
13-2	Standard Resequencing Options.....	13-13
13-3	Best Effort Resequencing Options.....	13-14
14-1	Pipeline - Communication Actions.....	14-3
14-2	Pipeline - Flow Control Actions.....	14-4
14-3	Pipeline - Message Processing Actions.....	14-4
14-4	Pipeline - Reporting Actions.....	14-5
14-5	Edit Stage Configuration Tasks.....	14-6
14-6	Service Callout Configuration Options.....	14-12
14-7	Log Action Severity Levels.....	14-32
14-8	Viewing and Changing the Error Handler.....	14-36
15-1	Palettes.....	15-5
15-2	Ways to Paste Items Into the Editor Text Fields.....	15-6
15-3	Create a New Variable Structure.....	15-9
16-1	Standard Resequencing Options.....	16-14
16-2	Best Effort Resequencing Options.....	16-15
17-1	Pipeline - Communication Actions.....	17-3
17-2	Pipeline - Flow Control Actions.....	17-4
17-3	Pipeline - Message Processing Actions.....	17-4
17-4	Pipeline - Reporting Actions.....	17-5
17-5	Edit Stage Configuration Tasks.....	17-6
17-6	Log Action Severity Levels.....	17-34
23-1	Character Delimiters.....	23-2
23-2	Find Options.....	23-16
23-3	Goto Options.....	23-17

23-4	Supported MFL Data Types	23-21
23-5	COBOL Data Types.....	23-26
23-6	Format Builder Field Description Properties.....	23-31
23-7	Format Builder Field Occurrence Properties.....	23-32
23-8	Format Builder Field Attributes.....	23-32
23-9	Format Builder Field Termination Properties Defined by Length.....	23-33
23-10	Format Builder Field Termination Properties Defined by Imbedded Length.....	23-33
23-11	Format Builder Field Termination Properties Defined by Delimiter.....	23-34
23-12	Format Builder Field Properties for Literal Data Types.....	23-35
23-13	Format Builder Group Description Properties.....	23-35
23-14	Format Builder Group Occurrence Properties.....	23-35
23-15	Format Builder Group Attributes.....	23-36
23-16	Format Builder Group Delimiter Properties.....	23-36
23-17	Format Builder Reference Description Properties.....	23-37
23-18	Format Builder Reference Occurrence Properties.....	23-37
25-1	JCA Transport Endpoint Properties.....	25-18
25-2	JCA Transport Properties.....	25-20
27-1	DSP Transport Configuration Properties for Business Services.....	27-6
28-1	EJB Transport Properties for Business Services.....	28-9
29-1	HTTP Transport Properties for Proxy Services.....	29-3
29-2	HTTP Transport Properties for Business Services.....	29-5
29-3	Response Code Handling for HTTP Business Services.....	29-11
29-4	Email Transport Properties for Proxy Services.....	29-13
29-5	Email Transport Properties for Business Services.....	29-14
29-6	File Transport Properties for Proxy Services.....	29-16
29-7	File Transport Properties for Business Services.....	29-18
29-8	FTP Transport Properties for Proxy Services.....	29-19
29-9	FTP Transport Properties for Business Service.....	29-21
29-10	Properties Imported from UDDI Registry.....	29-28
29-11	SFTP Transport Properties for Proxy Services.....	29-29
29-12	Transport Headers and Metadata.....	29-32
29-13	SFTP Transport Properties for Business Services.....	29-33
29-14	SFTP Transport Environment Values.....	29-34
30-1	JEJB Transport Properties for Proxy Services.....	30-10
30-2	JEJB Transport Configuration for Business Services.....	30-11
31-1	Differences Between Message ID and Correlation ID Patterns.....	31-14
31-2	JMS Transport Properties for Proxy Services.....	31-20
31-3	JMS Transport Headers.....	31-23
31-4	JMS Transport Properties for Business Services.....	31-26
33-1	MQ Transport Properties for Proxy Services.....	33-10
33-2	MQ Transport Properties for Business Services.....	33-13
33-3	MQ Transport and Connection Resource Environment Values.....	33-16
33-4	MQ Transport Headers.....	33-17
34-1	BPEL Transport Environment Variables.....	34-3
34-2	Specifying an Endpoint URI.....	34-18
34-3	BPEL Transport Properties for Business Services.....	34-19
35-1	SB Transport Environment Variables.....	35-4
35-2	SB Transport Properties for Proxy Services	35-5
35-3	SB Transport Properties for Business Services.....	35-6
36-1	SOA-DIRECT Transport Properties for Business Services.....	36-12
36-2	SOA-DIRECT Transport Environment Values.....	36-14
37-1	Tuxedo Exceptions	37-6
37-2	Buffer Transformation for Any XML Service.....	37-8
37-3	Buffer Transformation for Messaging Service.....	37-9

37-4	Tuxedo Transport Properties for Proxy Services.....	37-10
37-5	Tuxedo Transport Properties for Business Services.....	37-12
38-1	WS Transport Properties for Business Services.....	38-8
38-2	WS Transport Properties for Proxy Services.....	38-9
39-1	Built-In Transformations.....	39-21
43-1	Key Sample Transport Provider Directories.....	43-4
46-1	High-Level Description of UDDI Entities	46-3
46-2	Proxy Service Attributes and Service Types.....	46-22
46-3	Transport Attributes.....	46-27
46-4	Service Type Attributes.....	46-28
46-5	CategorizationGroup tModel Types.....	46-29
46-6	Categorization tModel Types.....	46-29
46-7	Transport tModel Types.....	46-30
46-8	Protocol tModel Types.....	46-30
47-1	Options for Identity Propagation.....	47-4
47-2	Combinations of Transport-Level Security Requirements.....	47-6
47-3	Combinations of Message-Level Security Requirements.....	47-8
47-4	Authentication Providers.....	47-16
47-5	ContextHandler Properties for HTTP Transport Authentication.....	47-20
47-6	ContextHandler Properties for Custom Authentication and Access Control.....	47-20
47-7	Additional ContextHandler Properties for HTTP Proxy Services.....	47-21
49-1	Supported Policy Categories.....	49-1
49-2	Condition Predicate Options.....	49-17
51-1	WS Transport Authentication Matrix.....	51-20
52-1	Valid and Invalid Combinations of the <code>&wsp</code> and <code>&wssp</code> Query Parameters.....	52-5
52-2	Supported OWSM Predefined Policies for WSDL (non-SOAP), XML, and Messaging Service Service Types with HTTP Transport.....	52-7
52-3	Unsupported assertions.....	52-10
58-1	Testing Results for Proxy Services.....	58-12
58-2	MFL Test Console Properties.....	58-13
58-3	Digital Signature and Encryption Scenarios.....	58-19
58-4	Identity Policy Scenarios (Assuming that the Policy has an Identity Assertion).....	58-20
58-5	Test Console Properties- Service Operation.....	58-24
58-6	Test Console Properties - Request Document.....	58-24
58-7	Test Console Properties - Security.....	58-26
58-8	Test Console Properties - Authentication.....	58-27
58-9	Test Console Properties - Transport.....	58-29
58-10	Limitations to Transport Header and Metadata Values You Specify in the Test Console.....	58-29
58-11	Proxy Service Test Console Properties.....	58-30
59-1	Initial and Updated Configuration for a Sample System.....	59-8
60-1	Maven Lifecycle Phases for Service Bus.....	60-2
60-2	Parameters for <code>servicebus:package</code> Goal.....	60-8
60-3	Parameters for <code>servicebus:deploy</code> Goal.....	60-10
A-1	Predefined Context Variables in Service Bus	A-2
A-2	Sub-Elements of the Attachments Variable	A-4
A-3	Message Mappings.....	A-5
A-4	Sub-Elements of the service Element	A-15
A-5	Sub-Elements of the Transport Element	A-16
A-6	Sub-Elements of the Security Element	A-20
A-7	Sub-Elements of the Fault Variable	A-21
A-8	Initializing Context Variables.....	A-23
B-1	Custom Function Registration File Properties.....	B-10
B-2	Supported Java Method Types for Custom Functions.....	B-11

F-1	IBM DB2 Data Type Mappings.....	F-1
F-2	SQL Server 2000 Data Type Mapping.....	F-2
F-3	Oracle 8.1.x Data Type Mapping.....	F-3
F-4	Oracle 9i and later Data Type Mapping.....	F-4
F-5	Sybase 12.5.2 Data Type Mapping.....	F-6
F-6	RDBMS Data Type Mapping.....	F-7

Preface

This preface describes the conventions of this guide--*Developing Services with Oracle Service Bus*.

Documentation Accessibility

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

Access to Oracle Support

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

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.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in This Guide

This preface introduces the new and changed features of Oracle Service Bus and other significant changes that are described in this guide, and provides pointers to additional information. This document is the new edition of the formerly titled *Oracle Fusion Middleware Developer's Guide for Oracle Service Bus*.

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

Significant Documentation Changes for 12c (12.1.3)

For this release, both the developer's guide and the administrator's guide were changed significantly. All of the development information that was previously in both books was combined, streamlined, and moved into this guide, *Developing Services with Oracle Service Bus*. *Administering Oracle Service Bus* was rewritten and only includes tasks you can perform from Oracle Enterprise Manager Fusion Middleware Control and the administrative tasks available on the Oracle Service Bus Console.

New and Changed Features for 12c (12.1.3)

For Oracle Service Bus 12c (12.1.3), this guide has been updated to include the following new and changed development features:

- New editors in Oracle JDeveloper and the new Oracle Service Bus Console, which are described through this book. Support for JDeveloper is new. See [Getting Started with Oracle Service Bus in JDeveloper](#). The web-based console has been redesigned. See [Getting Started with the Oracle Service Bus Console](#).
- The monitoring and runtime management features that were previously in the console are now provided in Oracle Enterprise Manager Fusion Middleware Control. See *Administering Oracle Service Bus*.
- A new Overview Editor in JDeveloper for developing Service Bus services. Similar to the Composite Editor for Oracle SOA Suite, it lets you drag and drop components on to the editor to create services. The editor provides direct creation and editing of JCA adapters, REST bindings, and Service Bus services. See [Developing Oracle Service Bus Applications in JDeveloper](#).
- Support for REST services through the REST binding. See [Creating REST Services with Oracle Service Bus](#).

- Support for WebLogic Server's new Coherence container for business service result caching. This changes the way Coherence caches are configured and the way out-of-process Coherence caches are set up. See [Improving Performance by Caching Business Service Results](#).
- Pipelines are now a separate component from proxy services, and can be created and configured separately. See [Working with Oracle Service Bus Pipelines](#). Additional pipeline enhancements include the following:
 - Resequencing messages using the same resequencer engine as Oracle Mediator. See [Configuring the Resequencer in the Console](#) and [Configuring the Resequencer in JDeveloper](#).
 - Creating pipeline templates on which to base new pipelines, allowing you to re-use your already developed message processing logic. See [Working with Pipeline Templates](#).
 - Sharing variables among all pipelines in a chain, which allows them to read and modify the same variable. See [How to Add Shared Variables to Pipelines in the Console](#) and [Adding Shared Variables to Pipelines in JDeveloper](#).
 - Sending and receiving attachments with the service callout action. See [Using Attachments with Service Callout Messages](#).
 - Using nXSD transformations in a message flow. See [Adding nXSD Translate Actions](#) and [Adding nXSD Translate Actions in JDeveloper](#).
 - Enabling and disabling a stage, which effectively allows you to skip certain actions. See [Disabling an Action or a Stage in the Console](#) and [Disabling an Action or a Stage in JDeveloper](#).
- Split-joins are a separate component, and can be created and configured separately. See [Improving Service Performance with Split-Join](#).
- Support for cross-reference tables and domain value maps, both features from Oracle SOA Suite. See [Mapping Data with Cross-References](#) and [Mapping Data with Domain Value Maps](#).
- Support for Oracle SOA Suite XSLT mappings. See [Transforming Data with XSLT](#). Service Bus also supports dynamic XSLT expressions. See [Binding Dynamic XSLT Expressions to Inline XQueries](#).
- Support for all Oracle Web Services Manager (OWSM) policies. Certain policies can also be attached to JCA services and to REST services. See [Securing Oracle Service Bus with Oracle Web Services Manager](#). More information is available in "Determining Which Predefined Policies to Use" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Support for WebLogic Server security policies is deprecated in this release. You can import services that use WLS9 policies from previous versions of Service Bus, and they will work in the runtime. However, WLS9 policies cannot be attached to new services, and Oracle recommends upgrading all policies to OWSM.
- Support for credential mapping for business services with OWSM policies through the association of a service account resource. This allows service account overrides for outbound ID propagation policies.

- Support for JDeveloper debugging tools, including setting breakpoints and stepping through message flows. The debugger is integrated with the Service Bus Test Console. See [Debugging Oracle Service Bus Applications](#) .
- Sharing artifacts with Oracle SOA Suite using the Oracle Metadata Services (MDS) Repository in JDeveloper. See [Sharing Data Using the Metadata Services Repository](#). Artifacts can also be shared from the application server, from the file system, and UDDI registries.
- Importing and exporting resources and configurations from both JDeveloper and the Oracle Service Bus Console to either a configuration JAR file or to a WebLogic Server. See [Importing and Exporting Resources and Configurations](#) .
- Support for the Maven project and build management system. See [Using the Oracle Service Bus Development Maven Plug-In](#).
- Several enhancements were made to Service Bus transports, including the following:
 - Support for all JCA adapters, as well as most JCA features, including streaming large messages. See [Using the JCA Transport and JCA Adapters](#).
 - Support for JBoss with the EJB transport. See [Working with JNDI Provider Resources](#).
 - Support for custom authorization for HTTP business services. See [Configuring Business Services to Use the HTTP Transport](#) and [Using Custom Authentication for Outbound HTTP Security](#).
 - Support for retries for non-XA connection factories and for reading the replyTo header in the JMS transport. See [Using the JMS Transport](#), specifically the configuration reference and [Access to the JMSReplyTo Property](#).
 - New JavaScript resource to support the JCA Adapter for Sockets. See [Working with JavaScript Resources](#).
- The ability to log malformed XML payloads in the `$fault` variable through a new `PayloadDetail` element. See [Fault Variable](#).
- Support for custom MIME headers in the `$attachments` variable. See [Table A-2](#).

Part I

Introduction to Oracle Service Bus

This part provides an overview of Oracle Service Bus and the two interfaces you can use to create Service Bus services. It also describes how to set up a Service Bus environment and create Service Bus services using the Service Bus Overview Editor in Oracle JDeveloper.

This part includes the following chapters:

- [Learning About Oracle Service Bus](#)
- [Getting Started with the Oracle Service Bus Console](#)
- [Getting Started with Oracle Service Bus in JDeveloper](#)
- [Setting up the Development Environment for JDeveloper](#)
- [Developing Oracle Service Bus Applications in JDeveloper](#)

Learning About Oracle Service Bus

This chapter provides an overview of Service Bus, its architecture and components, and how to use Service Bus to develop services. It also provides roadmaps for developing Service Bus applications and descriptions of different development approaches.

This chapter includes the following topics:

- [Oracle Service Bus Overview](#)
- [Service Bus Architectural Concepts](#)
- [Service Bus Components](#)
- [Service Bus Messaging](#)
- [Using Work Managers with Service Bus](#)
- [Service Bus Security](#)
- [Approaches for Designing Service Bus Services](#)
- [Naming Guidelines for Service Bus Components](#)
- [Viewing Service Bus Resources in a Web Browser](#)
- [Accessibility Options](#)
- [Additional Resources](#)

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

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

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

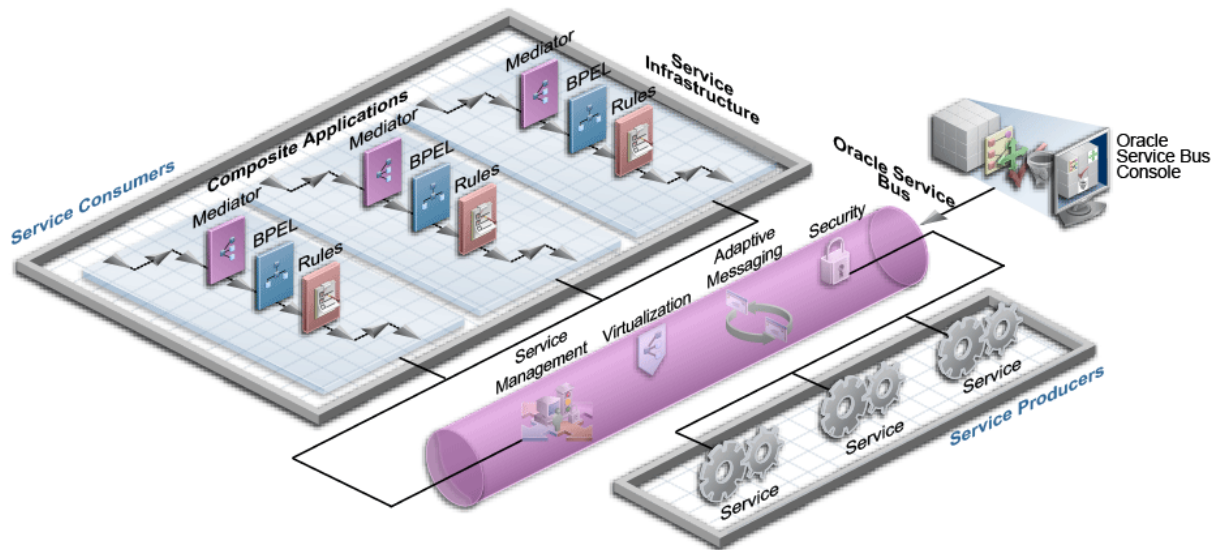
Oracle Service Bus Overview

Oracle Service Bus is a configuration-based, policy-driven enterprise service bus designed for SOA life cycle management. It provides foundation capabilities for service discovery and intermediation, rapid service provisioning and deployment, and governance. Service Bus provides scalable and reliable service-oriented integration, service management, and traditional message brokering across heterogeneous environments. It combines intelligent message brokering with routing and transformation of messages, along with service monitoring and administration. Service Bus leverages industry standards to connect services and support a high level

of heterogeneity, connecting your existing middleware, applications, and data sources, and protecting existing investments.

Service Bus adheres to the SOA principles of building coarse-grained, loosely coupled, and standards-based services, creating a neutral container in which business functions can connect service consumers and back-end business services, regardless of the underlying infrastructure. The following figure illustrates the role of Service Bus as a service intermediary in an enterprise IT SOA landscape.

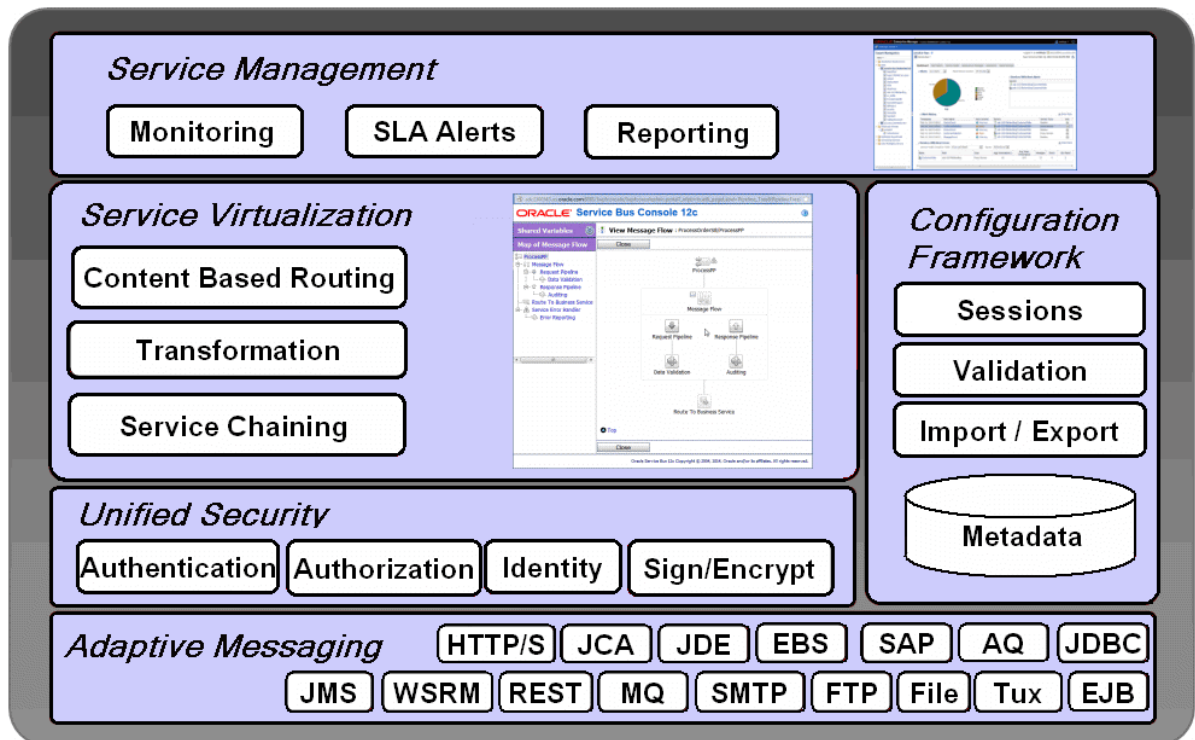
Figure 1-1 Service Bus Intermediary



Functional Areas

The following diagram illustrates the primary functional areas of Service Bus, including virtualization, messaging, security, configuration, and runtime management.

Figure 1-2 Service Bus Functional Features



Adaptive Messaging

Adaptive messaging provides flexible message handling and manipulation between clients and services. For example, a client sends a SOAP message over HTTP through Service Bus, which in turn transforms the message and invokes a back-end EJB. Or a client sends a REST/JSON message over HTTP, and Service Bus transforms the message and invokes a back-end SOAP/XML service (or uses any of the available adapters). Adaptive messaging also supports a variety of communication patterns such as request/response, synchronous and asynchronous, split-join, and publish/subscribe. It supports different patterns for inbound and outbound messages in a single message life cycle.

Service Security

Service Bus ensures service security at all levels, based on Oracle Platform Security Services and Oracle Web Services Manager (OWSM) for web services. You can plug in custom or third-party security components. Built-in capabilities allow flexibility in implementation by enabling security at the following levels:

- Transport-level security, including SSL, basic authorization, and custom security credentials
- Message-level security, including WS-Security, SAML, user ID and password, X509, signing and encryption, and custom security credentials
- Console security, including single-sign-on and role-based access
- Policy security

Service Virtualization

Service virtualization provides agility through message manipulation and control. Service Bus lets you flexibly control messages using validation, transformation, routing based on message content, parallel processing of multiple items in a message, alert triggering, and error handling at different points in a message flow. For example, Service Bus provides the following capabilities:

- XQuery-based policies or callouts to external services for message routing.
- Routing policies that apply to both point-to-point and one-to-many routing scenarios (publish). For publish, routing policies serve as subscription filters.
- Routing table abstracted from pipelines, which enables modification of routes without having to reconfigure pipelines.
- Identity-based routing, to classify clients into user-defined groups and apply routing policies based on these groups.
- Conditional routing, including dynamic content-based routing of messages and runtime protocol selection.
- Database lookups, which can be used for message enrichment, routing decisions, or customizing the behavior of a pipeline.
- Transformations using XQuery or XSLT maps.

Configuration Framework

The Configuration Framework gives you full control over your Service Bus production environment and its associated resources. The framework includes session management, the Test Console, and import/export tools. Service Bus configurations are managed in sessions, which provide the unique ability to lock the current configuration while changes are being made. Service Bus can continue to receive and process requests for services while configuration changes are being made in a session. These changes do not affect the runtime configuration until you activate the current session. This way, ongoing changes can be made without disrupting services. Configuration and resource changes you make are tracked, and you can undo or redo changes, resolve conflicts, maintain dependencies among resources, and test changes in the Test Console.

The built-in Test Console is a browser-based test environment used to validate resources as well as inline expressions used in pipelines or split-joins. Use the Test Console to configure the test object (such as a pipeline, business service, or XQuery expression), execute the test, and view test results. It allows message flow tracing when testing a service, to examine the state of the message at specific trace points.

Service Bus allows the propagation of configuration data from environment to environment by exporting and importing resources and projects. For example, you can transfer configurations from a development domain to a test domain to a production domain. The import and export features let you maintain resource dependencies and preserve environment values between environments.

The Configuration Framework also includes a metadata-driven interface for service discovery, publishing, and synchronization using UDDI registries, including automatic import and synchronization of services with UDDI.

Service Management

Service management includes a powerful set of runtime configuration tools for monitoring, alerting, and reporting. Service Bus is fully integrated with Fusion Middleware Control for SOA-wide service management. Service management lets you do the following:

- Gather statistics about message invocations, errors, performance characteristics, messages passed, and SLA violations.
- Send SLA and pipeline alerts as SNMP traps, enabling integration with third-party enterprise system management solutions.
- Log selected parts of messages for both systems operations and business auditing purposes.
- Search message reports by extracting key information from a message, which can then be used as a search index.
- Integrate with widely adopted third-party reporting tools as well as custom enterprise system management frameworks.
- Support open interfaces for operational and deployment customization, JMX monitoring interfaces, and SNMP Alerts.

Service Bus Architectural Concepts

Service Bus is an intermediary that processes incoming service request messages, determines routing logic, and transforms those messages for compatibility with other service consumers. It receives messages through a transport protocol such as HTTP(S), JMS, File, or FTP, and sends messages through the same or a different transport protocol. Service response messages follow the inverse path. Message processing by Service Bus is driven by metadata, specified in the message flow definition (pipeline).

Service Bus provides message delivery services based on standards including SOAP, HTTP, and Java Messaging Service (JMS). It supports XML as a native data type, while also offering alternatives for handling other data types. Service Bus lets you establish loose coupling between service clients and business services, while maintaining a centralized point of security control and monitoring. It stores persistent policy, service, and related resource configurations in metadata, which can be customized and propagated from development through staging to production environments. The message-brokering engine accesses this configuration information from its metadata cache.

Message Processing

Messages can contain data or status information about application processes, as well as instructions for the recipient. Service Bus lets you route messages based on their contents and perform transformations on that content. The processing happens through the transport and binding layers of Service Bus.

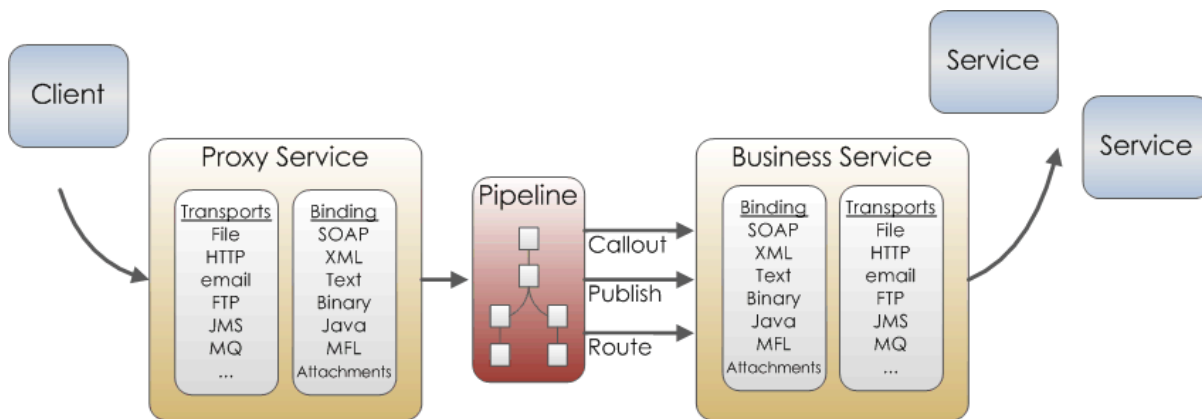
The processing of messages through Service Bus occurs in the following sequence of events:

1. A client sends a message to Service Bus using a specific transport protocol.

2. A transport provider processes the inbound message, handling communication with the service client endpoint and acting as the entry point for messages into Service Bus.
3. The binding layer packs and unpacks messages, handles message security, and hands messages off to the pipeline.
4. The pipeline performs any transformation, validation, logging, and reporting, and then routes the message to an endpoint (either a business service or another proxy service).
5. Service Bus processes the response message in a similar manner as the above steps.

The following figure illustrates the flow of data through Service Bus, from inbound endpoint (proxy service) to outbound endpoint (in this case, a business service). The transports listed are a subset of those available through Service Bus.

Figure 1-3 Oracle Service Bus Message Processing



The following sections describe each layer involved in this message processing.

Proxy Service Role in Message Processing

Proxy services are the interfaces that service consumers use to connect with managed back-end services. Proxy services are definitions of intermediary web services that Service Bus implements locally. The proxy service interface is defined in terms of Web Services Description Language (WSDL) or Web Application Definition Language (WADL) and the type of transport it uses.

Transport Layer (Inbound)

The inbound transport layer is the communication layer between client services (or service consumers) and Service Bus. It is responsible for handling communication with the service client endpoint and acts as the entry point for messages into Service Bus. The inbound transport layer primarily deals with raw bytes of message data in the form of input/output streams. The transport layer provides support for compatible transport protocols, including HTTP(S), JMS, FTP, File, email, and others. It is not involved in data processing but is responsible for returning response messages to service consumers and handles metadata for messages, including endpoint URIs, transport headers, and so on.

Binding Layer

The binding layer for both inbound and outbound performs the following functions in message processing:

- Packs and unpacks messages as necessary
- Handles security for messages
- Hands messages off to start the pipeline (request and response)

Pipeline Role in Message Processing

A pipeline defines the flow of request and response messages through Service Bus, including routing, transformations, validations, publishing, reporting and exception management. It accepts messages from the binding layer of the proxy service, performs any transformations or validations, and then forwards the message to the binding layer of the outbound service, either a proxy or business service. Along the way, the pipeline can make callouts to other services, Java objects, or POJOs.

Transport Layer (Outbound)

The outbound transport layer is responsible for the communication between external services (or service producers) and Service Bus. It is responsible for moving messages from Service Bus to the business service or proxy service and for receiving the response from the services. At the transport level, the message data are in raw bytes in the form of input/output streams. The outbound transport layer provides support for compatible transport protocols, including HTTP(S), JMS, FTP, File, email, and others. It is not involved in data processing but handles metadata for messages, including endpoint URIs, transport headers, and so on.

Business Service Role in Message Processing

Business services are the interfaces that connect with service producers. The business service interface is defined in terms of Web Services Description Language (WSDL) or Web Application Definition Language (WADL) and the type of transport it uses.

Service Bus Components

Service Bus routes message between external services (such as enterprise services and databases) and service clients (such as presentation applications or other business services). The service and flow components you create in a Service Bus project rely on local and system resources in Service Bus to define additional information like user names and passwords, keystore credentials, server connections, and transformations.

Service Bus resources are reusable definitions of entities that typically include metadata for those entities. Resources can be used by multiple services and provide standardized definitions or descriptions for use across an enterprise or department.

Service Components

Proxy services and business services define the endpoints in a Service Bus system. They include the binding and transport layers, and are the points at which Service Bus communicates with external services, including producers and consumers.

Proxy Services

Proxy services are Service Bus definitions of generic intermediary web services that are hosted locally on Service Bus. A proxy service communicates with external services through interfaces, which may or may not be identical to that of a service provider or service consumer business service. Through pipelines, you can route messages from a proxy service to multiple business services using their configured independent interfaces.

A proxy service's configuration includes its interface (service type), the type and configuration of the transport it uses to connect with client services, security requirements, and service level agreement (SLA) alert rules. When a proxy service interfaces with multiple business services, its associated pipeline is configured to route messages to the appropriate business service and map the message data into the format required by the business service's interface.

For more information, see [Creating and Configuring Proxy Services](#).

Business Services

Business services are Service Bus definitions of the enterprise services that exchange messages during business processes. A business service's configuration includes its interface (service type), the type and configuration of transport it uses to connect with service producers, security requirements, message handling, performance tuning, and SLA alert rules. A business service also specifies the endpoint URI, and can specify multiple endpoints for load balancing and high availability. A business service definition is similar to that of a proxy service, but with additional options for message handling, endpoint throttling, and result caching, which help improve performance.

You can create a service account to provide authentication when connecting to a business service. It acts as an alias resource for the required user name and password pair. You can also use Oracle WebLogic Server to directly manage security credentials for a business service requiring credential-level validation.

For more information, see [Creating and Configuring Business Services](#).

Message Flows

A message flow defines how messages are routed, validated, and transformed between services. The message flow is typically defined in a pipeline, but can also be defined in a split-join for parallel processing.

Pipelines

Pipelines define message routing and transformation logic, as well as message handling options. This logic includes activities such as transformation, publishing, logging, reporting, alerts, and exception management. Each of these activities are configured as individual actions within the message flow. Both JDeveloper and the Oracle Service Bus Console provide graphical modeling tools to help you model your pipelines.

The following primary elements are used to construct a pipeline:

- A *start node*.
- A *pipeline pair*, one for the request and one for the response. Each pipeline in a pair consists of a sequence of stages that specify actions to perform during request or response processing.

- A *branch node*, to branch based on the values in designated parts of the message or message context, or to branch based on the operation invoked.
- A *route node*, to define the message destination. The default route node is an *echo node* that reflects the request as the response.
- An *error handler*, which can be attached to any node or stage to handle potential errors at that location.

At a minimum, a start node and route node are required. While an error handler is not required, it is recommended. If an instance fails and no error handler is defined, the error is not recoverable. Pipeline elements can be combined in arbitrary ways to form a tree structure with the start node always (and only) occurring as the root of the tree and the route nodes. The last nodes in a branch (leaf nodes) can be route nodes or echo nodes.

Since a pipeline can route messages to multiple business services, a pipeline can be configured with an interface that is independent of the business services it communicates with. Using generic templates, the pipeline can be configured to dynamically route messages to appropriate business services based on content-based routing logic. A pipeline can also map message data into appropriate protocol formats required by the end-point business service, allowing for dynamic runtime protocol switching.

For more information, see [Working with Oracle Service Bus Pipelines](#)

How Data Flows Through a Pipeline

In a pipeline, the request message starts at the start node and follows a path to a leaf node, executing actions in the request pipelines. If the leaf is a route node, a response is generated. If the leaf is an echo node, the request is also considered to be the response. The response follows the inverse path in the tree, skipping actions in the branch nodes but executing actions in response pipelines. A response is then sent from the top of the tree if the interface or operation was request/response; otherwise the response is discarded.

A set of transformations that affects context variables can be defined before the message is sent to the selected endpoint or after the response is received. A web services callout can be an alternative to an XQuery or XSLT transformation to set the context variables.

Message Context

The context of a pipeline is a set of XML variables that are shared across the request flow and response flow. New variables can be dynamically added or deleted to the context, and these variables can be shared across multiple pipelines or used locally within one pipeline. Predefined context variables contain information about the message, transport headers, security principles, metadata for the current pipeline, and metadata for the primary routing and publishing services invoked by the pipeline.

The context can be read and modified by XQuery or XSLT expressions, and updated by transformation and in-place update actions. The core of the context contains the variables `$header`, `$body`, and `$attachments`. These wrapper variables contain the Simple Object Access Protocol (SOAP) header elements, SOAP body element, and Multipurpose Internet Mail Extensions (MIME) attachments, respectively. The context gives the impression that all messages are SOAP messages, and non-SOAP messages are mapped to this paradigm.

Split-Joins

The split-join message flow improves service performance by splitting a message payload and processing multiple operations in a message simultaneously and then combining, or joining, all results. A standard pipeline processes operations one after another.

The following primary elements are used to construct a message flow in a split-join:

- A *start node*, which contains the request and response variables introspected from the WSDL operation.
- A *receive node*, to receive incoming request messages.
- A *reply node*, to send response messages.
- A *scope*, which is a container that creates a context that influences the behavior of its enclosed elements.
- A *parallel node*, which is a placeholder for a fixed number of processing branches, each with its own scope.

The available elements can be combined in arbitrary ways to form a tree structure with the start node always (and only) occurring as the root of the tree. The last node is always the reply.

For more information, see [Improving Service Performance with Split-Join](#).

Transports, Adapters, and Bindings

Service Bus provides connectivity to external systems through a variety of transports, each of which is specific to a type of external system. Service Bus supports optimized database queries, and interoperability with web service integration technologies such as .NET, IBM MQ Series, IBM WebSphere, Apache Axis, and iWay adapters. The JCA transport expands the list of supported technologies by letting you connect to external systems using Oracle JCA technology and applications adapters. Additionally, Service Bus supports the REST binding, allowing you to connect to RESTful services using the HTTP transport.

You configure a transport's processing and connectivity information directly within a proxy or business service; you configure Oracle adapters using a configuration wizard specific to each adapter.

For more information, see [Working with JCA Adapters, Transports, and Bindings](#)

Supported Transport Protocols

Service Bus supports the following transport protocols:

- DSP (Oracle Data Service Integrator)
- EJB/RMI
- Email (POP/SMTP/IMAP)
- File
- (S)FTP
- HTTP(S)

- JCA
- JEJB
- JMS (including MQ using JMS, and JMS/XA)
- Local (Oracle proprietary for inter-ESB communication)
- MQ (WebSphere MQ)
- SB (RMI support)
- SOA-DIRECT (Oracle SOA Suite) and BPEL
- Tuxedo (Oracle Tuxedo)
- WS (Web Services Reliable Messaging)

Service Bus also provides the Custom Transport SDK so you can create new transports to connect with systems not covered above.

Service Types

Service Bus supports a variety of service types ranging from conventional web services (using XML or SOAP bindings in WSDL files) to non-XML (generic) services. You select and configure the service type when you create a business or proxy service. The available service types for a proxy or business service depend on the transport being used. Service Bus supports request and response as well as one-way paradigms, for both the HTTP and the JMS asynchronous transport protocols. If the underlying transport supports ordered delivery of messages, Service Bus also extends the same support.

Not all service types can be used with all transport protocols. The following table shows the service types and the transport protocols they support.

Service Type	Transport Protocols
WSDL Based Service	BPEL-10g, DSP, HTTP(S), JCA, JMS, Local, SB, SOA-DIRECT, WS JMS request and JMS response are not supported if WS-Security is enabled.
Any SOAP Service (non-WSDL)	HTTP(S), DSP, JMS, Local, SB JMS request and JMS response are not supported if WS-Security is enabled.
Any XML Service (non-WSDL)	DSP, email, File, FTP, HTTP(S), JMS, Local, MQ, SB, SFTP, Tuxedo HTTP GET is only supported for XML with no WSDL.
Messaging Service	email, File, FTP, HTTP(S), JMS, Local, MQ, SFTP, Tuxedo Business services using the email, File, FTP, or SFTP transport support one-way messaging services <i>only</i> ; the response message type should be none.

The BPEL-10g, DSP, EJB, and SOA-DIRECT transports are only supported with business services.

Transformation Resources

In addition to creating inline XQuery expressions directly in message flow actions, you can reference transformation maps that define more complex mappings between source and destination services. When disparate message data types exist between source and destination services, data mapping ensures service compatibility. Service Bus supports data mapping using XQuery and eXtensible Stylesheet Language Transformation (XSLT) standards, along with XPath expressions. You can also use cross reference tables and domain value maps to map field values between services.

Messages can be transformed in the following ways:

- Using XQuery or XSLT to reformat the message structure.
- Manipulating message content by adding, removing, or replacing certain data elements.
- Using cross reference or domain value map tables to map entities across systems.

XQuery Mappings

The XQuery Mapper in JDeveloper is a graphical tool that lets you define mappings that transform data between XML, non-XML, and Java data types so you can rapidly integrate heterogeneous applications. For example, XML data that is valid against one schema can be converted to XML that is valid against a different schema. The data can be based on XML schemas, Web Service Definition Language (WSDL) files, and Message Format Language (MFL) files. You can create an XQuery mapping in JDeveloper, and then upload the `.xqy` file generated by the mapper to an XQuery resource in the Oracle Service Bus Console. XQuery mappings are stored in XQuery resources in Service Bus, which can be referenced from the expressions you create using the expression editors in a message flow action.

The output of the XQuery Mapper is a query in the XQuery language, which is defined by the World Wide Web Consortium (W3C). For more information about W3C and the XQuery language, see <http://www.w3.org/XML/Query/>.

For more information, see [Transforming Data with XQuery](#) and "Creating Transformations with the XQuery Mapper" in *Developing SOA Applications with Oracle SOA Suite*.

XSLT Mappings

eXtensible Stylesheet Language Transformation (XSLT) maps describe XML-to-XML mappings. The XSLT mapper in JDeveloper is a graphical tool that lets you define mappings between schema root elements, Web Services Description Language (WSDL) message parts, or WSDL messages. Schema root elements can come from XSD schema files or WSDL files. Only those WSDL messages that contain a single message part can be mapped directly.

The XSLT Mapper in JDeveloper lets you define transformations that apply to the whole message body to convert messages from one XML schema to another, enabling data interchange among applications that use different schemas. You can create an XSLT mapping in JDeveloper, and then upload the `.xsl` file generated by the mapper to an XSLT resource in the Oracle Service Bus Console. XSLT mappings are stored in XSLT resources in Service Bus, which can be referenced from the expressions you create using the expression editors in a message flow action.

For more information, see [Transforming Data with XSLT](#) and "Creating Transformations with the XSLT Mapper" in *Developing SOA Applications with Oracle SOA Suite*.

Cross References

Cross reference tables map identifiers that represent equivalent objects across multiple applications, associating like objects created in different external applications. They are used to manage the runtime correlation between the various applications that share data through Service Bus. For example, you can use cross references to map customer identifiers for records that were created in multiple customer management systems. Cross reference values can be updated during runtime, allowing you to dynamically integrate values between systems. Any cross reference data updated at runtime is persisted in the database. Cross references can be used across Oracle SOA Suite components. In Service Bus, you can create cross reference tables in both JDeveloper and the Oracle Service Bus Console.

Service Bus provides a set of XPath functions for looking up and modifying cross reference values. These functions are available to use in the expressions you create using the expression editors in a message flow action. For more information, see [Mapping Data with Cross-References](#).

Domain Value Maps

A domain value map associates terms used by different domains to describe the same entity, providing the capability to map the terms across vocabularies or systems. For example, each domain might use different terminology for country codes, city codes, currency codes, and so on. You can enter these values in a map and look up those values at runtime to transform the data when passing it from one domain to another. Domain value maps are similar to cross references, but they are defined statically rather than dynamically. You create and populate domain value maps in the design time, and deploy them to the runtime. Domain value map data are not changed by runtime activities as it is for cross references, but rather the domain value maps are used for lookups only.

Domain value maps can be used across Oracle SOA Suite components. In Service Bus, you can create domain value maps in both JDeveloper and the Oracle Service Bus Console. Service Bus provides a set of XPath functions for looking up domain value map values. These functions are available to use in the expressions you create using the expression editors in a message flow action. For more information, see [Mapping Data with Domain Value Maps](#).

Transport and Adapter Related Resources

Some transports rely on specific types of files, such as JavaScript and JAR files or MQ connections. The JCA transport requires the JCA file and any files it references, such as a WSDL file. This section describes the resources that are specific to certain transports.

JCA Bindings

JCA binding resources in Service Bus let you create business and proxy services that interact with external services through Oracle SOA Suite JCA adapters. A JCA binding is made up of a service WSDL document and a corresponding JCA file created in Oracle JDeveloper. In JDeveloper, you can add a JCA adapter directly to a Service Bus project using the Service Bus Overview Editor by dragging and dropping the adapter type from the Components window to the editor's canvas. The proxy or business service is automatically generated from the JCA adapter configuration, and is based on the JCA transport. In the Oracle Service Bus Console, you need to upload the JCA file

into a JCA binding resource in order to create a business or proxy service based on that JCA adapter. You can also import the JCA file and its associated WSDL file using the import feature.

For more information, see [Working with JCA Binding Resources](#).

JAR Files (Archives)

A JAR (Java Archive) is a zipped file that contains a set of Java classes. It is used to store compiled Java classes and associated metadata that can constitute a program. A JAR acts like a callable program library for Java code elements, so a single compilation link provides access to multiple elements rather than requiring bindings for each element individually.

In JDeveloper, you can add JAR files to a project or component directly from the file system, but in the Oracle Service Bus Console, you need to upload each JAR file to add to a project into an *archive* resource. In Service Bus, JAR files are used the following components:

- Java callout actions (in pipelines) that provide a Java exit mechanism
- EJB-based business services
- JEJB-based business and proxy services
- Tuxedo-based proxy and business services

For more information, see [Working with JAR Files](#).

JavaScript Files

JavaScript files are used by the JCA Socket Adapter as a mechanism for handling the handshake. XSLT and custom Java code are also supported handshake mechanisms. In JDeveloper, you can create a JavaScript file and then select the file when you configure a JCA socket adapter. You can also create the JavaScript when you configure the adapter. In the Oracle Service Bus Console, you can either upload an existing JavaScript file to a JavaScript resource, or you can create the text for the JavaScript in a text editor in the console. Alternatively, you can use the console's import feature to import the Socket adapter's JCA file and its dependencies, such as WSDL and JavaScript files.

Service Bus supports JavaScript handshake for both inbound and outbound socket adapters, and for one-way and request/response messaging. Request/response handshakes require a separate JavaScript file for the request and the response.

For more information, see [Working with JavaScript Resources](#).

MQ Connections

MQ connection resources provide the connection parameters required to connect to an MQ queue manager. They are used in proxy and business services configured to use the MQ transport, and can be shared and reused across multiple services. MQ proxy and business services must connect to an MQ queue manager before they can access an MQ queue. Each MQ connection resource uses a connection pool, and every business or proxy service that connects to a queue manager using the same MQ connection resource also uses the same connection pool. Thus, multiple business and proxy services can use the same queue manager and share a connection pool.

In order to create MQ connections in the Oracle Service Bus Console, you must install the WebSphere MQ client library to the Service Bus domain. This is described in [How to Create MQ Connections](#).

Schema and Document Resources

Service Bus services rely on different document types to define information like message structures and web interfaces. These documents include XML schemas, MFL files, and XML files to describe data, and WSDL and WADL documents to describe interfaces.

Service Bus has a built-in type system that is available for use at design time. When creating an XQuery expression in a condition, in-place update action, or transformation, the variable can be declared to be of a given type in an editor to assist in easily creating the XQuery. The types can be the following:

- XML schema types or elements
- WSDL types or elements
- MFL types

XML Schemas

XML schemas are documents that define valid content for primitive or structured data in XML documents. XML schemas specify the structure of documents, the data type of each element and attribute contained in the document, and the rules that XML business data must follow. XML schemas can import or include other XML schemas. Schemas are used to add XML information to messages exchanged in Service Bus, and may be required for XQuery expressions, WSDL files, and so on.

For more information, see [Working with XML Schemas](#).

XML Documents

XML document resources store XML files that can then be referenced when configuring proxy or business services. For example, you might use XML documents for TopLink mapping files needed in JCA proxy or business services that communicate with JCA-compliant systems.

XML documents are a standard feature in JDeveloper. In the Oracle Service Bus Console, the easiest way to create XML documents is to use the import feature. For example, if you import JCA resources (JCA file, along with its associated WSDL and mapping files), Service Bus automatically generates XML document resources out of mapping files and maintains the dependencies among resource files. You can also create an XML document resource, and upload the contents of an XML file to the resource.

For more information, see [Working with XML Documents](#).

WSDL Documents

A Web Service Definition Language (WSDL) interface defines a service interface for a SOAP or XML service. For web services, a WSDL document describes what the web service's interface is, where it resides, and how to invoke it. Service Bus defines proxy and business services in terms of two WSDL entities:

- The abstract WSDL interface, which defines the operations in that interface and the types of message parts in the operation signature.

- The binding WSDL interface, which defines the binding of the message parts to the message (packaging), and the binding of the message to the transport.

A WSDL file can also describe the concrete interface of the service (for example, the transport URL).

You can base the definition of a proxy or business service on an existing WSDL file, which automatically configures portions of the service. WSDL files used as the basis for defining services are stored as Service Bus resources. In JDeveloper, you can create WSDL files using the built-in WSDL editor. You can then import those WSDL files, and any schemas used by the file, into the Oracle Service Bus Console. The console can also be used to resolve the references in the WSDL files, ensuring all schemas and WSDL files are linked correctly. Service Bus uses its own representation of the interface for messaging services.

For more information, see [Working with WSDL Documents](#).

WADL Documents

A Web Application Definition Language (WADL) document is similar to a WSDL document, described above, but it is specifically used to describe the interface for REST proxy or business services. When you create a proxy or business service based on the REST binding in JDeveloper, the required WADL document is automatically generated from the WSDL document you specify for the binding. A WADL file can have dependencies on a WSDL file and on one or more XML schemas.

If you are using the Oracle Service Bus Console, you can create WADL documents by importing them or by creating a WADL resource. For more information, see [Creating WADL Documents](#).

MFL Resources

Service Bus uses Message Format Language (MFL) to describe the structure of typed non-XML data. MFL is an Oracle proprietary language used to define the rules that transform formatted binary data into XML data. MFL documents are used at runtime to transform an instance of a non-XML data record to an instance of an XML document (or the other way around).

You create MFL documents using the Format Builder tool in JDeveloper. The Format Builder allows you to describe the layout and hierarchy of the non-XML data so it can be transformed to or from XML. Using the Format Builder, you define each field in the message, including the type and size of data, the name of the field, any delimiters, and so on. You can also indicate whether the field is repeating, and whether it is optional or required.

For more information, see [Defining Data Structures with Message Format Language](#).

Security Resources

Security information can be passed through proxy and business services using service accounts, which define how the user name and password are obtained, or using service key providers, which define encryption credentials.

Service Key Providers

Service Key Provider resources contain Public Key Infrastructure (PKI) credentials used by proxy services for decrypting inbound SOAP messages and for outbound authentication and digital signatures. PKI credentials are private keys paired with certificates that can be used for digital signatures and encryption (for Web Services

Security) and for outbound SSL authentication. The certificate contains the public key that corresponds to the private key.

Service Bus uses service key providers to supply the following types of credential-level validation to proxy services.

- SSL client authentication
- Digital signature
- Encryption
- Web Services Security X509 token

For more information, see [Working with Service Key Providers](#).

Service Accounts

Service account resources provide user names and passwords that Service Bus uses for authentication when connecting to a service or server. Service accounts are used by proxy and business services for outbound authentication or for authentication to a local or remote resource, such as an FTP server or a JMS server. You can configure a service account to use a specific user name and password pair, to use the user names and passwords received from incoming requests, or to map user names and passwords provided by clients to user names and passwords you specify. One service account can be used for multiple business and proxy services.

For more information, see [Working with Service Accounts](#).

WS-Policy Resources

In previous versions, WS-Policy resources were used to store custom web service policies so they could be referenced by multiple WSDL documents. Beginning in 12c, Oracle Web Services Manager (OWSM) policies replace WLS9 policies, so there is no longer an option to create new services with WSDL-based WLS9 policies. While WS-Policy resources are still visible in imported projects, the associated web services should be updated to use OWSM policies. However, you can still import and activate a project from a previous version that uses WS-Policy resources.

Alert Destinations

An alert destination resource defines a list of recipients that can receive alert notifications from Service Bus. For example, when a service level agreement (SLA) or pipeline alert is generated, you can specify that notifications be sent to specific email addresses or JMS queues, ensuring that only the relevant people receive the notifications. An alert destination could include one or more of the following types of destinations: the Service Bus reporting data stream, SNMP trap, alert log, email, JMS queue, or JMS topic. In the case of email and JMS destinations, a destination resource could include a list of email addresses or JMS URIs, respectively.

For more information, see [Working with Alert Destinations](#).

Throttling Group Resources

Throttling helps improve performance and stability by preventing message overload on high-traffic business services. To control the flow of messages to a business service and prevent backlogs, you can enable and configure message throttling for a business service or group of business services in your Service Bus applications. When messages are throttled, the business service can only concurrently process the number of

messages you specify. When that capacity is reached, messages are stored in an in-memory queue until the business service is ready to process more messages.

For more information, see "Configuring Business Services for Message Throttling" in *Administering Oracle Service Bus*.

System Resources

System resources are globally available resources that can be shared across all projects in your Service Bus instance. They define server connections and authentication information, and include the following:

- [JNDI Providers](#)
- [SMTP Servers](#)
- [Proxy Servers](#)
- [UDDI Registries](#)

In the Oracle Service Bus Console, system resources are stored in a separate project named System. In JDeveloper, system resources are stored in the Application Resources panel under **Service Bus System Resources**.

JNDI Providers

JNDI provider resources define the connection and authentication information needed to access JNDI-named objects. They describe the URL (or list of URLs in the case of clustered deployments) of the JNDI providers used by Service Bus. For example, in a business service used to invoke an EJB, you include the name of a JNDI provider resource in the endpoint URI. When the business service is invoked, Service Bus uses the details in the referenced JNDI provider resource to make the initial connection to the JNDI provider.

If the JNDI provider is secured, then the JNDI provider resource also defines a user name and password to gain access. JNDI providers offer a great deal of flexibility. If a JNDI connection changes, you only need to modify the JNDI provider resource, and anything that references the JNDI provider automatically uses the updated configuration.

For more information, see [Working with JNDI Provider Resources](#).

SMTP Servers

SMTP server resources define the address of SMTP servers corresponding to email destinations, port numbers, and, if required, authentication credentials. They describe the URL for the SMTP servers used by Service Bus. If the SMTP server is secured, the SMTP server resource description also includes a user name and password to gain access. SMTP server resources are referenced when configuring alert destination resources and email transport-based business services.

For more information, see [Working with SMTP Server Resources](#).

Proxy Servers

Proxy server resources define the connection and authentication information needed to access JNDI-named objects. They describe the URL for the proxy servers used by Service Bus. If the proxy server is secured, the proxy server resource description also includes a user name and password to gain access. You can use a proxy server to proxy requests from a client application, and you typically use a proxy server when

Service Bus is behind a firewall. When you configure business services to route messages through a proxy server, associate the proxy server resource with that business service. This instructs Service Bus to connect to the business service through the configured proxy server.

You can configure multiple proxy servers for each proxy server resource. In this case, Service Bus can perform load balancing and offer fault tolerance among the configured proxy servers.

For more information, see [Working with Proxy Server Resources](#).

UDDI Registries

Universal Description, Discovery and Integration (UDDI) registries are used to share web services. UDDI provides a framework in which to classify your business, its services, and the technical details about the services you want to expose. A UDDI registry resource stores information about a UDDI registry accessed by Service Bus for service discovery, publishing, and synchronization. After the UDDI registry resource is configured, you can publish Service Bus proxy services to the associated registry, or import business services from the registry to be used by a proxy service. UDDI registry resources define the inquiry, publish, security, and subscription URLs, along with a user name and password to gain access to the registry.

For more information, see [Working with UDDI Registries](#).

Service Bus Messaging

This section discusses the types of messaging supported by Service Bus, the message formats, and the context variables that are passed through, and optionally modified by, the components in a Service Bus project.

Messaging Models

Service Bus accommodates multiple messaging paradigms and supports the following types of communication:

- Synchronous request/response
- Asynchronous publish one-to-one
- Asynchronous publish one-to-many
- Asynchronous request/response (synchronous-to-asynchronous bridging)

In sync-async bridging, a synchronous client issues a request to an asynchronous provider. For this pattern, you can publish a message to one JMS queue and configure a second JMS queue for the response, with a timeout value for listening for the response. This type of service appears as a synchronous service to the service consumer. Using asynchronous request/response messages has these advantages:

- No blocking by the request thread, removing thread management issues that can occur when numerous blocking request/response invocations are made.
- More reliable messaging.

Message Formats

Service Bus supports the following message formats:

- Email with or without attachments
- Java
- JMS with headers
- MFL (Message Format Language)
- Raw Data (opaque non-XML data with no known schema)
- Text
- SOAP and SOAP with attachments (SOAP described or not described by a WSDL document)
- XML and XML with attachments (XML described or not described by a WSDL document or a schema)

Message Context

All messages sent to and received by a proxy service are defined internally by a set of properties that hold the message data and metadata related to that message. This set of properties is known as the message context and is implemented using context variables. The context is defined by an XML schema. Some context variables are predefined and others are user defined.

Predefined context variables contain information about the message, transport headers, security principals, metadata for the current proxy service, and metadata for the primary routing and publishing services invoked by the proxy service. You typically use an XQuery expression in a pipeline to manipulate context variables as a message moves through Service Bus. You can also modify context variables using transformation and in-place update actions.

For a complete description of the message context and context variables used in the message flow, see [Message Context](#).

Content Types

To support interoperability with heterogeneous endpoints, Service Bus lets service configurations control the content type, JMS type, and encoding used. It does not make assumptions about what the external client or service needs, but instead uses the configuration of the proxy or business service. Service Bus derives the content type for outbound messages from the service type and interface and uses the following specifications:

- For XML or SOAP (with or without a WSDL file), the content type is text/XML.
- For messaging services when the interface is MFL or binary, the content type is binary/octet-stream.
- For messaging services when the interface is text, the content type is text/plain.
- For messaging services when the interface is XML, the content type is text/XML.

The content type can be overridden in the outbound context variable (`$outbound`) for pipelines invoking a service, and in the inbound context variable (`$inbound`) for a pipeline response. Additionally, there is a JMS type (byte or text) that can be configured when the service is defined. Encoding is explicitly configured in the service definition for all outbound messages.

Using Work Managers with Service Bus

Service Bus uses Oracle WebLogic Server Work Managers to optimize performance and to maintain service-level agreements. Work Managers prioritize work and allocate threads based on rules you define and based on runtime performance. When you create and configure a Work Manager, you define the maximum and minimum number of threads to use, server capacity, and request and response classes that express scheduling guidelines. One default Work Manager is provided, but you can create as many Work Managers as you need to optimize your services. In Service Bus, you specify a Work Manager for a proxy service or business service in the Dispatch Policy property of the transport configuration.

For more information about Work Managers, see "Using Work Managers to Optimize Scheduled Work" in *Administering Server Environments for Oracle WebLogic Server*. For more information about Work Managers in Service Bus, see [Using Work Managers with Service Bus](#).

Service Bus Security

Service Bus uses Oracle Platform Security Services (OPSS) and Oracle Web Services Manager (OWSM) as the building blocks for higher-level security services including authentication, identity assertion, authorization, role mapping, auditing, and credential mapping. In order to configure Service Bus access security, you must first configure Oracle WebLogic Server security. Service Bus uses OWSM to provide a policy framework to manage and secure web services consistently across your organization.

Service Bus Security Features

Service Bus provides the following security features:

- Integration with OWSM and OPSS
- Authentication, encryption and decryption, and digital signatures as defined in the Web Services Security (WS-Security) specification
- SSL to support traditional transport-level security for HTTP and JMS transport protocols
- One-way and two-way certificate based authentication
- HTTP basic authentication
- Encryption and export of resources (such as service accounts, service key providers, UDDI registries, SMTP providers, and JNDI providers) that contain user names and passwords
- Service accounts and service key providers to define the user name, password, and credential alias binding
- Client-specified custom authentication credentials for both transport-level and message-level inbound requests

Service Bus Service Security Model

A Service Bus service can be secured by security policies that apply to messages in its interface. A security policy can be specified for a service or for individual messages

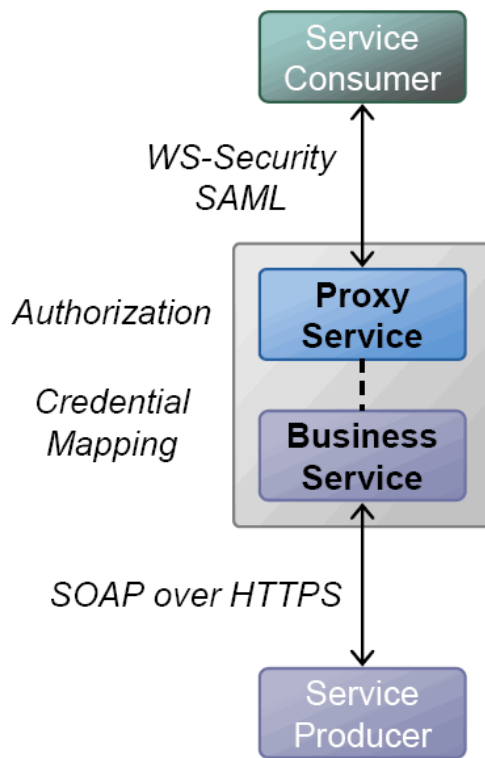
associated with the operations of a service. When a security policy is specified for a service, the policy applies to all messages sent to that service.

You can secure Service Bus services using the following types of security:

- Inbound security
- Outbound security
- Options for identity propagation
- Administrative security
- Supported standards and security providers

Figure 1-4 illustrates security features at different points in a message life cycle.

Figure 1-4 Optimized Pluggable Security Layer



Oracle Web Services Manager

You can secure your Service Bus services by attaching Oracle Web Services Manager (OWSM) policies. OWSM is a component of Oracle Enterprise Manager Fusion Middleware Control, a runtime framework that provides centralized management and governance of Oracle SOA Suite environments and applications. It provides capabilities to build, enforce, run, and monitor web service policies, such as security, reliable messaging, MTOM, and addressing policies. OWSM can be used by developers at design time and by system administrators in production environments. OWSM allows for policy-driven centralized management of web services with local enforcement. OWSM provides a policy framework to manage and secure web services consistently across your organization.

Oracle Platform Security Services

Oracle Platform Security Services (OPSS) provides a standards-based, portable, integrated, enterprise-grade security framework for Java Standard Edition (Java SE) and Java Enterprise Edition (Java EE) applications. OPSS provides an abstraction layer in the form of standards-based APIs that insulate developers from security and identity management implementation details. Developers do not need to know the details of cryptographic key management or interfaces with user repositories and other identity management infrastructures.

WS-Policies

Through OWSM, Service Bus security supports the Web Services Policy (WS-Policy) specification, a standards-based framework for defining a web service's security constraints and requirements using policies, each of which contains one or more assertions. WS-Policy assertions specify a web service's requirements for digital signatures and encryption, along with the security algorithms and authentication mechanisms that it requires.

You can include WS-Policy policies directly in a WSDL document or include them by reference. A WSDL document can import other WSDL documents that contain or refer to WS-Policy policies. The runtime environment recognizes both abstract and concrete WS-Policy statements. Abstract WS-Policy statements do not specify security tokens. Concrete WS-Policy statements specify the security tokens for authentication, encryption, and digital signatures. The Service Bus runtime environment determines which security token types an abstract policy will accept.

For more information on WS-Policy specification, see the Web Services Policy Framework (WS-Policy) and Web Services Policy Attachment (WS-PolicyAttachment) which is available at <http://specs.xmlsoap.org/ws/2004/09/policy/>.

Types of Security

The following sections discuss the security features available in the Service Bus security model.

- [Inbound Security](#)
- [Outbound Security](#)
- [Identity Propagation](#)
- [User Management and Administrative Security](#)
- [Transport-Level Security](#)
- [Message-Level Security](#)

Inbound Security

Inbound security ensures that proxy services handle only the requests that come from authorized clients, and that no unauthorized user has viewed or modified the data sent from the client. For outward-facing proxy services, which receive requests from service consumers, strict security requirements such as two-way SSL over HTTPS are used. If a proxy service uses public key infrastructure (PKI) technology for digital signatures, encryption, or SSL authentication, you can create a service key provider to provide private keys paired with certificates.

Outbound Security

Outbound security secures communication between a proxy service and a business service. Most of the tasks involve configuring proxy services to comply with the transport-level or message-level security requirements that business services specify. If a business service requires SSL authentication or PKI technology for digital signatures, a service key provider is required, which provides private keys paired with certificates.

Identity Propagation

The options provided by Service Bus for identity propagation allow for decision making when designing security, including how to propagate the identities that clients provide. Service Bus can be configured to authenticate the credentials provided by clients, perform authorization checks, pass credentials through as is, and map credentials.

User Management and Administrative Security

Service Bus user management is based on WebLogic Server security, which supports task-level authorization based on security policies associated with roles assigned to named groups or individual users. You use Fusion Middleware Control and the Oracle WebLogic Server Administration Console to manage Service Bus users, groups, and roles.

To give users access to functions, such as creating proxy services and other resources, you assign them to one or more of the predefined security roles with predefined access privileges. The access privileges for the Service Bus administrative security roles cannot be changed but the conditions under which a user or group is in one of the roles can be changed. By default, the first user created for an Service Bus domain is WebLogic Server administrator. This user has full access to all Service Bus objects and functions, and can execute user management tasks to provide controlled access to Service Bus functions.

For more information, see "Defining Access Security for Oracle Service Bus" in *Administering Oracle Service Bus*.

Transport-Level Security

Service Bus supports transport-level confidentiality, message integrity, and client authentication for one-way requests or request/response transactions over HTTPS. HTTP(S) proxy services or business services can be configured to require basic authentication, client certificate (two-way SSL) authentication, custom authentication, or no client authentication at all. Transport security for transports other than HTTP is supported in Service Bus as follows:

- For the email and FTP transports, security is provided using credentials to connect to a FTP or email server.
- For the file transport, security is provided using a login control to the machine on which the files are located.

Message-Level Security

Service Bus supports OASIS Web Services Security (WSS) 1.0. WSS defines a framework for message confidentiality, integrity, and sender authentication for SOAP messages. Using WSS, Service Bus provides support for securing messages using digital signatures, encryption, or both. Though it is not a substitute for transport-level

security, WSS is ideal for end-to-end message confidentiality and integrity. WSS is more flexible than SSL since individual parts of the SOAP envelope can be signed, encrypted or both, while other parts are neither signed nor encrypted. This is a powerful feature when combined with the ability of Service Bus to make routing decisions and perform transformations on the data based on the message content. Service Bus supports WSS over HTTP(S) and JMS.

For more information on the WSS specification, see the OASIS Web Services Security TC which is available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

Custom Security Credentials

Service Bus supports client-specified custom authentication credentials for both transport-level and message-level inbound requests. The custom authentication credentials can be in the form of tokens, or a user name and password token combination. Service Bus accepts and attempts to authenticate the following:

- A custom token passed to a proxy service in an HTTP header, SOAP header (for SOAP-based proxy services) or in the payload (for non-SOAP proxy services).
- A user name and password token passed in a SOAP header (for SOAP based proxy services), or in the payload for non-SOAP proxy services.

For outbound requests, custom authentication is supported at the transport-level based on a custom authenticator Java class you create. The custom authentication mechanisms work alone or in concert with the message-level security for web services. For more information on custom security credentials, see [Configuring Custom Authentication](#).

Approaches for Designing Service Bus Services

When creating Service Bus services, you have a choice of approaches, depending on whether you use the Oracle Service Bus Console or JDeveloper. JDeveloper supports both approaches; the console supports the bottom-up approach.

- **Top-Down:** With this approach, you analyze your processes and identify activities in support of this process. You create a Service Bus application and project, and define the Service Bus components through the Service Bus Overview Editor.
- **Bottom-Up:** With this approach, you analyze existing applications and assets to identify those that can be used as services. As you create a Service Bus application, you build the services on an as-needed basis. This approach works well when IT must react to a change.

Service Bus Top-Down Roadmap

With this approach to developing Service Bus services, you can create all your primary components at one time using the Service Bus Overview Editor. This includes proxy and business services, pipelines, split-joins, and JCA adapters. The editor simplifies the creation and configuration of project components, and provides a graphic view of the overall structure of the data flow. Once you create and wire these components, you can define the configuration options for each, and define message routing and transformation in pipelines and split-joins.

The following table outlines the steps and provides links for further information.

Table 1-1 Service Bus Development Roadmap - Top-Down Approach

Step	Description	More Information
1	Create the necessary supporting resources, such as service accounts, WSDL files, or XQuery maps.	Links for each type of resource are provided in Oracle Service Bus Overview .
2	Create any proxy services, pipelines, business services, and optionally split-joins.	Adding Service Bus Components
3	Wire the components together.	Wiring Service Bus Components
4	Configure the proxy services and their transports.	Configuring Proxy Services Working with JCA Adapters, Transports, and Bindings
5	Configure the pipelines and split-joins.	Working with Oracle Service Bus Pipelines Improving Service Performance with Split-Join
6	Configure the business services and their transports.	Configuring Business Services Working with JCA Adapters, Transports, and Bindings
7	Configure security for the business and proxy services.	Securing Business and Proxy Services
8	Test and debug the services and resources.	Debugging Oracle Service Bus Applications Using the Test Console
9	Deploy the service.	Deploying Oracle Service Bus Services
10	Monitor and administer the runtime.	Oracle Service Bus Runtime Monitoring

Service Bus Bottom-Up Roadmap

With this approach to developing Service Bus services, you create and configure each project component individually. The flow of messages through the system is defined by references you create between project components, such as proxy services, business services, pipelines, split-joins. This approach does not use the Service Bus Overview Editor, so you are not working with a graphical representation of the components. However, if you are working in JDeveloper, the components you create are added to the overview file and appear in the Service Bus Overview Editor.

The following table outlines the steps and provides links for further information.

Table 1-2 Service Bus Development Roadmap - Bottom-Up Approach

Step	Description	More Information
1	Create the necessary supporting resources, such as service accounts, WSDL files, or XQuery maps.	Links for each type of resource are provided in Oracle Service Bus Overview .

Table 1-2 (Cont.) Service Bus Development Roadmap - Bottom-Up Approach

Step	Description	More Information
2	Create a proxy service and pipeline. You can generate a proxy service when you create the pipeline.	Creating Proxy Services Working with Oracle Service Bus Pipelines
3	Configure the proxy service and its transport.	Configuring Proxy Services Working with JCA Adapters, Transports, and Bindings
4	Define the message flow in the pipeline.	Working with Oracle Service Bus Pipelines
5	Optionally, create and configure a split-join for parallel processing.	Working with Split-Joins in JDeveloper
6	Create and configure a business service.	Creating Business Services Configuring Business Services
7	Configure security for the services.	Securing Business and Proxy Services
8	Test and debug the services and resources.	Debugging Oracle Service Bus Applications Using the Test Console
9	Deploy the service.	Deploying Oracle Service Bus Services
10	Monitor and administer the runtime.	Oracle Service Bus Runtime Monitoring

Naming Guidelines for Service Bus Components

When naming any directory or resource in a Service Bus project, the following characters are allowed:

- All Java identifier characters, including Java keywords, as described in the "Identifiers" and "Keywords" sections of the Java Language Specification at http://java.sun.com/docs/books/jls/third_edition/html/lexical.html#3.8.
- Blanks, periods, and hyphens within the names (not leading or trailing).

Characters such as / \ * : " < > ? | are not allowed.

Viewing Service Bus Resources in a Web Browser

You can view some of the Service Bus resources described in this document in a standard web browser using the URLs, described in the following sections:

- [WSDL Documents](#)
- [WS Policies](#)
- [Message Format Language \(MFL\) Resources](#)
- [Schema Resources](#)

- [Notes About Viewing Service Bus Resources in a Web Browser](#)

WSDL Documents

- URL to display a WSDL document:
`http://host:port/sbresource?WSDL/project_path/wsdlname`
- URL to display the WSDL document for WSDL-based HTTP proxy services:
`http://host:port/proxy_service_endpoint_URI?WSDL`
- URL to display the WSDL document for WSDL-based proxy services:
`http://host:port/sbresource?PROXY/project_path/proxyname`
- URL to display the WSDL document for proxy services with Oracle Web Services Manager policies attached:
`http://host:port/sbresource?ORAPROXY/project_path/proxyname`
- URL to display the WSDL document for WSDL-based business services:
`http://host:port/sbresource?BIZ/project_path/bizname`
- URL to display the WSDL document for WSDL-based business services with Oracle Web Services Manager policies attached:
`http://host:port/sbresource?ORABIZ/project_path/bizname`

WS Policies

Use the following URL to display WS security policies:

`http://host:port/sbresource?POLICY/project_path/policyname`

Message Format Language (MFL) Resources

Use the following URL to display an MFL file:

`http://host:port/sbresource?MFL/project_path/mflname`

Schema Resources

Use the following URL to display an XML schema:

`http://host:port/sbresource?SCHEMA/project_path/schemaname`

Notes About Viewing Service Bus Resources in a Web Browser

You can also retrieve WSDL documents containing Oracle Web Services Manager policies so the policies conform to supported WS-Policy and WS-Security Policy standards. For more information, see [Advertising WSDL Files to Support WS Standards..](#)

If you use special characters in your resource names, the URLs used to expose the resources in Service Bus must be encoded in UTF-8 in order to escape special characters.

Accessibility Options

Service Bus uses both JDeveloper and the Oracle Service Bus Console for development. The following sections describe accessibility options for both environments.

- [Setting Accessibility Options in JDeveloper](#)
- [Notes on Screen Reader Mode](#)
- [How to Set Accessibility Options in the Oracle Service Bus Console](#)

Setting Accessibility Options in JDeveloper

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

Notes on Screen Reader Mode

When you log in to the Oracle Service Bus Console with the screen reader mode enabled, selecting the context menus from the Project Navigator requires extra steps. To access the context menus for projects and components in the Project Navigator, navigate to the component using the **Tab** key, press **Enter** to select the component, and then press **Ctrl+Alt+M** to launch the menu. Use the up and down arrows to navigate the options in the menu.

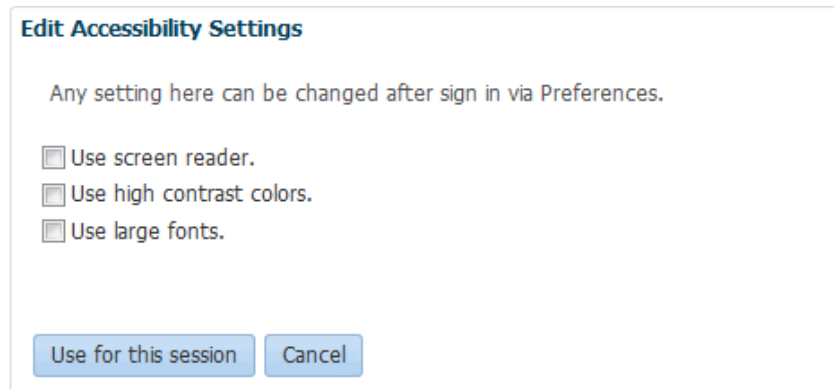
How to Set Accessibility Options in the Oracle Service Bus Console

Accessibility settings help you read all components of the application. You can set accessibility options in the Oracle Service Bus Console for the current instance only. The console presents the Accessibility menu on the login page, so you can configure accessibility before you log in.

To set accessibility options:

1. Launch the Oracle Service Bus Console.
2. On the login page, click **Accessibility** in the top right corner.

The Edit Accessibility Settings page appears, as shown below.

Figure 1-5 Edit Accessibility Settings Page

Note:

This page indicates that settings can also be changed using the Preferences option once you log in. Currently, you can only configure accessibility options from the Accessibility Settings page.

3. Select any of the following options:
 - Use screen reader.
 - Use high contrast colors.
 - Use large fonts.
4. Click **Use for this session**.

Additional Resources

In addition to this developer's guide, Oracle also offers the following resources to help you learn how you can best use Service Bus.

- *Understanding Oracle SOA Suite* introduces you to Oracle SOA Suite and Oracle Service Bus, and provides you with a high-level understanding of what you can accomplish with the suite.
- *Administering Oracle Service Bus Services* provides information on how to monitor running services and update the runtime environment.
- Oracle Service Bus samples provide additional learning tools to help you get started with Service Bus features at <http://www.oracle.com/technetwork/middleware/service-bus/learnmore/index.html>.

The *Oracle Fusion Middleware Documentation Library* provides a complete list of Service Bus documents.

Getting Started with the Oracle Service Bus Console

This chapter provides general information about how to use the Oracle Service Bus Console to configure services and other Service Bus resources. These are largely design-time activities that require a running WebLogic Server instance, but you can also perform runtime tasks using the console.

This chapter includes the following topics:

- [Overview of the Oracle Service Bus Console](#)
- [Getting Started](#)
- [Working with Sessions](#)
- [Working with Projects, Folders, and Resources in Oracle Service Bus Console](#)
- [Viewing and Resolving Conflicts](#)
- [Viewing Historical Data](#)
- [Undoing Changes and Activations](#)
- [Viewing References](#)
- [Customizing the Appearance of the Oracle Service Bus Console](#)

Overview of the Oracle Service Bus Console

Oracle Service Bus Console is a web-based console where you can create and configure most Service Bus resources, test the resources, and activate your changes to the runtime. You can also import and export Service Bus configuration JAR files. The console utilizes a *change session* mechanism similar to the WebLogic Server Administration Console, where you can complete your changes within a session, and once you are satisfied with those changes, activate them into the runtime. Service Bus components, called *resources*, are grouped into projects and folders.

Service Bus Sessions

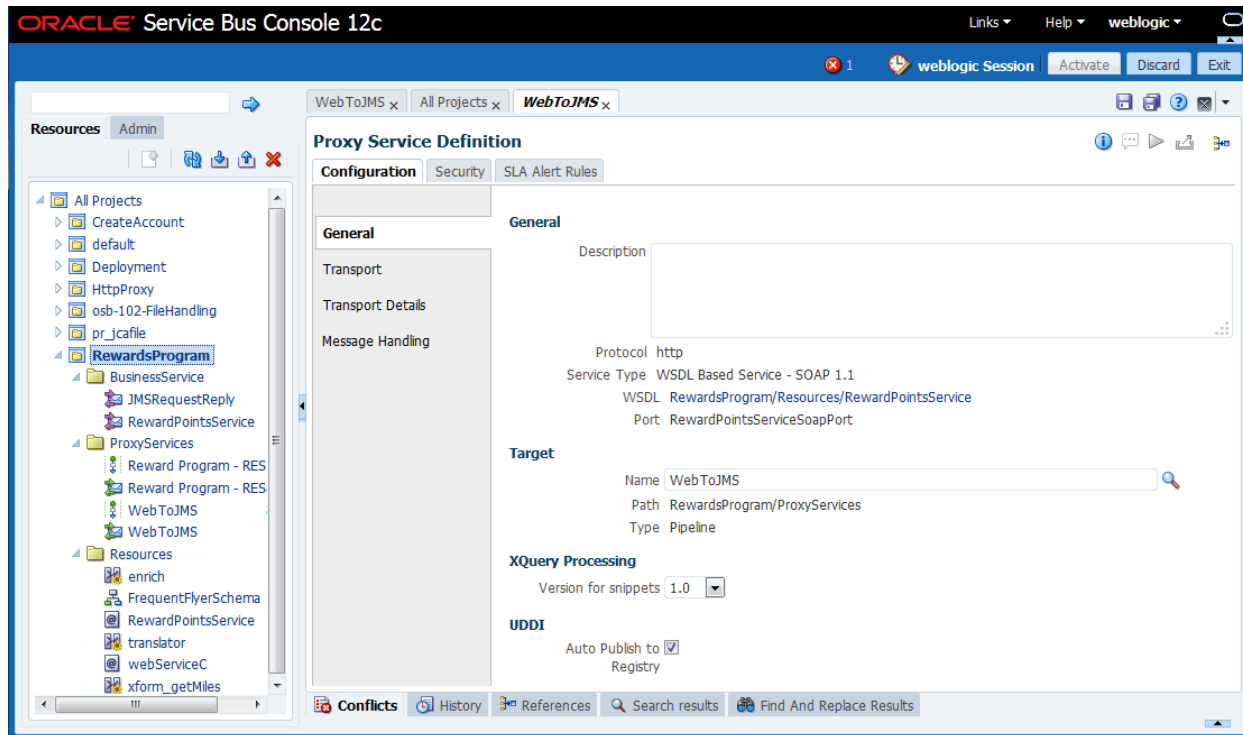
Most of what you do in the Oracle Service Bus Console is done within an open session. Once you make your changes to Service Bus components and projects, resolve any conflicts, and are ready to propagate the changes to the runtime, you can activate the session. Sessions allow team collaboration when services and metadata are being configured in Service Bus. Each team member works in a sandbox session until they are ready to check in the working configuration to the core configuration of the bus. Sessions provide multiple levels of undo, and visibility into conflicts, as multiple users work on the configuration.

The available options on each page vary depending on whether you are in a session. For example, you cannot edit resources outside of a session, and you can only test services when you are working outside of a session. Also, in a session, the **Changes** page that appears when you click **History** lists all the changes you have made in that session; outside a session, the page lists no changes.

Oracle Service Bus Console Layout

The Oracle Service Bus Console lists all projects, folders, and resources in the Project Navigator in a tree view. Selecting any component in the navigator displays its configuration in an editor. Most editors are divided into tabbed pages, each of which let you configure specific types of properties. [Figure 2-1](#) shows an example of a proxy service displayed in the Proxy Service Definition Editor. The top toolbar provides links to the WebLogic Server Administration Console, Fusion Middleware Control, and online help topics. The editor toolbar lets you save changes, view information about the displayed component, launch the Test Console or Pipeline Editor, and export the WSDL file on which a service is based.

Figure 2-1 Proxy Service Configuration on the Oracle Service Bus Console



By selecting any of the auxiliary tabs at the bottom of the page, you can view conflicts, any references for the currently displayed resource, a history of changes and activations, the results of a Find and Replace action, and the results of a search performed from the Search field above the Project Navigator. Use these tabs to resolve conflicts before activating changes, and to undo actions or even session activations. [Figure 2-2](#) shows the **History** tab with the **Changes** subtab selected.

Figure 2-2 Auxiliary Tabs in the Oracle Service Bus Console

The screenshot shows the Oracle Service Bus Console interface. At the top, there are several tabs: Conflicts, History, References, Search results, and Find And Replace Results. Below these, there are more tabs: Changes, Sessions, and Activations. Under the 'Changes' tab, there are sub-tabs for View, Detach, and another icon. The main area displays a table with the following data:

Task	Execution Time	User	Task Status	Undone By
Update Pipeline RewardsProgram/Pro...	3/24/2014 1:07 ...	weblogic	Completed	
Update Pipeline RewardsProgram/Pro...	3/24/2014 1:07 ...	weblogic	Completed	
Update Pipeline RewardsProgram/Pro...	3/24/2014 1:06 ...	weblogic	Completed	
Delete BusinessService RewardsProgr...	3/24/2014 12:59...	weblogic	Completed	
Delete BusinessService RewardsProgr...	3/24/2014 12:58...	weblogic	Completed	
Clone Folder RewardsProgram/Busine...	3/24/2014 12:58...	weblogic	Completed	
Clone Folder pr_jcafile/Resources int...	3/24/2014 12:58...	weblogic	Completed	

Service Bus Projects and Folders

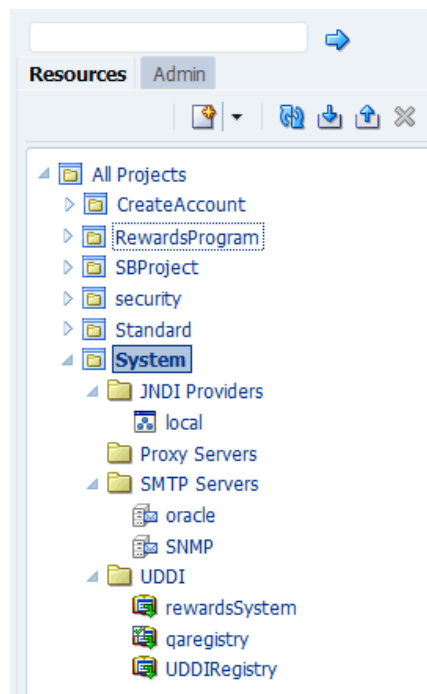
The Project Navigator organizes configurations and resources into projects and folders. You can view all the projects in a domain on the All Projects Definition Editor; all the resources in a project on the Project Definition Editor; and all the resources in a folder on the Folder Definition Editors. On the Project and Folder Definition Editors, you can also take actions against resources, such as launching the Test Console or launching the Pipeline Editor. Click the name of a project, folder, or resource on any of these editors to display the editor for that component.

All Service Bus resources, such as services, WSDL files, and XQuery transformations, reside in only one project. Projects do not overlap. Resources can be created directly under a project, or they can be further organized into folders. However, you can reference any resource regardless of the project in which it resides.

When you create a domain in Service Bus, a default project is automatically created.

The System Project

The Project Navigator includes a default project named System, which contains a flat file of global resources such as JNDI providers, SMTP servers, proxy servers, and UDDI registries. The project includes a folder for each type of global resource. You cannot create any projects or folders in the System folder, only global resources. The global resources you create in the System project can be used by the resources in any of the Service Bus projects you create.

Figure 2-3 System Project in the Console

Projects and Folder Names

Project names and folder names are limited to 64 characters and must follow the guidelines list in [Naming Guidelines for Service Bus Components](#).

The names, length, and levels of nesting of projects and folders are ultimately affected by the limits of your operating system. Creating folders or projects with very long names or deeply nesting folders can fail due to the limitations of the operating system.

Qualified Resource Names Using Projects and Folders

Projects and folders qualify the names of Service Bus resources. A reference to a resource is constructed as follows:

```
project-name/folder/.../subfolder/resource-name
```

Service Bus Resources

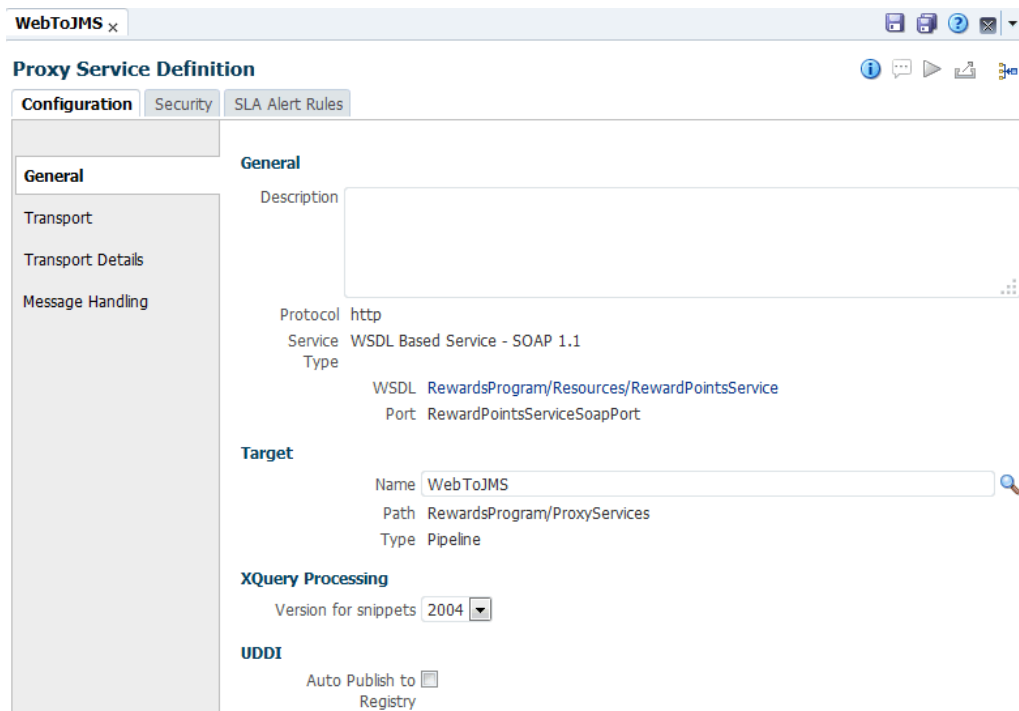
The components you create in Service Bus are known as *resources*. Resources can include proxy and business services that defines the endpoints of the service, and pipelines and split-joins that define the message processing logic for the service. Resources also include supporting resources, such as service accounts, WSDL documents, JNDI providers, MQ connections, and so on. Resources are stored in the projects and folders in which they are created. Some resources can be shared among different projects.

Oracle Service Bus Console Editors

Each type of resource can be configured using an editor in the console that is specific to that type of resource. When you open a resource from the Project Navigator, that resource's definition editor appears, and you can configure the properties for the resource. The definition editors have a standard set of tools in the upper right toolbar, including Save, Save All, Help, Close, and Close All. In addition, each editor includes

tools specific to the resource being configured. These tools allow you to perform additional tasks, like launch the Test Console, launch the Pipeline Editor, export a WSDL file, view references, and so on. The following figure shows the Proxy Service Definition Editor.

Figure 2-4 Proxy Service Definition Editor



Getting Started

To get started using the Oracle Service Bus Console, you need to have a web browser and a login ID and password.

- [How to Access the Oracle Service Bus Console](#)
- [How to Exit the Oracle Service Bus Console](#)

How to Access the Oracle Service Bus Console

You access the Oracle Service Bus Console from a web browser.

To start the Oracle Service Bus Console:

1. Start the Service Bus domain.
2. When the server starts, enter one of the following URLs in your browser:

`http://hostname:port/servicebus`

or, for SSL:

`https://hostname:port/servicebus`

where `hostname` represents the name of the Service Bus Admin Server and `port` represents the port number on which it is listening.

The login page appears.

3. Enter the user name and the password you specified during installation.

Note:

The user name and password for the Service Bus Examples domain is `weblogic/welcome1`.

4. Click **Login**.

How to Exit the Oracle Service Bus Console

To log out of the Oracle Service Bus Console, click **Logout** on the banner near the upper right.

Working with Sessions

When you create and modify Service Bus resources in the console, you do so within the context of a session. When you are ready to test your resources, you activate the session to promote them to the runtime. You must save all changes before activating or exiting a session.

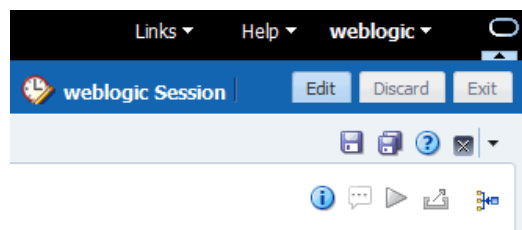
To view a history of sessions, see [How to View the Existing Sessions](#).

- [How to Create a Session](#)
- [How to Activate a Session](#)
- [How to Exit a Session](#)

How to Create a Session

Before making any changes to your Service Bus projects in the Oracle Service Bus Console, you must start a new session or edit an existing session using the buttons in the Sessions toolbar. These buttons change depending on the state of the session. If there is no active session, the **Edit** button in the figure below is **Create**. If you are working in a session, the Edit button is **Activate**.

Figure 2-5 Sessions Toolbar



To create a session:

1. Do one of the following:
 - To begin a new session, click **Create**.
 - To modify an existing session, click **Edit**.

The name of the session appears in the upper right.

2. Make your changes within the session.

3. For each component you change, click **Save**.
4. To discard the changes you made in the current session, click **Discard**.
5. When you are done making changes, activate the session as described in [How to Activate a Session](#).

How to Activate a Session

Creating a session and discarding a session proceeds regardless of other activity in the system. However, if another session is in the process of being activated, an error occurs indicating the user that has the pending WebLogic Server changes.

You cannot activate a session until you have resolved all error conflicts. You can activate a session that has only warnings. For information on resolving conflicts, see [Viewing and Resolving Conflicts](#).

Note:

When you try to activate a session with a JMS endpoint URI on another server (a single server other than the one on which you are working or a Managed Server in a cluster), ensure that the destination server is available. Service Bus does not allow registration of proxy services with the JMS transport if the JMS endpoint URL specifies a destination that is unreachable. In other words, for JMS services, Service Bus checks if the specified connection factory exists; if it does not, a session activation error occurs.

To activate a session:

1. Once you finish making changes for a session, click **Activate**.
2. If there are no validation errors, skip to [step 5](#).
3. If there are validation errors, an error message appears. View and fix configuration conflicts before you proceed.

If new conflicts arise while you view the existing conflicts, a message appears informing you of the new conflicts.
4. Once all conflicts are resolved, click **Activate** again.
5. In the **Description** field, enter a brief description to identify the session. This is displayed in the **Description** column when you to view the history of configuration changes caused by session activations.
6. To complete the activation, click **Activate**. If no new conflicts have arisen in the interim, the session ends and the configuration is deployed to the runtime.

How to Exit a Session

Exiting a session does not end the session or activate any changes to the runtime. After exiting, the console displays the core configuration that is active in the runtime state. Any changes you made during the session that were not activated are not shown. This behavior also applies if you log out of the console or close your browser. The session and all changes that you have made in the session persist even if you log out or the server is restarted. If you return to the session by clicking **Edit**, your previous changes reappear and you can continue making changes.

The session ends only after it has been activated or discarded. See [How to Activate a Session](#).

To exit a session:

- Click **Exit** at any time to exit the session.

Working with Projects, Folders, and Resources in Oracle Service Bus Console

Service Bus resources can be organized into separate projects, which are non-hierarchical, disjointed, top-level grouping constructs. All resources (such as services, WSDL files, XQuery transformations, and so on) reside in exactly one non-overlapping project.

You create resources directly under a project or in folders within the project to organize them further. Each folder contains a set of resources and can also contain additional folders, like directories in a file system with the project level being the top-level directory. Resources located in one project can reference and use resources that are defined in other projects. You can move resources between projects or folders, and rename or delete them. Service Bus also lets you clone a resource, project, or folder to create a copy of that resource with the specified target identity. Cloning a Service Bus component copies all artifacts in the project or folder to a different location. Service Bus preserves dependencies when resources are renamed or moved, and also adjusts any references to a renamed or moved resource.

How to Locate Services

You can find services in the Oracle Service Bus Console by navigating through the projects and folders in the Project Navigator, or you can perform a search for the service you want to find.

Note:

Searches accept wildcard characters. Use an asterisk (*) to represent multiple unknown characters, and use a question mark (?) to represent a single unknown character. The search is case-sensitive.

To search for services in any project:

1. In the **Search** field above the Resources tab, enter the name of the proxy service, business service, pipeline, or split-join you want to find, and then click **Search** (the right arrow icon).

The results of your search appear on the **Search Results** tab in the lower portion of the page. You might need to click the up arrow to the right of the tabs to view the results.

Figure 2-6 Service Search Results Tab

Name	Path	Type	Modified By	Date
JMSRequestReply	RewardsProgram/BusinessS...	Business Service	weblogic	5/30/2014 7:27 PM
JMS_write	RewardsProgram/TestServi...	Business Service	weblogic	5/30/2014 7:27 PM
JMSResponder	RewardsProgram/TestServi...	Proxy Service	weblogic	5/30/2014 7:27 PM
JMSResponder	RewardsProgram/TestServi...	Pipeline	weblogic	5/30/2014 7:27 PM

2. To view the results in a full page, click **Detach**.
3. To change your view options, see [How to Customize Table Views](#).

Working with the Project and Folder Definition Editors

Use the Project and Folder Definition Editors to view information about the projects, folders, and resources in your Service Bus session and in the configuration framework. From these editors, you can view each project in the session, each folder and resource in a project, and each resource in a folder. You can also take actions against certain objects, such as testing a resource, launching the Pipeline Editor, and so on. The available actions depend on the type of resource you are viewing.

About Viewing Project, Folder, and Resource Information

The Project and Folder Definition Editors let you customize the way you view information about Service Bus projects, folders, and resources. In the table views on the editors, you can show and hide columns, change the order of columns, and filter the projects, folders, and resources you see by their names.

Viewing All Projects in the Session

To view all projects in the session:

1. In the Project Navigator, click **All Projects**.

The All Project Definition Editor appears, with a list of projects displayed in the All Projects table.

Figure 2-7 All Projects Definition Editor

Name	Type	Actions
CreateAccount	Project	
default	Project	
Deployment	Project	
HttpProxy	Project	
osb-102-FileHandling	Project	
pr_jcafile	Project	
RewardsProgram	Project	
SBProject	Project	
security	Project	
Standard	Project	
System	Project	

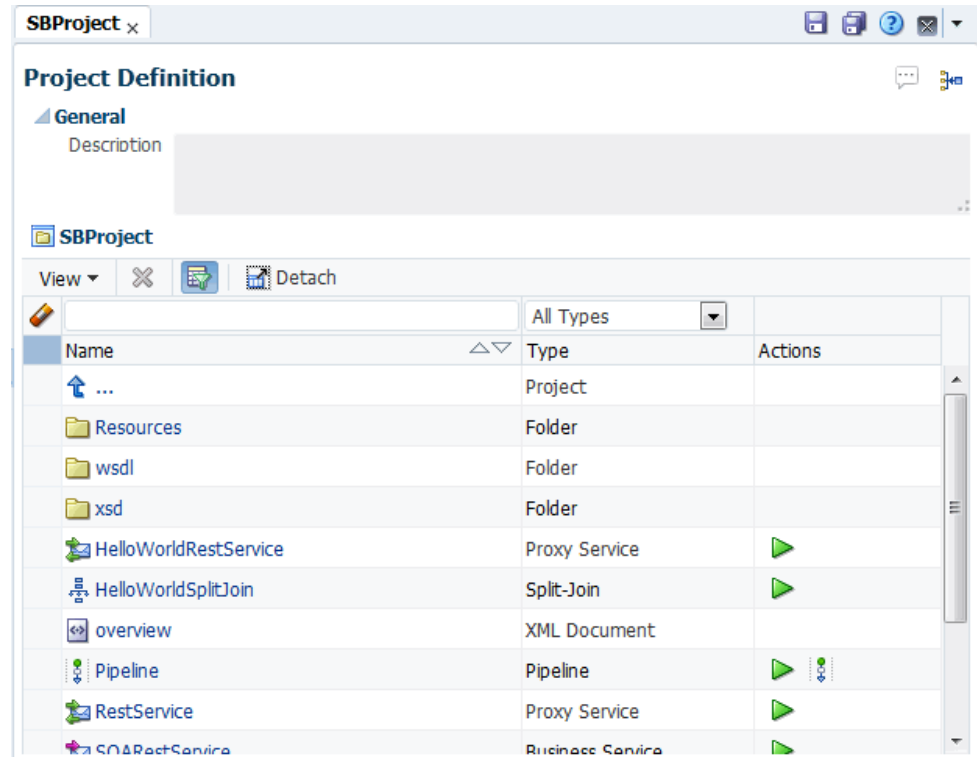
2. To filter the projects listed in the table by name, see [How to Filter Components on the Project and Folder Definition Editors](#).
3. To change the way information appears in the table, see [How to Customize Table Views](#).

Viewing Folders and Resources in a Project

To view the folders and resources in a project:

1. In the Project Navigator, click the name of the project.

The Project Definition Editor appears, with a list of each folder and Service Bus resource contained in the project you selected.

Figure 2-8 Project Definition Editor

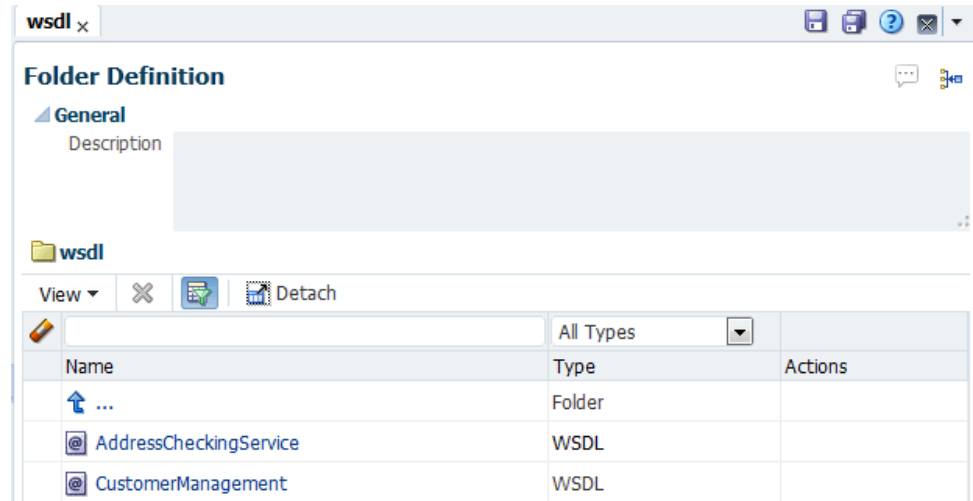
2. To filter the folders and resources listed in the table by name, see [How to Filter Components on the Project and Folder Definition Editors](#).
3. To change the way information appears in the table, see [How to Customize Table Views](#).

Viewing the Subfolders and Resources in a Folder

To view the subfolders and resources in a folder:

1. In the Project Navigator, navigate to and click the name of the folder to view.

The Folder Definition Editor appears, with a list of each subfolder and Service Bus resource contained in the folder you selected.

Figure 2-9 Folder Definition Editor

2. To filter the folders and resources listed in the table by name, see [How to Filter Components on the Project and Folder Definition Editors](#).
3. To change the way information appears in the table, see [How to Customize Table Views](#).

How to Filter Components on the Project and Folder Definition Editors

Service Bus provides a Query by Example feature that lets you filter the projects, folders, or resources displayed on the Project or Folder Definition Editor so you can view only the Service Bus components you need without having to scroll through all the components listed.

To filter components on the Project and Folder Definition Editors:

1. Launch the Project or Folder Definition Editor, as described in [About Viewing Project, Folder, and Resource Information](#).
2. If the Query by Example bar is not visible above the top row of the table, click **Query by Example** (the filter icon).
3. In the field above the **Name** column, enter the name of the project, folder, or resource you want to view, and press **Enter**.

The list displays only the components matching the name you entered.

Note:

Queries accept wildcard characters. Use an asterisk (*) to represent multiple unknown characters, and use a question mark (?) to represent a single unknown character. The query is case-sensitive.

Create New Projects and Folders for Resources

Resources in the console are grouped in projects. You must create a project before you can create Service Bus resources. You can create folders within projects and within other folders to organize the project components into logical groupings.

Creating a Project in the Project Navigator

To create a project:

1. In the Project Navigator, right-click **All Projects**.
2. Select **Create > Project**.

The **Create a New Project** dialog appears.

3. Enter a unique name for the project, and optionally add a description.

For information about naming guidelines, see [Projects and Folder Names](#).

4. Click **Create**.

The new project appears in the Project Navigator and the Project Definition Editor.

Creating a Folder in the Project Navigator

To create a folder:

1. In the Project Navigator, right-click the project or folder to which you want to add the new folder.
2. Select **Create > Folder**.

The **Create a New Folder** dialog appears.

3. Enter a unique name for the folder, and optionally add a description.

4. Click **Create**.

The new folder appears in the Project Navigator and the Folder Definition Editor.

How to Clone Projects, Folders, and Resources

Cloning a project or folder copies all resources in a project or folder to a different location. You can also clone individual resources to copy them to a different folder or project. Service Bus preserves dependencies when an object is cloned, and also adjusts any references.

To clone projects, folders, and resources:

1. In the Project Navigator, navigate to the project, folder, or resource you want to clone and right-click it.

2. Click **Clone**.

The Clone dialog appears.

3. Optionally, in the **New <Resource> Name** field, enter a new name for the component.

4. In the Destination section, select the project or folder to which you want to clone the selected component.

5. Click **Clone**.

The cloned component appears in the location you selected. Any dependencies are retained.

What Happens When You Clone a Project

Depending on the name and destination of the new project, the project can be demoted to a folder, merged with an existing project, or merged with an existing folder. If you clone a project into the root location, you must enter a new name for the cloned component. You can clone a project in the following ways. In all cases, the original project remains unchanged.

- Clone the project with a new name at the All Projects level. The original project and its clone exist as peer projects, and the cloned project contains the same folders and resources as the original.
- Clone the project using the name of an existing project other than the source project. The folders and resources from the source project are merged with the folders and resources of the project whose name was given to the clone.
- Clone the project in a new location (in an existing project or project folder) with a unique name. The cloned project is demoted to a folder in the new location and it contains the same folders and resources as the original project.
- Clone the project in a new location (in an existing project or project folder) with the same name as an existing folder. The contents (subfolders and resources) of the project are merged with the contents of the target folder in the target folder.

What Happens When You Clone a Folder

Depending on the name and destination of the new folder, the folder can be converted to a project, merged with an existing project, or merged with an existing folder. If you clone a folder into its current location, you must enter a new name for the cloned component. You can clone a folder in the following ways. In all cases, the original folder remains unchanged.

- Clone the folder with a new name at the same location as the original folder. The original folder and its clone exist as peers, and the cloned folder contains the same subfolders and resources as the original.
- Clone the folder using the name of an existing folder other than the source folder. The subfolders and resources from the source folder are merged with the subfolders and resources of the folder whose name was given to the clone. The original folder remains unchanged.
- Clone the folder in a new location, in an existing project or project folder. In this case, the cloned folder contains the same folders and resources as the original folder. The original folder remains unchanged.

How to Rename Projects, Folders, and Resources

When you rename Service Bus components, any references are automatically updated.

Note:

If you rename a business service imported from the UDDI registry, the service will become detached from the registry.

To rename a Service Bus component:

1. In the Project Navigator, navigate to the component whose name you want to change.
2. Do one of the following:
 - Right-click the component and select **Rename**.
 - Select the component to rename, and then click the component again.
3. Select the text of the component's name, and enter the new name. For naming guidelines, see [Projects and Folder Names](#).
4. Press **Enter**.

The component is renamed.

How to Move Projects, Folders, and Resources

When you move resources to a different location, Service Bus retains any dependencies against that resource. If you move a project into another project, Service Bus converts it to a folder within the second project. Conversely, if you move a folder into the All Projects node, Service Bus converts it to a project.

To move a Service Bus component:

1. In the Project Navigator, navigate to the project, folder, or resource you want to move and right-click it.
2. Click **Move**.

The Move dialog appears.

3. In the Destination section, select the project or folder to which you want to move the selected component.
4. Click **Move**.

The moved component appears in the location you selected. Any dependencies are retained.

Note:

You cannot move a component to a project or folder that already contains a component by the same name.

How to Delete Projects, Folders, and Resources

When you delete a project or folder, all resources under the folder are deleted. If any resources under this folder are referenced by resources under a different project or folder, you can still delete it but with a warning confirmation. This might result in conflicts due to unresolved references to the deleted resource.

You can delete Service Bus components from the Project Navigator or from the Project or Folder Definition Editor. When you delete from an editor, you can delete multiple components in the same container at once.

Caution:

If you delete a project or folder that contains a pipeline template resource, all the concrete pipelines derived from that template are unlinked.

Deleting a Service Bus Component using the Project Navigator

To delete a Service Bus component using the Project Navigator:

1. In the Project Navigator, navigate to the component you want to delete.
2. Right-click the component and select **Delete**.
3. On the Confirmation dialog that appears, click **Yes** to complete the process.

The component is removed from the Project Navigator and from the session.

Deleting a Service Bus Component Using an Editor

To delete Service Bus components using an editor:

1. In the Project Navigator, navigate to the project or folder containing the components you want to delete.
2. Click the project or folder to display its editor. To delete projects, click the **All Projects** node instead.

The Project or Folder Definition Editor appears.

3. In the components table, select each component you want to delete, and then click the **Delete** icon.

Tip:

To select multiple components, hold down the **Ctrl** key and click each component to delete.

4. On the Confirmation dialog that appears, click **Yes** to complete the process.

The components are removed from the editor, the Project Navigator, and the session.

Viewing and Resolving Conflicts

The Conflicts tab at the bottom of each page displays diagnostic messages about errors in your configuration, along with any conflicts between changes made in your session and other activated sessions. A conflict occurs if there is a semantic error in a Service Bus resource or if a resource modified in the current session has already been modified and activated in another session. Two changes to the same resource by two sessions do not cause a conflict until one of the sessions is activated. You can view and resolve conflicts using the Conflicts tab at the bottom of the page.

The **Conflicts** icon in the toolbar displays the number of live conflicts in the session.

Figure 2-10 Conflict Icon in the Toolbar



The Conflicts tab displays the following sets of information, depending on the nature of the conflicts:

- **Errors:** An Error icon denotes non-committable, critical conflicts within your configuration. You cannot commit your changes without resolving the conflicts.
- **Concurrent Updates:** A Warning icon denotes committable, non-critical conflicts. These warn you of incompatible changes with other activated sessions.
- **Informational Messages:** A Warning icon denotes committable, non-critical conflicts within your configuration.

How to View Conflicts and Errors

The Conflicts tab displays errors in your current Service Bus configuration and reflects any conflicts your changes have with other activations. You can view all conflicts, or just those for the resource displayed in the current resource editor.

Figure 2-11 Conflicts Tab

Name	Path	Type	Message
Pipeline	BusServProject	Pipeline	1 Error(s)
ProxyService	BusServProject	Proxy Service	2 Error(s)
ps_filePipeline	BusServProject	Pipeline	2 Error(s)
PxytoBix	BusServProject	Proxy Service	1 Error(s)
SplitJoin	BusServProject	Split-Join	2 Error(s)
SplitJoinProxyService	BusServProject	Proxy Service	1 Error(s)

Viewing All Conflicts and Errors in the Service Bus Console

To view all conflicts and errors:

1. Access the Conflicts tab by doing one of the following:
 - Click the **Errors** icon in the uppermost toolbar.
 - Click the **Conflicts** tab at the bottom of the page.
2. To view errors in your configuration click **Errors** on the Conflicts tab.
3. To view conflicts with concurrent updates in other activations, click **Concurrent Updates** on the Conflicts tab.
4. To display details of a specific conflict, click the link in the **Name** column for that conflict.

Viewing Conflicts and Errors for a Deployed Resource

If a Service Bus resource contains errors, when you display that component in its editor, a conflict icon appears next to the name of the editor.

To view conflicts and errors for the displayed resource:

1. With a resource displayed in its editor, click the **Conflicts** icon next to the editor title (for example, next to Business Service Definition).

The first conflict is highlighted in the Conflict tab at the bottom of the page.

2. Expand the conflict to view additional information about the error that caused the conflict.
3. To scroll through the conflicts for a component, click the left and right arrows next to the **Conflicts** icon.

How to Resolve Conflicts and Errors

You must resolve all error conflicts in a session before you can activate that session. To resolve a conflict, use the information provided in the Errors and Concurrent Updates tables on the Conflicts tab to understand the problem and then modify the component that is causing the conflict.

Resolving Concurrent Update Conflicts

If you have a conflict that occurs because a resource was modified in the current session and it was already modified and activated in another session, you can resolve the conflict in one of two ways.

- To save your changes to the runtime and override the changes deployed in the conflicting session, click **Activate** in the upper right toolbar.

The changes activated by the conflicting session's user are overwritten by your changes in this session.

- To restore a component in this session to the state in which it was saved in the runtime, select the component in the Concurrent Updates table and click **Synchronize** above the table.

The resource in your session is updated with the changes activated in the conflicting session. You can then make your updates and activate your changes.

Resolving Error Conflicts

To resolve error conflicts:

1. In the Errors table of the Conflicts tab, click the **Expand** icon to the left of the resource you want to resolve.

The error message appears below the resource.

2. Click the name of the resource to open it in a Service Bus editor.
3. Update the resource to fix the issue based on the information provided in the error message.
4. Click **Save**.
5. Repeat these steps until you resolve all conflicts.

Viewing Historical Data

The History tab has three different views: Changes, Sessions, and Activations.

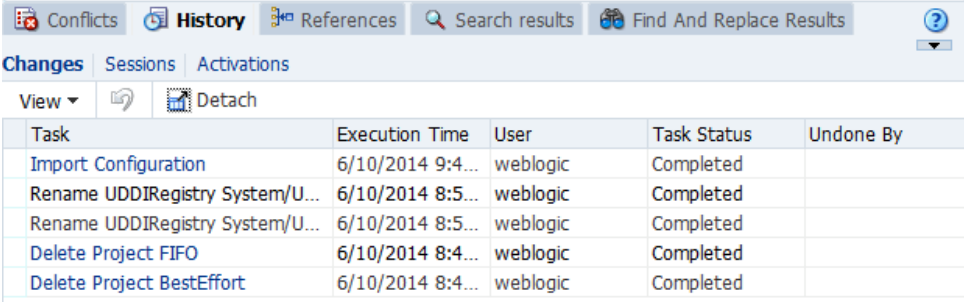
- [How to View the Changes in the Current Session](#)
- [How to View the Existing Sessions](#)

- [How to View the Changes in an Activated Session](#)
- [How to Purge Activated Sessions](#)

How to View the Changes in the Current Session

The Changes view of the History tab displays different information based on whether or not you are in a session. When you are in a session, the Changes view displays a list of configuration changes that you have made during the current session. When you are outside a session, the Changes view does not display any changes.

Figure 2-12 History Tab



Task	Execution Time	User	Task Status	Undone By
Import Configuration	6/10/2014 9:4...	weblogic	Completed	
Rename UDDIRegistry System/U...	6/10/2014 8:5...	weblogic	Completed	
Rename UDDIRegistry System/U...	6/10/2014 8:5...	weblogic	Completed	
Delete Project FIFO	6/10/2014 8:4...	weblogic	Completed	
Delete Project BestEffort	6/10/2014 8:4...	weblogic	Completed	

To view the changes in the current session:

1. Click the **History** tab at the bottom of the page.

The History tab appears with the Changes view selected. The Changes table displays the type of change made, the time each change was made, the person who made each change, the status, and, if the change was reversed, the person who reversed it.

2. To undo any of the changes listed, see [How to Undo Specific Changes in the Current Session](#).

How to View the Existing Sessions

The Sessions view of the History tab displays a list of all existing sessions within the Oracle Service Bus Console. You can view these sessions if you are currently in a session or outside a session.

You only view all sessions if you are logged in with an Administration role. For more information, see "Defining Access Security for Oracle Service Bus" in *Administering Oracle Service Bus*.

To view the existing sessions:

1. Click the **History** tab at the bottom of the page.

The History tab appears with the Changes view selected.

2. Click **Sessions** above the table.

The Sessions table appears with a list of sessions, the user who created each session, the date and time each session was created and modified, and the number of resources that were changed.

3. To switch to a session in the list, select that session and click the **Switch to Session** icon.

Note:

The same user logged in to multiple browsers is not supported. It causes unpredictable behavior in the console.

How to View the Changes in an Activated Session

The Activations view of the History tab displays a list of all Service Bus activations. For each activation listed, you can view a history of changes that were activated.

To view the changes in an activated session:

1. Click the **History** tab at the bottom of the page.

The History tab appears with the Changes view selected.

2. Click **Activations** above the table.

The Activations table appears with a list of activations, their descriptions, the time each activation occurred, the user who activated each session, the status of the activation, and, if the activation was reversed, the user who reversed it.

3. To view information about each change in an activation, click that activation in the **Activation** column.

The Task Details dialog appears with a list of each change made in the selected session.

4. To view any of the resources or locations (paths) listed in the task details, click the name of the resource or location.

The resource or location opens in a Service Bus editor.

5. To undo activations or purge tasks for an activation, see the following topics:

- [How to Purge Activated Sessions](#)
- [How to Undo a Session Activation](#)
- [How to Undo Specific Changes in the Current Session](#)

How to Purge Activated Sessions

You can purge all sessions activated or only those over a specific period, delimited by start and end dates. This action can only be performed outside a session.

Caution:

Purging session activation history involves deleting data that enables multiple levels of undo. If you purge session activation history for a specific period, you will not be able to undo sessions activated during that period.

To view the changes in an activated session:

1. Make sure you are not in an active session.

2. Click the **History** tab at the bottom of the page.

The History tab appears with the Changes view selected.

3. Click **Activations** above the table.

4. Click the **Purge Tasks** icon above the table.

The Purge Activation History dialog appears.

5. On the Purge Activation History dialog, do one of the following:

- To purge all tasks for all sessions, select **Purge All Tasks**.
- To purge tasks for a specific time period, select **Purge Selected Tasks**, and enter the beginning and ending dates and times for the period to purge.

6. Click **Purge**, and then click **Yes** to confirm the purge.

The selected activations are removed from the list, and a new entry appears with a description of the purge.

Undoing Changes and Activations

When you are working in a session, you can undo tasks in any order. The undo operation sets the configuration of a resource to its configuration prior to the change you are undoing. If the task being undone was one that created a resource, there is no previous state to which the resource can be returned; in other words, no resource existed before this task was performed. Effectively, the undo operation deletes the resource from the session. In this case, errors occur if there are any reference to the resource being deleted. You can view such errors on the Conflicts tab. Service Bus supports unlimited undo operations. This means you can even reverse undo operations.

When you are not working in a session, you can view a history of sessions that were previously activated. You can also undo those sessions. The system does not allow you to undo a session that was previously activated if an error in the runtime configuration would result from the undo action. For example, if you attempt to undo a session activation that results in the removal of a resource that is being referenced by another resource, that undo action is not allowed.

How to Undo Specific Changes in the Current Session

In the Changes view of the History tab, you can undo specific tasks that you performed during your current session. You can undo any change in the current session. However if you realize you performed an undo operation in error, you can undo that action as well.

To undo specific changes in the current session:

1. Click the **History** tab at the bottom of the page.

The History tab appears with the Changes view selected.

2. Select the change you want to undo, and then click the **Undo** icon above the table.

The change is reversed. If you reversed a Create task, the selected resource is removed from the session and no longer appears in the Project Navigator. If you

reversed a Delete task, the selected resource is added back to the session and reappears in the Project Navigator.

How to Undo a Session Activation

In the Activations view of the History tab, you can undo session activations. If semantic errors result from undoing a session activation, you are prevented from undoing an activation. The alternative is to undo the session activation and have the changes put into a new session. You can then fix the semantic errors and activate the new session. You can also use this capability of undoing into a session to explore the ramifications of undoing a session activation. You can examine all the changes that result, and determine whether to undo the activation. Service Bus lets you undo multiple levels of session activation, constrained only by your system resources.

To undo a session activation:

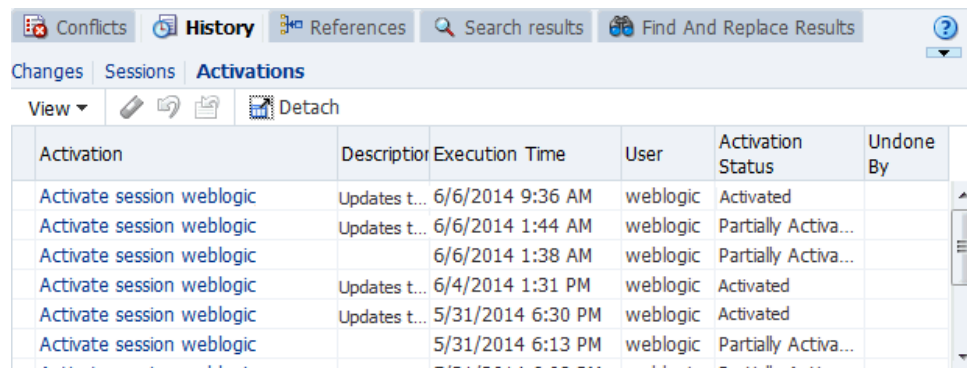
1. Click the **History** tab at the bottom of the page.

The History tab appears with the Changes view selected.

2. Click **Activations** above the table.

The Activation table displays the previous activations.

Figure 2-13 Activations Tab



Activation	Description	Execution Time	User	Activation Status	Undone By
Activate session weblogic	Updates t...	6/6/2014 9:36 AM	weblogic	Activated	
Activate session weblogic	Updates t...	6/6/2014 1:44 AM	weblogic	Partially Activa...	
Activate session weblogic		6/6/2014 1:38 AM	weblogic	Partially Activa...	
Activate session weblogic	Updates t...	6/4/2014 1:31 PM	weblogic	Activated	
Activate session weblogic	Updates t...	5/31/2014 6:30 PM	weblogic	Activated	
Activate session weblogic		5/31/2014 6:13 PM	weblogic	Partially Activa...	

3. To undo an activation, do the following:

- Click the empty field to the right of the session you want to undo to select it.
- Click the **Undo** icon above the table.

The session activation is reversed.

4. To undo an activation into a session, do the following:

- Click the empty field to the right of the session you want to undo to select it.
- Click **Undo in a Session**.

The session activation is reversed and a new session is created pre-populated with the tasks needed to undo the changes activated in the session.

Viewing References

The References tab displays information about resources that either reference other resources or are referenced by other resources. For projects and folders, the References tab lists the following:

- Resources outside of the current project or folder that are referenced by resources inside the project or folder.
- Resources outside of the current project or folder that reference resources inside the project or folder.

For any project, folder, or resource in the console, click the References icon in the upper right on the editor to view information about these references on the References tab.

Viewing Resource References

You can view references to resources whether or not you are in a session.

To view resource references:

1. Open a project, folder, or resource in its Service Bus editor.
2. Click the **References/Referenced By** icon in the upper right portion of the editor.

The References page appears, and shows the selected resource in bold. The **Referenced By** column lists any resources that reference the selected resource. The **References** column lists any resources that the selected resource references.

Figure 2-14 References Tab

Referenced By	Resource	References
← SplitJoin	wsdl	Customer →

3. Click the name of a resource to display it in its editor.
4. Click the left arrow next to a resource in the **Referenced By** column to view the references for that resource.
5. Click the right arrow next to a resource in the **References** column to view the references for that resource.

Customizing the Appearance of the Oracle Service Bus Console

Several editors in the console display information in tables. You can customize how tables display this information.

- [How to Customize Table Views](#)

How to Customize Table Views

The menu bar above each table lets you specify which columns to display and in what order. You can also specify the sort order for the columns and view the table full-screen.

Specifying the Columns to Display

To specify the columns to display:

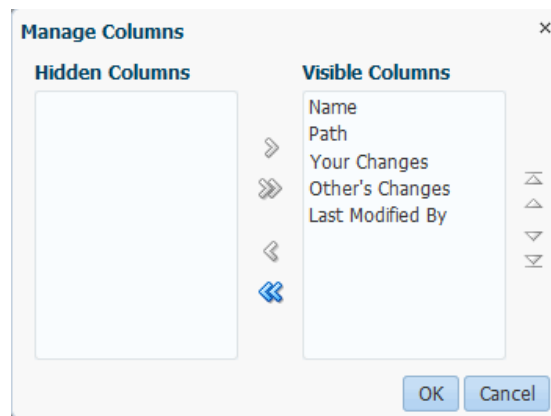
1. In the upper left of the table, click **View** and select **Columns**.

A sub-menu appears, displaying a list of available columns. A check next to a column indicates that it is visible.

2. To display all available columns, select **Show All**.
3. To specify which columns to display, select **Manage Columns**.

The Manage Columns dialog appears.

Figure 2-15 *Manage Columns dialog*



4. To hide a displayed column, select the column name in the Hidden Columns panel and click the right arrow button.
5. To display a hidden column, select the column name in the Visible Columns panel and click the left arrow button.

An asterisk denotes a required column.

6. Repeat until you have listed all the column names you want to display in the **Visible Columns** field.

In the **Visible Columns** field, you can use the up and down arrows to reorder the column names.

7. Click **OK**.

Sorting the Columns in a Table

To sort the columns in a table:

Note:

The default sort order for any table is determined by the first column in the table.

1. In the upper left of the table, click **View** and select **Sort**.
2. Select **Advanced Sort**.
3. In the **Sort By** field, select the column name by which to sort first, and then select either **Ascending** or **Descending**.
4. In the **Then By** field, select the column name by which to sort second, and then select either **Ascending** or **Descending**.
5. Click **OK**.

Reordering Columns in a Table

To reorder the columns in a table:

1. In the upper left of the table, click **View** and select **Reorder Columns**.
2. On the Reorder Columns dialog, use the up and down arrows to change the order of the columns.
3. When you are done, click **OK**.

Tip:

You can also move a column by clicking on the column heading and dragging it to a new location in the table.

Viewing a Table in Full-Screen Mode

To view the table full-screen:

To detach the table from its current page and view it on a page the size of your browser, click **View** above the table and select **Detach**.

Getting Started with Oracle Service Bus in JDeveloper

This chapter provides general information about how to use Service Bus in JDeveloper to configure services and other service bus resources. These are design-time activities that, except where noted, do not require a running WebLogic Server instance.

This chapter includes the following topics:

- [JDeveloper Concepts for Service Bus](#)
- [Managing Service Bus Components in JDeveloper](#)
- [Creating Service Bus Applications and Projects in JDeveloper](#)
- [Refactoring Service Bus Projects, Folders, and Resources](#)

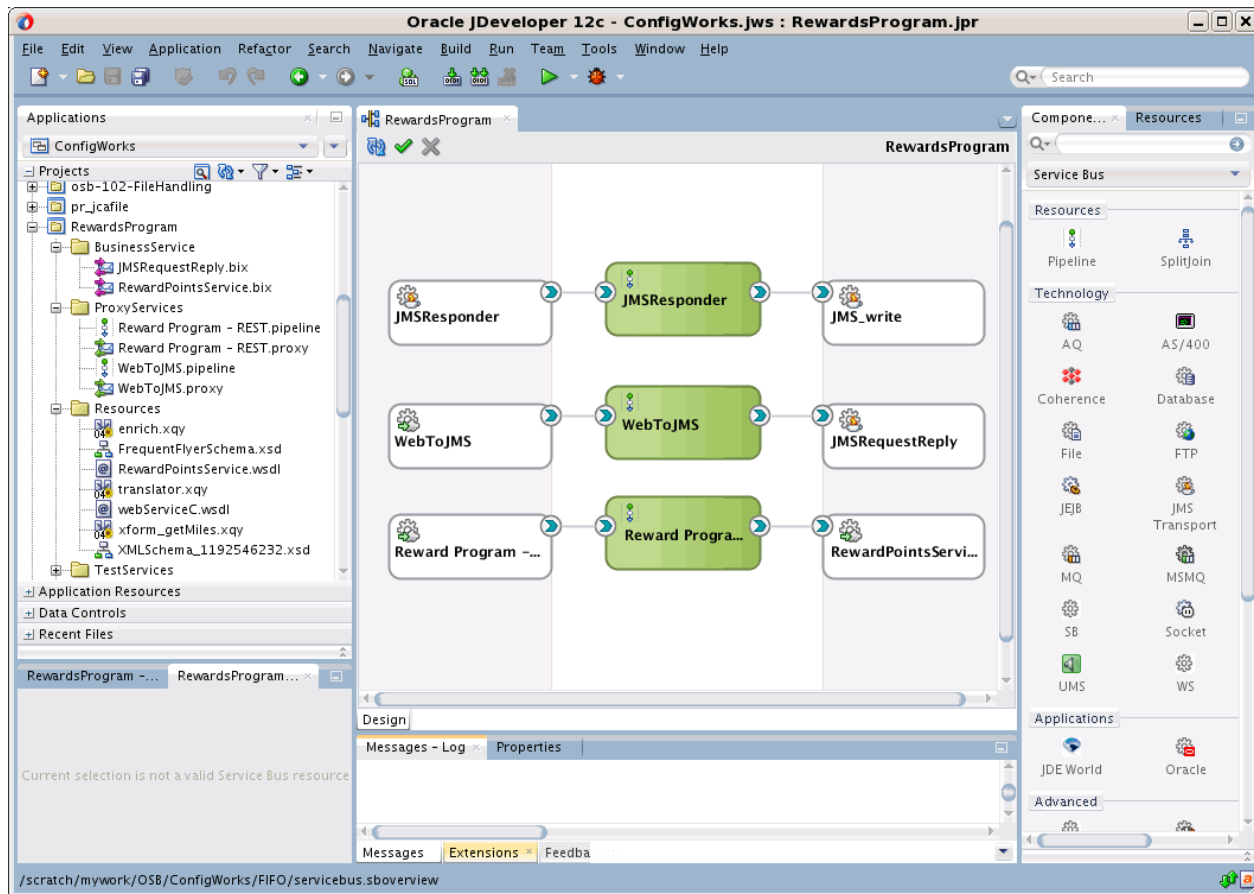
For complete information on using JDeveloper, see *Developing Applications with Oracle JDeveloper*.

JDeveloper Concepts for Service Bus

Service Bus uses editors, wizards, and dialogs in JDeveloper to create and configure Service Bus applications. Some of these are specific to Service Bus components, and some are standard JDeveloper tools. For example, proxy service and business service editors are specific to Service Bus, but XSLT and XQuery mappers are shared by other products in JDeveloper. Service Bus shares many features with SOA Suite components in JDeveloper.

The following figure shows Service Bus artifacts in JDeveloper, including project files in the application navigator, the Service Bus Overview Editor in the center, and the Components window on the right. The Service Bus Overview Editor is where you define services and message flows. The Components window lists all the different Service Bus and SOA Suite components you can drag onto the Overview Editor design canvas.

Figure 3-1 Oracle Service Bus in JDeveloper



Application Navigator

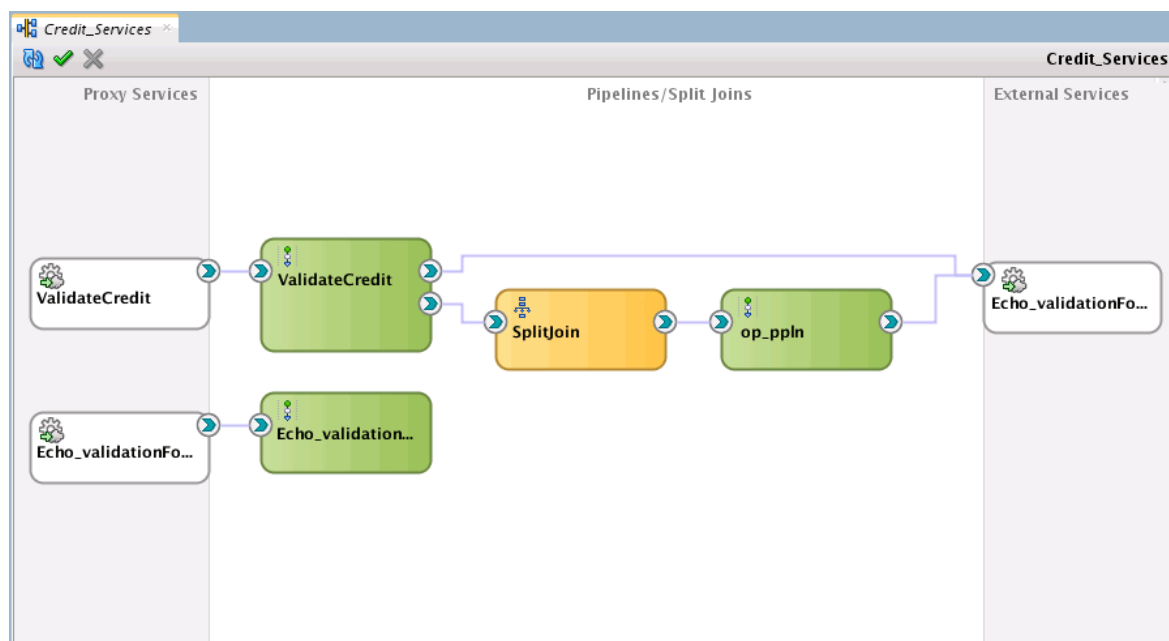
The Application Navigator displays the key files for all the resources and services included in the Service Bus project, which can include the following:

- An XML file that is automatically created when you create a Service Bus project. This file describes the entire Service Bus application, including services, resources, references, and wires. In the Application Navigator, this file has the same name as the project. In the file system, it is named `servicebus.sboverview`.
- Any proxy service files (`service_name.proxy`).
- Any business service files (`service_name.bix`).
- Pipeline files (`pipeline_name.pipeline`).
- Split-join files (`splitjoin_name.flow`).
- Any resource files, such as service accounts, WSDL files, service key providers, alert destinations, and so on.
- Additional subfolders for class files, XSD files (schemas), and XSL files (transformations).

Service Bus Overview Editor

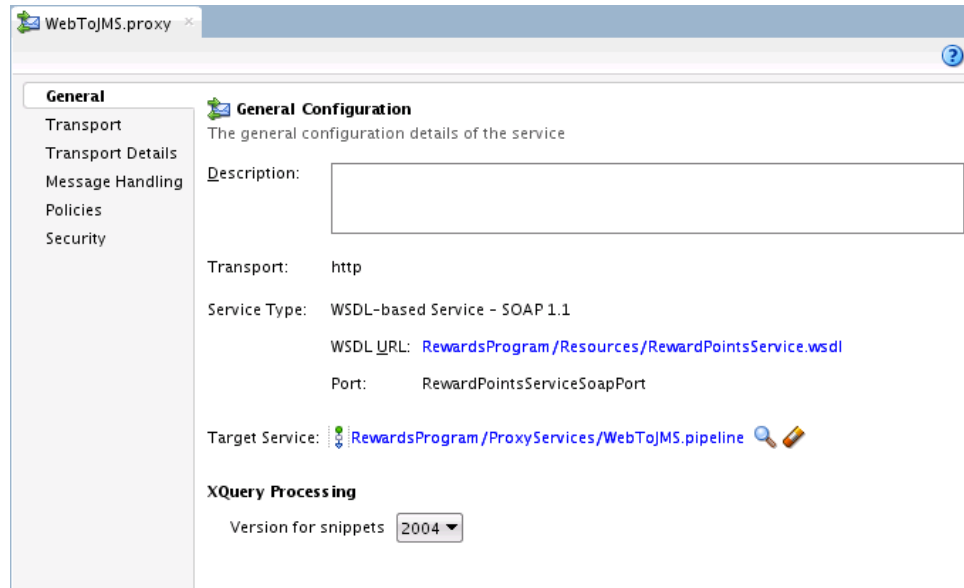
The Service Bus Overview Editor lets you design your Service Bus application from the top down in a graphical view of the components included in the application. You drag pipelines, split-joins, transports, adapters, and bindings from the Components window into the designer window of the editor. When you drag and drop a component into the designer window, a corresponding wizard appears so you can perform basic configuration tasks related to that component. For example, when you drag and drop a pipeline to the Pipelines/Split Joins section of the editor, the Create Pipeline Service wizard appears. After you complete the wizard, the component appears in the editor, and you can double-click the component to open its editor and further define its configuration. The following figure shows the Overview Editor with a mixture of services, pipelines, and split-joins.

Figure 3-2 Service Bus Overview Editor



Resource Editors

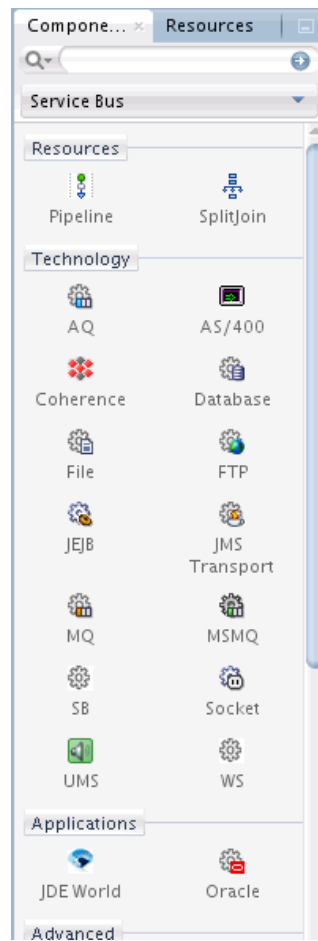
Each type of Service Bus resource can be configured in an editor that is specific to that resource. Most editors have multiple views, which you select in the lower left corner of the editor. The Configuration view is available for most resources, and is where you do most of the resource configuration. Design view is available for the Service Bus Overview Editor, pipelines, and split-joins, and provides a graphical representation of how messages are processed. In design mode, you can drag and drop activities from the Components window directly into a split-join or pipeline. For some resources, you can view the source code using the Source view. Several editors also include a History view, where you can look at a history of changes to the displayed resource. The following figure shows the General configuration page of the Proxy Service Definition Editor.

Figure 3-3 Proxy Service Definition Editor in JDeveloper

Components Window

The Components window appears when the Service Bus Overview Editor, the Pipeline Editor, or the Split-Join Editor is open. The Components window lists the various components that you can use in a Service Bus application, pipeline, or split-join, depending on which editor is visible. For the overview editor, the Components window displays the components you can add to the application, including resources, adapters, and transports. For pipelines and split-joins, the Components window displays routing, transformation, and error handling actions that you can use to define the flow of data between services.

The elements listed in the Components window can be dragged and dropped from the Components window to the visible editor. If the Components window is not visible, select **Components** from the Window main menu. The following figure shows the Components window for the Service Bus Overview Editor.

Figure 3-4 Service Bus Components Window

Resources Window

When you select an item in the Resources window, a dialog appears in which you can browse both local and remote resources. For example, you can access the following resources:

- Shared local application metadata such as schemas, WSDL files, event definitions, business rules, and so on.
- WSIL browser functionality that uses remote resources that can be accessed through an HTTP connection, file URL, or Application Server connection.
- Remote resources that are registered in a Universal Description, Discover, and Integration (UDDI) registry.

If the Resources window is not visible, select **Resources** from the Window main menu.

Properties Window

The Properties window displays properties for the selected Service Bus resource or component, and lets you modify the properties for pipeline actions and split-join operations. For information about the properties you can configure, see the online help for Service Bus and the following chapters in this guide:

- [Improving Service Performance with Split-Join](#)

- [Working with Pipeline Actions in Oracle Service Bus Console](#)

If the Property Inspector is not visible, select **Property Inspector** from the Window main menu.

Structure View

The Structure window offers a structural view of the component currently displayed in the editor. Depending on the document currently open, the Structure Window lets you view data in two modes, as indicated by the tabs near the bottom of this window:

- *Source mode* displays the code structure of the file currently open in the editor. This is applicable to technologies that allow code editing, such as XML. For example, this tab will not be available when a pipeline is open for editing.
- *Design mode* displays the tree structure of the file currently open in the editor, and lets you navigate through the different nodes of the component.

The Structure window is dynamic, always tracking the current selection of the active

Log Window

The Log window displays messages about application compilation, validation, and deployment.

Managing Service Bus Components in JDeveloper

In JDeveloper, Service Bus components are contained within a Service Bus project, and Service Bus projects can be gathered into Service Bus applications. The resources are stored as files in the file system. All resources (such as services, WSDL files, XQuery transformations, and so on) reside in one non-overlapping project.

You create resources directly under a project or in folders within the project to organize them further. Each folder contains a set of resources and can also contain additional folders, like directories in a file system. Resources located in one project can reference and use resources that are defined in other projects. You can move resources between projects or folders, and rename or delete them. Service Bus preserves dependencies when resources are renamed or moved, and also adjusts any references to a renamed or moved resource.

Creating Service Bus Applications and Projects in JDeveloper

In JDeveloper, Service Bus resources are organized into projects, which can be further divided into folders. Service Bus projects are grouped into Service Bus applications.

Note:

To create and deploy Service Bus applications and projects in JDeveloper, you must install the Service Bus extension. For instructions on installing this extension for JDeveloper, see *Enabling Oracle JDeveloper Extensions in Installing Oracle JDeveloper*.

Guidelines for Creating Applications and Projects

Make sure to follow these guidelines when you create Service Bus applications and projects in JDeveloper.

- When you create applications and projects, you can specify the directory in which the associated files are stored. You must create Service Bus projects in the same folder as their containing application.
- Do *not* create application or project names with spaces.
- Do *not* create applications and projects in directory paths that have spaces (for example, `c:\Program Files`).
- The combination of application and component name cannot exceed 500 characters.
- A project deployed to the same infrastructure *must* have a unique name across Service Bus applications. For example, do not perform the actions described in [Table 3-1](#). During deployment, the second deployed project overwrites the first deployed project.

Table 3-1 Restrictions on Naming a Service Bus Project

Create an Application Named...	With a Project Named...
Application1	Project1
Application2	Project1

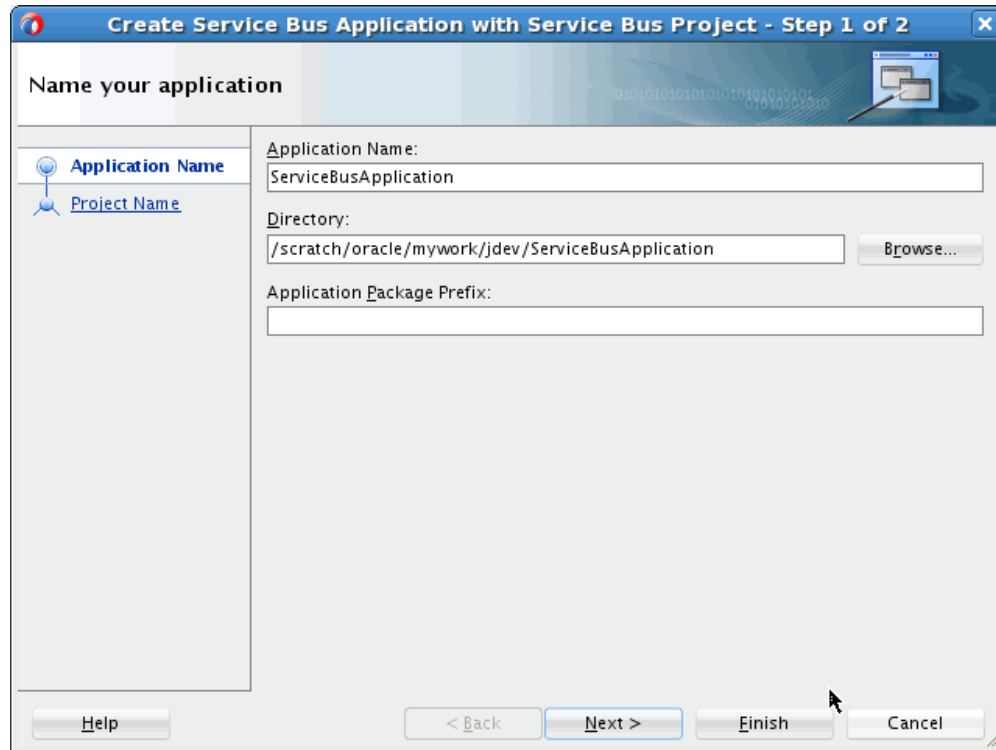
Caution:

Do not create SOA Tier components in Service Bus applications, nor Service Bus Tier components in SOA Suite applications. Mixing technologies within an application will result in errors.

How to Create a Service Bus Application and Project

The first steps in building a new application are to assign it a name and to specify the directory in which to save source files. By creating an application using application templates provided by JDeveloper, you automatically get the organization of the workspace into projects, along with the project overview file.

You can create a Service Bus application with a Service Bus project, or you can create just the application and create the project separately. The following figure shows the creation wizard when you create the application and project together.

Figure 3-5 Create Service Bus Application with Service Bus Project Wizard

Creating a Service Bus Application with No Project

Before you begin:

To avoid errors, become familiar with the information in [Guidelines for Creating Applications and Projects](#) before creating applications or projects.

To create a Service Bus application with no project:

1. Start Oracle JDeveloper Studio Edition.
2. Do one of the following to create a new Service Bus application:
 - If no applications are open, click **New Application** in the Application Navigator.
 - In the JDeveloper toolbar, click **File**, point to **New**, and select **Applications**.
 - In the JDeveloper toolbar, click **Application** and then select **New**.

The New Gallery opens, where you can select different application components to create.

3. In the **Categories** tree, select **General > Applications**.
4. In the **Items** pane, select **Service Bus Application**, and click **OK**.

The Create Service Bus Application wizard appears.

5. On the Name your Application page, optionally change the name and directory location for the application.

6. Click **Finish**.
7. From the **File** main menu, select **Save All**.
8. To add projects to your application, see [“To add a Service Bus project to a Service Bus application:”](#)

Creating a Service Bus Application and Project

Before you begin:

To avoid errors, become familiar with the information in [Guidelines for Creating Applications and Projects](#) before creating applications or projects.

To create a Service Bus application and project:

1. Start Oracle JDeveloper Studio Edition.
2. Do one of the following to create a new Service Bus application:
 - If no applications are open, click **New Application** in the Application Navigator.
 - In the JDeveloper toolbar, click **File**, point to **New**, and select **Applications**.
 - In the JDeveloper toolbar, click **Application** and then select **New**.

The New Gallery opens, where you can select different application components to create.

3. In the **Categories** tree, select **General > Applications**.
4. In the **Items** pane, select **Service Bus Application with Service Bus Project**, and click **OK**.

The Create Service Bus Application wizard appears.

5. On the Name your Application page, optionally change the name and location for the application.
6. On the Name your Project page, optionally change the name of your Service Bus project. Make sure the project directory is in the application directory (this is the default).
7. Click **Finish**.

JDeveloper adds the Service Bus project technology, the project's XML file that describes the Service Bus application, and the necessary libraries to your project.

8. From the **File** main menu, select **Save All**.

Adding a Service Bus Project to a Service Bus Application

Before you begin:

To avoid errors, become familiar with the information in [Guidelines for Creating Applications and Projects](#) before creating applications or projects.

To add a Service Bus project to a Service Bus application:

Note:

The following instructions are for the first time you create a Service Bus project. You can continue to follow these steps for future project, but a Project option will also appear when you right-click in the Application Navigator and select **New**.

1. Start Oracle JDeveloper Studio Edition.
2. Using the **Application** menu, open the Service Bus application.
3. Click **File**, point to **New**, and then select **Project**.
The New Gallery dialog appears.
4. In the **Categories** panel, scroll down to and select **Service Bus Tier**.
5. In the **Items** panel, select **Service Bus Project**, and click **OK**.
The Create Service Bus Project wizard appears.
6. On the Name your Project page, you can optionally change the name for your Service Bus project. Leave the directory at its default value, which is located in the containing application's folder.
7. Click **Finish**.
JDeveloper adds the Service Bus project technology, the project's XML file that describes the Service Bus application, and the necessary libraries to your project.
8. From the **File** main menu, select **Save All**.

Refactoring Service Bus Projects, Folders, and Resources

You can rename, move, and delete Service Bus resources in JDeveloper using JDeveloper's refactoring features. You can also rename and move folders. Refactoring maintains references between the resources you change, except when you delete a resource or folder. Refactoring also moves, renames, or deletes the resource files and updates information in the Service Bus Overview Editor. In addition to JDeveloper's refactoring features, Service Bus lets you clone a project or folder to a different location.

You can rename and delete resources from the Application Navigator or from the Service Bus Overview Editor. This section describes the Application Navigator steps. For information on refactoring in the Overview Editor, see [How to Rename Components in the Service Bus Overview Editor](#) and [How to Delete Components in the Service Bus Overview Editor](#).

How to Rename a Service Bus Folder or Resource in JDeveloper

When you rename Service Bus components, any references are automatically updated.

Note:

If you rename a business service imported from the UDDI registry, the service will become detached from the registry.

To rename a folder or resource:

1. In the Application Navigator, right-click the folder or resource you want to rename.
2. Point to **Refactor** and click **Rename**.
3. For resources, do the following:
 - a. In the **Rename To** field of the Rename dialog, enter a new name for the resource. Do not change the file extension.
 - b. If the resource is referenced by another resource, click **Show Usages** to view those references.
 - c. Click **OK** to finalize the new name.
4. For folders, do the following:
 - a. In the **Name** field of the Rename Directory dialog, enter the full path and a new name for the folder.
 - b. To view a list of files before confirming the change, select **Preview**.
 - c. Click **OK**.
 - d. If you selected Preview, review the list of files in the Rename Directory Log. Click **Refactor** to complete the change.
5. In the JDeveloper toolbar, click **Save All**.

How to Move a Service Bus Folder or Resource in JDeveloper

When you move resources to a different location, Service Bus retains any dependencies against that resource. If you move a project into another project, Service Bus converts it to a folder within the second project.

To move a folder or resource:

1. In the Application Navigator, right-click the project, folder, or resource you want to move.
2. Point to **Refactor** and click **Move**.
3. For resources, do the following:
 - If the resource is referenced by another resource, click **Show Usages** to view those references.
 - In the **Move To** field of the Move dialog, enter the new directory path for the resource or click **Browse** to navigate to and select a new directory.
 - Click **OK**.
4. For folders, do the following:
 - In the Move Directory dialog, navigate to and select the new location for the folder.
 - Click **Select**.

5. In the JDeveloper toolbar, click **Save All**.

How to Delete a Project or Resource:

When you delete a project, all resources under the project are deleted. If any resources under this folder are referenced by resources under a different project or folder, you can still delete it but this might result in conflicts due to unresolved references to the deleted resource.

Caution:

If you delete a project or folder that contains a pipeline template resource, all the concrete pipelines derived from that template are unlinked.

Deleting a Resource

To delete a resource:

1. In the Application Navigator, right-click the project, folder, or resource you want to delete.
2. Point to **Refactor** and click **Delete**.
3. If the resource is referenced by another resource, click **Show Usages** to view those references.
4. Click **Yes** to delete the resource.

Deleting a Project

To delete a project:

1. In the Application Navigator, right-click the project you want to delete.
2. Click **Delete**.
3. Select whether to just remove the project from the application or to also delete all of its files and directories from the file system.
4. Click **Yes**.
5. On the Confirmation Dialog, click **Yes**.

How to Clone a Project or Folder:

Cloning a project or folder copies all resources in a project or folder to a different location. Service Bus preserves dependencies when an object is cloned and also adjusts any references. For information on clone processing, see [What Happens When You Clone a Project](#) and [What Happens When You Clone a Folder](#).

To clone a project or folder:

1. In the Application Navigator, right-click the project, folder, or resource you want to clone.
2. Point to **Service Bus** and click **Clone**.

3. On the Select Clone Target dialog, enter a name for the cloned component.
4. Do one of the following:
 - To clone this project or folder as a project, select **As project**.
 - To clone this project or folder as a folder, select **As folder in location** and then select the project or folder where you want to locate the cloned project.
5. Click **OK**.

Setting up the Development Environment for JDeveloper

This chapter describes development environment setup topics that are relevant to Service Bus, such as using the default Derby database and disabling the default JMS reporting provider.

This chapter includes the following topics:

- [Creating Server Connections in JDeveloper](#)
- [Creating Connection Factories for Oracle JCA Adapters](#)
- [Disabling the JMS Reporting Provider](#)

Creating Server Connections in JDeveloper

In order to deploy services from JDeveloper, your JDeveloper environment needs to be connected to an Oracle WebLogic Server. To do this, you create an application server connection. You can also create connections to the Oracle Metadata Services (MDS) repository to share artifacts with SOA Suite applications.

- [How to Create an Application Server Connection](#)
- [How to Create a SOA-MDS Connection](#)
- [How to Change the MDS Repository Location](#)

How to Create an Application Server Connection

When developing in JDeveloper, you must create a connection to the application server to which Service Bus applications will be deployed. Once you create this connection, you can deploy your applications to the server and you can access other applications and artifacts already deployed to that server.

For instructions, see "How to Create a Connection to the Target Application Server" in *Developing Applications with Oracle JDeveloper*.

How to Create a SOA-MDS Connection

To deploy a Service Bus application that shares data with other composites, use the Create SOA-MDS Connection wizard to create a connection to a database-based Oracle MDS Repository server.

To create a SOA-MDS connection:

1. From the **File** main menu, point to new **New** and select **From Gallery**.

The New Gallery wizard appears.

2. In the **General** category, select **Connections**.
3. Select **SOA-MDS Connection**, and click **OK**.

The Create SOA-MDS Connection dialog appears.

4. In the connection fields, provide values appropriate to your environment.

Click **Help** on the dialog to get information about each field and the values you need to enter.

5. Click **OK**.

You can now browse the connection in the Resources window and view shared artifacts under the **/apps** node.

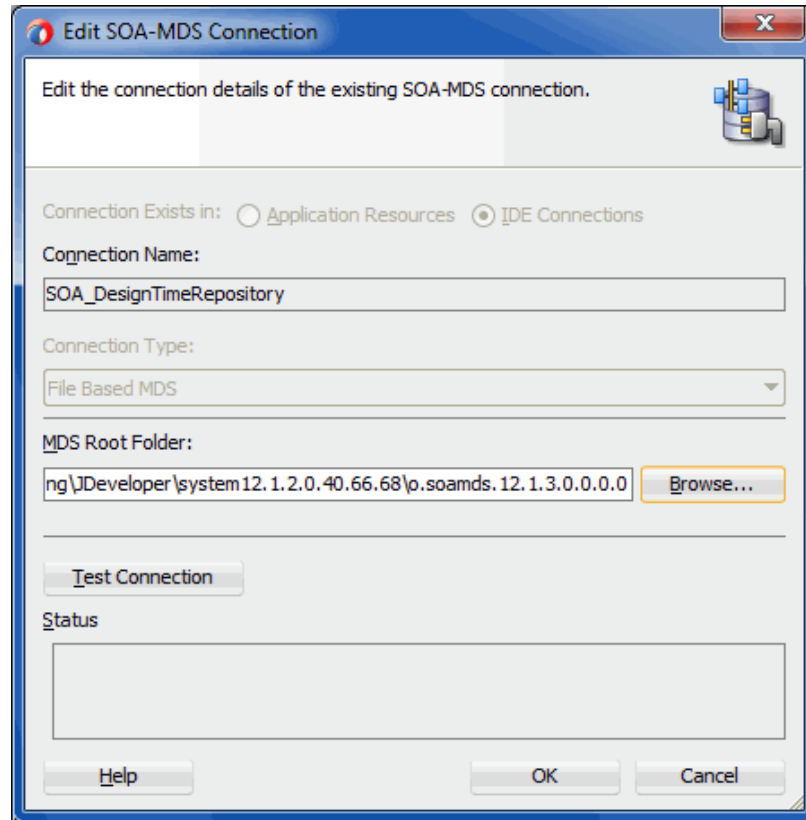
How to Change the MDS Repository Location

The default MDS Repository connection uses a default repository located in `$JDEV_USER_DIR/soamds` in the JDeveloper system or application data folders. You can change the location of the repository if needed.

To change the MDS Repository location:

1. If the Resources window is not visible in JDeveloper, click the **Window** menu and select **Resources**.
2. In the Resources window, expand **SOA-MDS**, right-click the repository name (the default name is **SOA_DesignTimeRepository**), and click **Properties**.

The Edit SOA-MDS Connection dialog appears.

Figure 4-1 Edit SOA-MDS Connection Dialog

3. In the **MDS Root Folder** field, select the new root directory for the repository.

Note:

You can select any directory, but it must have a folder named **apps** directly beneath it.

4. Click **Test Connection** to verify the directory.
5. Upon successful completion of the test, click **OK**.

Creating Connection Factories for Oracle JCA Adapters

The Oracle JCA adapters are deployed as JCA resource adapters in a WebLogic Server container. Adapters are packaged as Resource Adapter Archive (RAR) files using a JAR format. When adapters are deployed, the RAR files are used and the adapters are registered as connectors with the WebLogic Server or middle-tier platform. The RAR file contains the following:

- The `ra.xml` file, which is the deployment descriptor XML file containing deployment-specific information about the resource adapter
- Declarative information about the contract between Oracle WebLogic Server and the resource adapter

Adapters also package the `weblogic-ra.xml` template file, which defines the endpoints for connection factories. For information about creating connection factories and connection pools, see "Adapter Framework" in *Understanding Technology Adapters*.

Disabling the JMS Reporting Provider

By default, the Service Bus JMS reporting provider is deployed in an Service Bus domain. The reporting provider uses a database to persist reporting data. If you do not want to use the JMS reporting provider in your development domain, you can disable or untarget it during the domain creation process.

For more information, see "How to Untarget a JMS Reporting Provider" in *Administering Oracle Service Bus*. Disabling the reporting provider prevents benign JMS reporting provider errors at server startup.

Developing Oracle Service Bus Applications in JDeveloper

This chapter describes how to use JDeveloper to develop Service Bus applications from the top down, using the Service Bus Overview Editor. It guides you through the basic steps of creating Service Bus services using proxy services, business services, pipelines, and split-joins, along with final steps, like attaching security policies and testing or debugging components. It also provides an overview of transports, JCA Adapters, and REST bindings.

This chapter includes the following sections:

- [Introduction to the Service Bus Overview Editor](#)
- [Adding Service Bus Components](#)
- [Modifying and Deleting Components in the Service Bus Overview Editor](#)
- [Synchronizing the Overview Diagram](#)
- [Wiring Service Bus Components](#)
- [Attaching Security Policies to Service Bus Components](#)
- [Testing Service Bus Components in the Overview Editor](#)
- [Deploying a Service Bus Application](#)

Introduction to the Service Bus Overview Editor

The Service Bus Overview Editor provides a graphical interface for you to design and configure Service Bus projects. A project overview file is automatically generated when you create a project, which describes the Service Bus project. In the Application Navigator, the node representing this file has the same name as the project it represents; in the file system, it is named `servicebus.sboverview`. You can open this file in the Service Bus Overview Editor to create and configure the Service Bus components.

Opening the overview file launches the Service Bus Overview Editor, which appears as a tab in the JDeveloper designer. This file describes the entire application assembly of proxy services, business services, pipelines, and split-joins. There is one overview file for each Service Bus project.

Service Bus Overview Editor Components

When you work with the project overview file, you primarily use the Overview Editor canvas, the Components window, and the component configuration wizards. The editor lets you view many of your files in a WYSIWYG environment, and you can view a file in an overview editor where you can declaratively make changes, or you

can view the source code for the file. The Structure window shows the structure of the currently selected file.

Figure 5-1 Service Bus Overview Editor

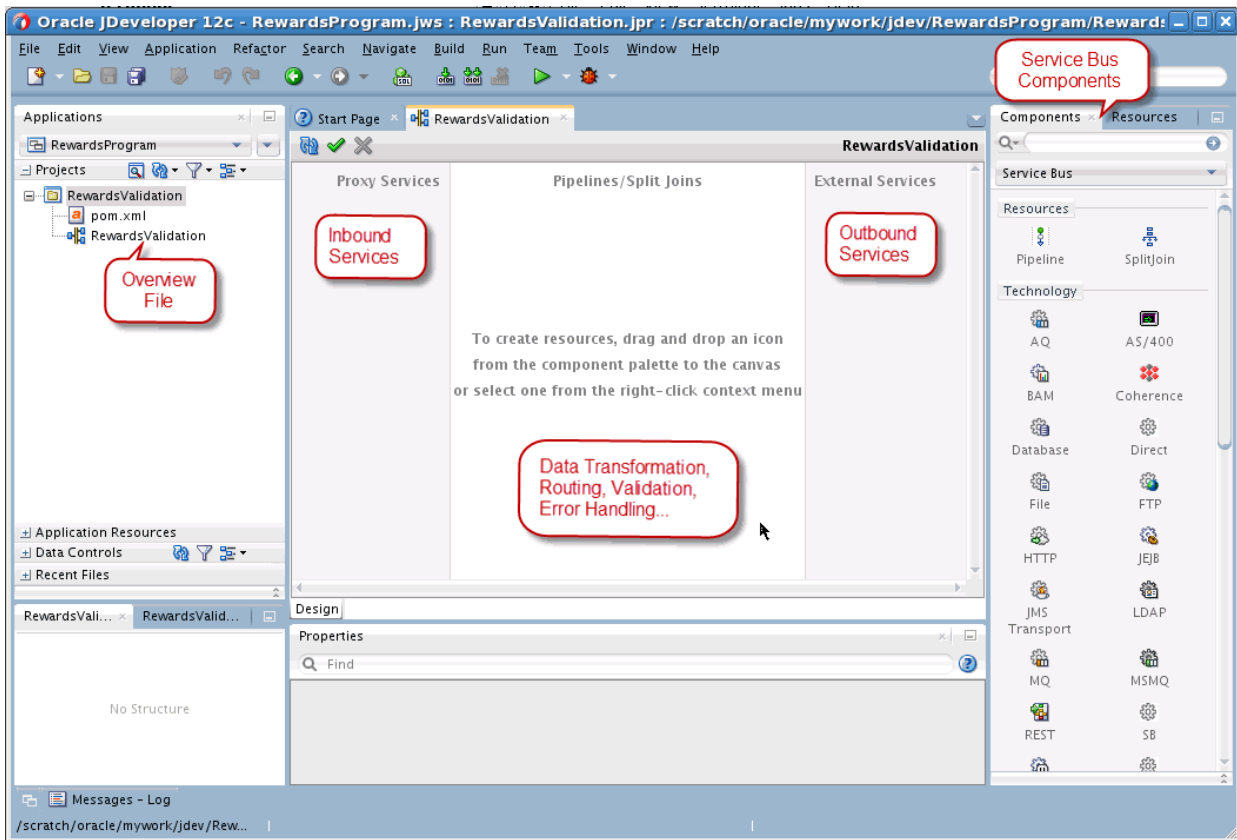


Table 5-1 describes the Service Bus Overview Editor.

Table 5-1 Service Bus Overview Editor

Element	Description
Application Navigator	<p>Displays the key files for the components included in the Service Bus project:</p> <ul style="list-style-type: none"> • A <i>project_name</i> node (servicebus.sboverview file) that is automatically created when you create a Service Bus project. This file describes the entire composite assembly of proxy services, business services, pipelines, split-joins, and wires. Once you open the file in the Service Bus Overview Editor, the name changes to match that of the containing project. • The pipeline component file (<i>pipeline_name.pipeline</i>) • The proxy service component file (<i>proxy_name.proxy</i>) • The business service component file (<i>business_name.bix</i>) • The split-join component file (<i>splitjoin_name.flow</i>) • The individual files that describe Service Bus components, such as alert destinations, service key providers, and service accounts. • Optional subfolders for class files, WSDL files, schema definitions, transformations, and test suites. These can also be created directly under the project. By default, components generated by the adapter wizard are created in a subfolder named <code>Resources</code>.
Structure Window	The Structure window provides a structural view of the data in the document currently selected in the active window.
Overview Editor	<p>You drag Service Bus components from the Components window into the canvas of the Overview Editor. When you drag and drop a component into the designer, a corresponding wizard appears so you can create and configure that component.</p> <p>For all subsequent editing sessions, double-click these components to re-open their editors.</p>
Middle Swimlane (Pipelines/Split Joins)	The middle swimlane is for the components that provide routing and transformation logic, restricting this lane to pipelines and split-joins.
Left Swimlane (Proxy Services)	The left swimlane is for services that provide an entry point to the application, so this lane is restricted to proxy services. If you drag a JCA adapter or a transport to this lane, you generate a proxy service based on the selected adapter or transport.
Right Swimlane (External Services)	<p>The right swimlane is for references that send messages to external services in the outside world, which can be business services or proxy services. If you drag a JCA adapter or a transport to this lane, you generate a business service.</p> <p>Proxy services that appear in this lane are actually references to existing proxy services, so they cannot be added by dragging a JCA adapter or a transport to the canvas. They can only be added by selecting the existing proxy service to use.</p>

Table 5-1 (Cont.) Service Bus Overview Editor

Element	Description
Components Window	<p>The Components window provides the various Service Bus components that you can use in a Service Bus application. It contains pipelines, split-joins, technology adapters, application adapters, transports, and a REST binding. Proxy services and business services are not included in the Components window, but you can create them by dragging adapters, transports, or bindings to the left or right swimlanes of the canvas.</p> <p>For more information about the available components, see the online help for the Overview Editor in JDeveloper. If the Components window is not visible, select Components from the View main menu.</p>
Resources Window	<p>The Resources window provides a single dialog from which you can browse both local and remote resources. For example, you can access the following resources:</p> <ul style="list-style-type: none"> • Shared local application metadata such as schemas, WSDL documents, event definitions, and so on. • WSIL browser functionality that uses remote resources that can be accessed through an HTTP connection, file URL, or application server connection. • Remote resources that are registered in a Universal Description, Discovery, and Integration (UDDI) registry. <p>You select these resources for the Service Bus application through the Resource Browser dialog. This dialog is accessible through a variety of methods. For example, when you select the WSDL file to use with a pipeline or drag a business service from the Components window, the Resource Browser dialog appears. Click Resources at the top of this dialog to access available resources.</p> <p>If the Resources window is not visible, then select Resources from the View main menu.</p>
Log Window	<p>The Log window displays messages about application compilation, validation, and deployment.</p>
Properties Window	<p>The Properties window displays properties for the selected Service Bus component. You also use the Properties window to define properties for pipeline and split-join actions.</p> <p>If the Properties window is not visible, select Properties from the View main menu.</p>
Application Resources	<p>In the Application Resources panel, the system resources for Service Bus applications and projects are listed under Service Bus System Resources. System resources include UDDI registries, proxy services, JNDI providers, and SNMP servers.</p>

Transports, Adapters, and Bindings

Using the Service Bus Overview Editor, you can create proxy and business services based on Service Bus transports, Oracle JCA Adapters, or the REST binding. Service Bus is fully integrated with JCA adapters, and you can create and configure a JCA adapter and its associated files directly in a Service Bus project using the Overview Editor. In previous versions, you created the JCA adapter in a separate SOA Suite project and then imported it into Service Bus. When you create a JCA adapter, the

concrete WSDL file and proxy or business service are created automatically. The same applies when you create a service using the REST binding; the associated WADL file and proxy or business service are automatically created for you.

In addition to the full range of support for Oracle JCA Adapters, Service Bus provides connectivity to external systems through a variety of transports, each of which is specific to a type of external system. You can use a combination of transports, adapters, and REST bindings to create services in Service Bus.

[Table 5-2](#), [Table 5-3](#), and [Table 5-4](#) describe the transport, adapters, and bindings available in the Components window. For more information, see [Working with JCA Adapters, Transports, and Bindings](#).

Table 5-2 Technology Components

Component	Description
AQ	<p>Drag an AQ adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle AQ Adapter, and to create its associated service. The AQ Adapter lets Service Bus interact with a single consumer or a multi-consumer queue.</p> <p>This action launches the AQ Adapter Configuration wizard.</p>
AS/400	<p>Drag an AS/400 adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle Database Adapter for AS/400, and to create its associated service.</p> <p>This action launches the Database Adapter Configuration wizard.</p>
BAM	<p>Drag a BAM adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle BAM Adapter, and to create its associated service. The BAM Adapter lets Service Bus interact with Oracle Business Activity Monitoring.</p> <p>This action launches the BAM Adapter Configuration wizard.</p>
Coherence	<p>Drag a Coherence adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle Coherence Resource Adapter, and to create its associated service. Use the Coherence Resource Adapter to perform cache operations in a transaction.</p> <p>This action launches the Coherence Adapter Configuration wizard.</p>
Database	<p>Drag a Database adapter to either the Proxy Services or External Services swimlane to create and configure a Database JCA Adapter, and to create its associated service. The Database Adapter lets Service Bus communicate with Oracle and other databases through JDBC.</p> <p>This action launches the Database Adapter Configuration wizard.</p>
Direct	<p>Drag a Direct transport to the External Services swimlane to create a business service using the SOA-DIRECT transport. Use the SOA DIRECT transport to exchange messages over a remote method invocation (RMI).</p> <p>This action launches the Create Business Service wizard. The Direct transport can only be used with business services.</p>

Table 5-2 (Cont.) Technology Components

Component	Description
File	<p>Drag a File adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle File Adapter, and to create its associated service. The File Adapter lets Service Bus applications read and write messages from files on a local file system.</p> <p>This action launches the File Adapter Configuration wizard.</p>
FTP	<p>Drag an FTP adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle FTP Adapter, and to create its associated service. The FTP Adapter lets Service Bus applications read and write messages from remote service files.</p> <p>This action launches the FTP Adapter Configuration wizard.</p>
HTTP	<p>Drag an HTTP transport to the Proxy Services swimlane to create an HTTP proxy service, or to the External Services swimlane to create an HTTP business service. The HTTP transport lets you invoke applications through HTTP POST and GET operations.</p> <p>This action launches either the Create Proxy Service wizard or the Create Business Service wizard.</p>
JEJB	<p>Drag a JEJB transport to the Proxy Services swimlane to create a JEJB proxy service, or to the External Services swimlane to create a JEJB business service. The JEJB transport lets you pass Plain Old Java Objects (POJOs) through Service Bus.</p> <p>This action launches either the Create Proxy Service wizard or the Create Business Service wizard.</p>
JMS	<p>Drag a JMS Transport to the Proxy Services swimlane to create a JMS proxy service, or to the External Services swimlane to create a JMS business service. The JMS transport configures services that interact with Java Messaging Service.</p> <p>This action launches either the Create Proxy Service wizard or the Create Business Service wizard.</p>
LDAP	<p>Drag an LDAP adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle LDAP Adapter, and to create its associated service. The LDAP Adapter lets Service Bus interact with an LDAP directory.</p> <p>This action launches the LDAP Adapter Configuration wizard.</p>
MQ	<p>Drag an MQ adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle MQ Series Adapter, and to create its associated service. The MQ Series Adapter lets Service Bus connect to MQ Series queue managers and to add and remove messages in queues.</p> <p>This action launches the MQ Series Adapter Configuration wizard.</p>

Table 5-2 (Cont.) Technology Components

Component	Description
MSMQ	<p>Drag an MSMQ adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle MSMQ Adapter, and to create its associated service.</p> <p>This action launches the MSMQ Adapter Configuration wizard.</p>
REST	<p>Drag REST binding to either the Proxy Services or External Services swimlane to create and configure a REST binding component, and to create its associated service.</p> <p>This action launches the Create REST Binding dialog.</p>
SB	<p>Drag an SB transport to the Proxy Services swimlane to create an SB proxy service, or to the External Services swimlane to create an SB business service. Use the SB transport to allow Oracle products to synchronously invoke a proxy service using RMI.</p> <p>This action launches either the Create Proxy Service wizard or the Create Business Service wizard.</p>
Socket	<p>Drag a Socket adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle Socket Adapter, and to create its associated service. The Oracle Socket Adapter lets Service Bus create a client or server socket, and establish a connection.</p> <p>This action launches the SOCKET Adapter Configuration wizard.</p>
Tuxedo	<p>Drag a Tuxedo transport to the Proxy Services swimlane to create a Tuxedo proxy service, or to the External Services swimlane to create a Tuxedo business service. The Tuxedo transport to access a Tuxedo domain from Service Bus.</p> <p>This action launches either the Create Proxy Service wizard or the Create Business Service wizard.</p>
UMS	<p>Drag a UMS adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle UMS Adapter, and to create its associated service. The Oracle UMS Adapter lets Service Bus send and receive notifications using email, SMS, or instant messaging.</p> <p>This action launches the UMS Adapter Configuration wizard.</p>
WS	<p>Drag a WS transport to the Proxy Services swimlane to create a WS proxy service, or to the External Services swimlane to create a WS business service. The WS transport implements requests for services derived from based on SOAP 1.1- or SOAP 1.2-based WSDL documents with WSRM policy.</p> <p>This action launches either the Create Proxy Service wizard or the Create Business Service wizard.</p>

Table 5-3 Application Components

Component	Description
JDE World	Drag a JDE World adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle JDE World Adapter, and to create its associated service. This action launches the JDE World Adapter Configuration wizard.
Oracle	Drag an Oracle adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle Applications Adapter, and to create its associated service. This action launches the Oracle Applications Adapter Configuration wizard.
SAP	Drag a SAP adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle SAP Adapter, and to create its associated service. This action launches the Oracle SAP Adapter Configuration wizard.

Table 5-4 Advanced Components

Component	Description
BPEL 10g	<p>Drag a BPEL 10g transport to the External Services swimlane to create a BPEL Process Manager business service. Use the BPEL Processing Manager (BPEL-10g) to define messaging with Oracle SOA Suite 10g BPEL processes.</p> <p>This action launches the Create Business Service wizard. The BPEL 10g transport can only be used with business services.</p> <p>Note: For messaging with Oracle SOA Suite 11g BPEL processes, use the SOA_DIRECT transport.</p>
Custom	<p>Drag a Custom adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle Custom Adapter, and to create its associated service. The Oracle Custom Adapter lets you create a customized adapter to connect to external systems.</p> <p>This action launches the Custom Adapter Configuration wizard.</p>
DSP	<p>Drag a DSP transport to the External Services swimlane to create a DSP business service. Use the DSP transport to communicate with Oracle Data Service Integrator.</p> <p>This action launches the Create Business Service wizard. The DSP transport can only be used with business services.</p>
EJB	<p>Drag an EJB transport to the External Services swimlane to create an EJB business service. Use the EJB transport to create an Enterprise JavaBeans service for using SDO parameters or Java interfaces with Enterprise JavaBeans.</p> <p>This action launches the Create Business Service wizard. The EJB transport can only be used with business services.</p>

Table 5-4 (Cont.) Advanced Components

Component	Description
Email	<p>Drag an Email transport to the Proxy Services swimlane to create an email proxy service, or to the External Services swimlane to create an email business service. The Email transport lets Service Bus communicate with email servers.</p> <p>This action launches either the Create Proxy Service wizard or the Create Business Service wizard.</p>
File Transport	<p>Drag a File Transport to the Proxy Services swimlane to create a file proxy service, or to the External Services swimlane to create a file business service. The File transport configures services that read and write messages from files on a local file system.</p> <p>This action launches either the Create Proxy Service wizard or the Create Business Service wizard.</p>
FTP Transport	<p>Drag an FTP Transport to the Proxy Services swimlane to create an FTP proxy service, or to the External Services swimlane to create an FTP business service. The FTP transport configures services that read and write messages from remote service files.</p> <p>This action launches either the Create Proxy Service wizard or the Create Business Service wizard.</p>
JCA	<p>Drag a JCA transport to the Proxy Services swimlane to create a proxy service based on an existing JCA adapter, or to the External Services swimlane to create a business service based on an existing JCA adapter.</p> <p>This action launches either the Create Proxy Service wizard or the Create Business Service wizard.</p>
JMS	<p>Drag a JMS adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle JMS Adapter, and to create its associated service. The JMS Adapter lets Service Bus interact with Java Messaging Service.</p> <p>This action launches the JMS Adapter Configuration wizard.</p>
Local	<p>Drag a Local transport to the Proxy Services swimlane to create a Local proxy service. Local proxy services can only be invoked by other proxy services, not by external clients.</p> <p>This action launches the Create Proxy Service wizard. The Local transport can only be used with proxy services.</p>
MQ Transport	<p>Drag an MQ Transport to the Proxy Services swimlane to create an MQ proxy service, or to the External Services swimlane to create an MQ business service. The MQ transport provides access to IBM WebSphere MQ.</p> <p>This action launches either the Create Proxy Service wizard or the Create Business Service wizard.</p>

Table 5-4 (Cont.) Advanced Components

Component	Description
SFTP	<p>Drag an SFTP Transport to the Proxy Services swimlane to create an SFTP proxy service, or to the External Services swimlane to create an SFTP business service. The SFTP transport lets you transfer files securely over the SSH File Transfer Protocol (SFTP).</p> <p>This action launches either the Create Proxy Service wizard or the Create Business Service wizard.</p>
Third Party	<p>Drag a Third Party adapter to either the Proxy Services or External Services swimlane to create and configure an Oracle Third Party Adapter, and to create its associated service. The Third Party Adapter lets Service Bus interact with third-party services as long as a WSDL document is already defined.</p> <p>This action launches the Create Third Party Adapter Service dialog.</p>

Project and Overview Diagram Synchronization

Performing certain tasks outside of the Overview Editor results in components being automatically added to or changed in the Service Bus Overview Editor diagram. When you create proxy services, pipelines, and split-joins outside of the Service Bus Overview Editor (that is, you create them directly in the project using their creation wizards), Service Bus adds the generated components to the Overview Editor diagram. This does not occur when you add business services, since they can be used by multiple projects.

When you add a new outbound messaging or callout activity to a pipeline or split-join and configure the activity to call an external service, Service Bus adds the referenced component to the External Services swimlane in the overview diagram if it is not already there. Service Bus also wires the pipeline or split-join to the referenced component. If you remove the call to the external service, the wire is also removed. When you add, remove, or modify the target service in the Proxy Service Definition Editor, Service Bus modifies the wires on the overview diagram accordingly.

If you import a new Service Bus project, Service Bus generates the project overview file and adds all derivable Service Bus components to the diagram in the Service Bus Overview Editor. If your import updates an existing Service Bus project, the overview diagram is also updated with any new components or updated wiring.

The overview is synced when you make changes to or import resources. After you add, update, or delete a resource, the changes will appear in the Service Bus Overview Editor the next time you refresh or open it.

Adding Service Bus Components

Once you create your Service Bus applications and projects, the next step is to add Service Bus components that implement the business logic or processing rules of your application. You can use the Components window from the Service Bus Overview Editor to drag and drop components into the overview editor. Dragging Service Bus components to the editor launches a corresponding creation wizard where you can create the component and any associated resources, such as WSDL files and schema definitions. You can also create components by right-clicking in one of the swimlanes,

and then selecting from the available options for that lane. Once you create the component, the corresponding files are generated in the project directory structure and appear in the Application Navigator.

How to Launch the Service Bus Overview Editor

The Overview Editor displays a graphical representation of the `servicebus.sboverview` file. In the Application Navigator, this file is represented by the node with the same name as the project.

Note:

When you open the Overview Editor, Service Bus always tries to synchronize the diagram to the current state of the project components.

To open the project overview:

- Do one of the following:
 - Double-click the node in the Service Bus project with the same name as the project.
 - Right-click the node in the Service Bus project with the same name as the project and select **Open**.

How to Add a Pipeline

Add a pipeline to define the message flow and any data transformation, error handling, and validation for the project. When you create a new pipeline, you have the option to generate a proxy service from the pipeline configuration.

To add a pipeline:

1. From the Components window, select **Service Bus**.
2. From the **Resources** list, drag a Pipeline into the Pipelines/Split Joins lane in the designer.

The Create Pipeline Service wizard appears.

3. Configure the settings for the pipeline.

For help with the configuration fields, click **Help** or press **F1**. For more information about creating pipelines, see [Working with Pipelines in Oracle JDeveloper](#).

4. To generate a proxy service to associate with the pipeline, click **Expose as a Proxy Service** on the Type page of the wizard. Select a transport for the proxy service and, optionally, modify the name.
5. On the last page of the wizard, click **Finish**.

The pipeline file is added to the project, and the pipeline appears in the Pipelines/Split Joins section of the designer. If you exposed the pipeline as a proxy service, the proxy service also appears in the Proxy Services swim lane, and the components are automatically wired.

6. To define the message flow in the pipeline, see [Working with Pipeline Actions in Oracle JDeveloper](#).

7. Click **Save All** in the JDeveloper toolbar.

How to Add a Split-Join

Add a split-join to define a message flow that performs concurrent processing to improve service performance. You can also define data transformations, validations, error handling, and reporting in a split-join. When you create a new split-join, you have the option to generate a proxy service from the split-join configuration.

To add a split-join:

1. From the Components window, select **Service Bus**.
2. From the **Resources** list, drag a SplitJoin into the Pipelines/Split Joins lane in the designer.

The Create Split-Join Service wizard appears.

3. Configure the settings for the split-join.

For help with the configuration fields, click **Help** or press **F1**. For more information about creating split-joins, see [How to Create a Split-Join in JDeveloper](#).

4. To generate a proxy service to associate with the split-join, click **Expose as a Proxy Service** on the Type page of the wizard. Select a transport for the proxy service and, optionally, modify the name.
5. On the last page of the wizard, click **Finish**.

The split-join file is added to the project, and the split-join appears in the Pipelines/Split Joins section of the designer. If you exposed the split-join as a proxy service, the proxy service also appears in the Proxy Services swim lane, and the components are automatically wired.

6. To define the message flow in the split-join, see [Improving Service Performance with Split-Join](#).
7. Click **Save All** in the JDeveloper toolbar.

How to Create a Proxy Service

A proxy service is the entry point of data into the Service Bus application. There are several ways to create a proxy service using the Service Bus Overview Editor.

- Drag an adapter or transport from the Components window to the Proxy Services lane. Adapters and transports are under **Technology**, **Applications**, and **Advanced** in the Components window.
- When you create a pipeline or split-join, expose it as a proxy service. This creates the pipeline or split-join, the proxy service, and the connecting wire between the two. For more information, see [How to Add a Pipeline](#) and [How to Add a Split-Join](#).
- After you create a pipeline or split-join, drag the input anchor to the Proxy Services swimlane to create a new proxy service.

Creating a Proxy Service with an Adapter

When you use a JCA adapter to create a proxy service, Service Bus generates a concrete WSDL file along with a JCA-based proxy service. The WSDL file generated

for the JCA adapter is abstract. The concrete WSDL file has the text "concrete" appended to the file name.

To create a proxy service with an adapter:

1. Do one of the following:

- Right-click in the Proxy Services swim lane, point to **Insert Adapters**, and select the adapter to use from the list of options.
- From the Components window, select **Service Bus** and drag an adapter into the Proxy Services lane.

Tip:

Adapters are designated by a unique icon in the Technology section of the Components window.

The creation wizard for the selected adapter appears.

2. Configure the settings for the adapter.

For help with the configuration fields, click **Help** or press **F1**. For more information about configuring adapters, see *Generic Oracle JCA Adapter Properties* in *Understanding Technology Adapters*.

3. On the last page of the wizard, click **Finish**.

The proxy service, adapter, and associated WSDL files are added to the project, and the proxy service appears in the Proxy Services section of the designer. The proxy service is configured for a JCA adapter, and it is named based on the type of JCA adapter used.

4. To configure the proxy service, see [Configuring Proxy Services](#) and [Using the JCA Transport and JCA Adapters](#).

5. Click **Save All** in the JDeveloper toolbar.

Creating a Proxy Service with a Transport

To create a proxy service with a transport:

1. Do one of the following:

- Right-click in the Proxy Services swim lane, point to **Insert Transports**, and select the transport to use from the list of options.
- From the Components window, select **Service Bus** and drag a transport into the Proxy Services lane.

Tip:

Transports are designated by a plain gear icon in the Technology and Advanced sections of the Components window.

The Create Proxy Service Wizard appears.

2. Configure the settings for the proxy service.

For help with the configuration fields, click **Help** or press **F1**. For more information about creating proxy services, see [How to Create a Proxy Service](#).

3. On the last page of the wizard, click **Finish**.

The proxy service file is added to the project, and the proxy service appears in the Proxy Services section of the designer.

4. To configure the proxy service and transport, see [Configuring Proxy Services and Working with JCA Adapters, Transports, and Bindings](#)
5. Click **Save All** in the JDeveloper toolbar.

Creating a Proxy Service from an Existing Pipeline or Split-Join

To create a proxy service from an existing pipeline or split-join:

1. Click the left anchor of the pipeline or split-join and drag it to the Proxy Services swimlane.

The Create Proxy Service wizard appears.

2. Configure the settings for the proxy service.

For help with the configuration fields, click **Help** or press **F1**. For more information about creating proxy services, see [How to Create a Proxy Service](#).

3. On the last page of the wizard, click **Finish**.

The proxy service file is added to the project, and the proxy service appears in the Proxy Services section of the designer.

4. To configure the proxy service and transport, see [Configuring Proxy Services and Working with JCA Adapters, Transports, and Bindings](#)
5. Click **Save All** in the JDeveloper toolbar.

How to Reuse Existing Proxy Services in the Overview

A proxy service can be called as an external service from a pipeline or split-join in the Service Bus application. You can add a proxy service to reference from the External Services context menu or from an existing pipeline or split-join. The proxy service to reference must already be created in the current application. Typically, proxy services are referenced so their logic is not exposed, and they usually use the local transport.

To add an existing proxy service as a reference:

1. Do one of the following:
 - Right-click in the External Services swim lane, point to **Choose**, and select **Proxy Service**.

The Select Proxy Service dialog appears.

- Click the right anchor of a pipeline or split-join, and drag it to the External Services swimlane.

The Resource Chooser dialog appears.

2. In the list of projects, expand the project and any folders containing the proxy service you want to use.
3. Select the proxy service to use, and then click **Finish**.

4. Click **Save All** in the JDeveloper toolbar.

How to Create a Business Service

A business service communicates with the external systems with which you share data. You can create a business service by dragging either an adapter or a transport from the Components window to the External Services lane. Adapters and transports are under **Technology**, **Applications**, and **Advanced** in the Components window.

Creating a Business Service with an Adapter

When you use a JCA adapter to create a business service, Service Bus generates a concrete WSDL file along with the business service. The WSDL file generated for the JCA adapter is abstract. The concrete WSDL file has the text "concrete" appended to the file name.

To create a business service with an adapter:

1. Do one of the following:
 - Right-click in the External Services swim lane, point to **Insert Adapters**, and select the adapter to use from the list of options.
 - From the Components window, select **Service Bus** and drag an adapter into the External Services lane in the designer.

Tip:

Adapters are designated by a unique icon in the Technology section of the Components window.

The creation wizard for the selected adapter appears.

2. Configure the settings for the adapter.

For help with the configuration fields, click **Help** or press **F1**. For more information about configuring adapters, see *Generic Oracle JCA Adapter Properties* in *Understanding Technology Adapters*.

3. On the last page of the wizard, click **Finish**.

The business service, adapter, and associated WSDL files are added to the project, and the business service appears in the External Services section of the designer. The business service is configured for a JCA adapter, and is named based on the type of JCA adapter used.

4. To configure the business service, see [Configuring Business Services](#) and [Using the JCA Transport and JCA Adapters](#).
5. Click **Save All** in the JDeveloper toolbar.

Creating a Business Service with a Transport

To create a business service with a transport:

1. Do one of the following:
 - Right-click in the External Services swim lane, point to **Insert Transports**, and select the transport to use from the list of options.

- From Components window, select **Service Bus** and drag a transport into the External Services lane in the designer.

Tip:

Transports are designated by a plain gear icon in the Technology and Advanced sections of the Components window.

The Create Business Service Wizard appears.

2. Configure the settings for the business service.

For help with the configuration fields, click **Help** or press **F1**. For more information about creating business services, see [How to Create a Business Service](#).

3. On the last page of the wizard, click **Finish**.

The business service file is added to the project, and the business service appears in the External Services section of the designer.

4. To configure the business service and transport, see [Configuring Business Services](#) and [Working with JCA Adapters, Transports, and Bindings](#)
5. Click **Save All** in the JDeveloper toolbar.

How to Reuse Existing Business Services in the Overview

If the business service you want to use already exists in the project, you can reference that business service from the Overview Editor. You can add the business service from the External Services context menu or from an existing pipeline or split-join. The business service must already be created in the current application.

To add an existing business service to the overview:

1. Do one of the following:
 - Right-click in the External Services swim lane, point to **Choose**, and select **Business Service**.
The Select Business Service dialog appears.
 - Click the right anchor of a pipeline or split-join, and drag it to the External Services swimlane.
The Resource Chooser dialog appears.
2. In the list of projects, expand the project and any folders containing the business service you want to use.
3. Select the business service to use, and then click **Finish**.
4. Click **Save All** in the JDeveloper toolbar.

How to Invoke Deployed Service Bus and SOA Applications

From your Service Bus applications, you can invoke other Service Bus or SOA applications that have already been deployed to the Oracle WebLogic Server.

To invoke deployed Service Bus and SOA applications:

1. Launch the Create Business Service or Create Proxy Service wizard by dragging a transport or adapter to the Proxy Services or External Services swim lane on the Service Bus Overview Editor.
2. On the Type page, select **WSDL** and click the **Find existing WSDLs** icon.
The Select WSDL dialog appears.
3. In the list at the top, select **Application Server**.
4. Select the Oracle WebLogic Server on which the application is deployed.
5. Expand the tree to display the Service Bus or SOA application you want to invoke.
6. Continue expanding the application until the service to invoke is visible.
7. Select the service and click **OK**.
The Import Service Bus Resources dialog appears.
8. Verify the resource and location information, and click **Next**.
9. On the Configuration page, make sure the service and any dependent resource (such as schemas) are selected, and click **Finish**.
10. On the Type page of the Create Business/Proxy Service wizard, verify the binding or port information, and click **Next**.
11. Verify the transport and URI information, and click **Finish**.

For information about creating an application server connection, see [Deploying Oracle Service Bus Services](#).

What You May Need to Know About Adding Components

Note the following about adding components:

- You can create a component from either the Service Bus Overview Editor or the Application Navigator menu, which is accessed by right-clicking a project or folder. Both ways add the component to the overview and add the component's file to the project.

The following chapters provide instructions for adding components from the Application Navigator:

- [Creating and Configuring Proxy Services](#)
- [Creating and Configuring Business Services](#)
- [Improving Service Performance with Split-Join](#)
- [Working with Oracle Service Bus Pipelines](#)
- You can also create Service Bus components from web services deployed to the runtime and those shared using the Metadata Services (MDS) repository. Use the Resources window to browse for the web services. For more information, see [Sharing Data Using the Metadata Services Repository](#).

Modifying and Deleting Components in the Service Bus Overview Editor

From the Service Bus Overview Editor, you can rename a component, delete a component, and access a component's definition editor to update its configuration.

- [How to Edit Components from the Service Bus Overview Editor](#)
- [How to Rename Components in the Service Bus Overview Editor](#)
- [How to Delete Components in the Service Bus Overview Editor](#)

How to Edit Components from the Service Bus Overview Editor

Once you create components in the Overview Editor, additional configuration is required. You can access a component's definition editor from the Overview Editor in order to configure specific details about that component.

To edit a Service Bus component:

1. Double-click the component in the Service Bus Overview Editor to display the appropriate editor or designer, as described in [Table 5-5](#).

Table 5-5 Starting Service Bus Editors from the Service Bus Overview Editor

Component	Description
Pipeline	Double-click a pipeline in the Service Bus Overview Editor to launch the Pipeline Definition Editor, where you can define data flow and transformations. This editor provides a graphical format where you can drag and drop stages, actions, and error handlers into the message flow.
Split-Join	Double-click a split-join in the Service Bus Overview Editor to launch the Split-Join Definition Editor, where you can define the data flow and transformations for processing message parts in parallel. This editor provides a graphical format where you can drag and drop operations and error handlers into the message flow.
Proxy Service	Double-click a proxy service in the Service Bus Overview Editor to launch the Proxy Service Definition Editor, where you can further configure the transport, connectivity information, and security policies for the proxy service.
Business Service	Double-click a business service in the Service Bus Overview Editor to launch the Business Service Definition Editor, where you can further configure the transport, connectivity information, and security policies for the business service.

2. To edit the JCA Adapter on which a proxy or business service is based, right-click the proxy or business service in the Overview Editor and select **Edit JCA**.

The configuration wizard for the associated adapter appears.

3. To edit the REST Binding on which a proxy or business service is based, right-click the proxy or business service in the Overview Editor and select **Edit REST**.

The REST Binding wizard appears.

4. To return to the Service Bus Overview Editor from the editor for any Service Bus component, double-click the *project_name* node in the Application Navigator or single-click the *project_name* tab above the designer.

For help with a service component editor, click **Help** or press **F1**.

5. From the **File** main menu, select **Save All**.

How to Rename Components in the Service Bus Overview Editor

When you rename a component from the Service Bus Overview Editor, the names of the file and the corresponding node in the Application Navigator are updated to reflect the change. Any references to the renamed component are also updated.

Note:

When you change a component's name using JDeveloper's refactoring feature, the same processing occurs.

To rename a Service Bus component:

1. Right-click the component you want to rename in the Service Bus Overview Editor.
2. Click **Rename**.
3. On the Rename dialog, enter a new name for the component in the **Rename To** field. Do not change the file extension.
4. If the component is referenced by another component, click **Show Usages** to view those references.
5. Click **OK** to finalize the new name.
6. In the JDeveloper toolbar, click **Save All**.

How to Delete Components in the Service Bus Overview Editor

When a component is deleted, all references pointing to it are invalidated and all wires are removed. This also deletes the file that defines the component. Associated files, like WSDL documents or XML schemas, are not deleted.

To delete a Service Bus component:

1. In the Service Bus Overview Editor, right-click the component to delete and select **Delete**.

The Confirm Delete dialog appears.

2. If the component is used in any projects, click **Show Usages** to view more information.
3. To proceed with deleting the component, click **Yes**.

Synchronizing the Overview Diagram

You can add and edit Service Bus components from both inside and outside the Service Bus Overview Editor. Changes made from outside the Overview Editor are automatically synchronized with the overview diagram when you launch the

Overview Editor, but the Overview Editor toolbar gives you options to refresh and update the diagram if you suspect the diagram is not displaying properly.

- **Refresh Diagram:** Synchronizes the composite diagram with the current state of the project components as stored in the configuration framework.
- **Refresh Validation State:** Runs a validation check and highlights all the conflict indicators on the composite nodes. Conflicts are indicated by a red icon with an "x" in the middle.

When you close the Service Bus Overview Editor while there are unsaved changes to project components, a dialog appears with a list of files that need to be saved. You can select which files to save before closing the editor. Any files you do not save revert to their last saved state when the editor closes. It is always good practice to save your work frequently to avoid loss of data and to keep project files synchronized.

Wiring Service Bus Components

You wire (connect) proxy services, pipelines, split-joins, and external services to indicate the order in which components are called during message processing. The following guidelines apply when wiring Service Bus components.

- Since a proxy service is an inbound service, the reference handle appears on the right side. External services are outbound, so the service handle appears on the left side. Pipelines and split-joins have handles on both sides.
- You can wire a proxy service to any other component or reference type. When you wire from a proxy service to another component, Service Bus updates the Target Service field for the proxy service with the new target information.
- You can wire pipelines and split-joins to any other component or reference type. When you wire from a pipeline or a split-join to another component, Service Bus adds the component information to the External Services node of the pipeline or split-join.
- A proxy service can only post to one component or reference, so each proxy service can only have one outbound wire.
- If the source and destination services you are wiring have incompatible binding types, you cannot wire the components. This validation is based on the currently saved state.

How to Wire Service Bus Components

You can wire a proxy service to a pipeline, split-join, or external service. You can wire pipelines and split-joins to each other and to an external service. For example, you can wire a proxy service to a pipeline, wire that pipeline to a split-join, and then wire the split-join to a business service.

To wire Service Bus components:

1. Click the handle on the component to wire from, and then drag a wire to the component to wire to.

Tip:

When you drag the wire, make sure to drag into the receiving handle on the component. Otherwise, the wire might not be created.

2. Select **Save All** from the **File** main menu.
3. To verify the target service for a proxy service, do the following:
 - a. Double-click the proxy service.
 - b. Verify that the wired component appears in the **Target Service** field on the General tab.
4. To verify the external services for a pipeline, do the following:
 - a. Double-click the pipeline.
 - b. In the Pipeline Definition Editor, click the left-arrow button next to the root node.
 - c. Expand **External Services**.
The wired proxy services appear in the list.
5. To verify the external services for a split-join, do the following:
 - a. Double-click the split-join.
 - b. In the Split-Join Definition Editor, click the left-arrow button next to the root node.
 - c. Expand **External Services**.
The wired business services appear in the list.

How to Delete Wires Between Services

When you delete wires, any linking information is cleaned up and removed as well.

- When you remove a wire from a proxy service to a pipeline, split-join, or business service, Service Bus removes the service information from the Target Service field for the proxy service.
- When you remove a wire from a pipeline or a split-join to another component, Service Bus removes the component information from the External Services node of the pipeline or split-join.
- When an activity in a component is linked with a deleted link, validating that component reports the error so you can explicitly correct it. Service Bus does not perform automatic corrections for this when you delete a wire.

To delete wires between services:

1. Right-click the wire to remove and click **Delete**.
2. On the Confirm Delete dialog, click **Yes**.

Attaching Security Policies to Service Bus Components

As you create your Service Bus components in the Service Bus Overview Editor, you can secure services by attaching security policies to proxy and business services. Security can also be defined at the message and transport levels for certain services. For more information about implementing policies, see [Securing Business and Proxy Services](#).

To attach a security policy to a proxy or business service:

1. In the Service Bus Overview Editor, right-click the proxy or business service to which you want to add security.

2. Select **Configure OSB WS Policies**.

The editor for the selected service appears with the Policies tab displayed.

3. Attach policies as described in [Attaching and Configuring Policies in JDeveloper](#).
4. When you are done, click **Save** in the JDeveloper toolbar.

Testing Service Bus Components in the Overview Editor

You can access the Run and Debug features of JDeveloper directly from the Service Bus Overview Editor to test and debug the components you create. Selecting Run or Debug for a component in the editor launches the Service Bus Test Console, where you can enter your test input and configure additional options for testing. Debugging lets you set breakpoints in pipelines and split-joins so you can step through the message flow and test it in manageable sections.

How to Test a Service Bus Component

Access the Test Console directly from the Overview Editor to test Service Bus components.

To test a Service Bus component:

1. In the Service Bus Overview Editor, right-click the component to test, and select **Run**.

The component to test can be a proxy service, business service, pipeline, or split-join. The Test Console appears in a web browser.

2. Enter the input in the Test Console, and then click **Execute**.

For information about setting properties in the Test Console, see [Test Console Page Reference for Services](#).

3. When you are done testing, click **Terminate** in the JDeveloper toolbar and select the name of the running process.

How to Debug a Service Bus Component

Use JDeveloper's debug feature to set breakpoints in pipelines and split-joins and test them in the Test Console.

To debug a Service Bus pipeline or split-join:

1. Set the breakpoints for the pipeline or split-join, as described in [How to Set Breakpoints on Service Bus Components](#).

2. In the Service Bus Overview Editor, right-click the pipeline or split-join to debug, and select **Debug**.

The Test Console appears in a web browser.

3. Enter the input in the Test Console, and then click **Execute**.

For information about setting properties in the Test Console, see [Test Console Page Reference for Services](#).

4. When you are done testing, click **Terminate** in the JDeveloper toolbar and select the name of the debugging process.

Deploying a Service Bus Application

In order to deploy a Service Bus application from JDeveloper, your instance of JDeveloper must have a connection to an Oracle WebLogic Server. You can also export Service Bus projects, import them into the Oracle Service Bus Console, and activate the projects from the console. For more information about deploying Service Bus applications, see [Deploying Oracle Service Bus Services](#).

Part II

Working with Oracle Service Bus Resources

This part describes the primary resources that you can create and use in Service Bus projects, and provides instructions for creating and maintaining those resources.

This part includes the following chapters:

- [Creating and Configuring Project Resources](#)
- [Creating and Configuring System Resources](#)
- [Creating and Configuring Proxy Services](#)
- [Creating and Configuring Business Services](#)
- [Improving Service Performance with Split-Join](#)
- [Working with WSDL Documents](#)

Information about pipelines is provided in a separate part, [Working with Oracle Service Bus Pipelines](#).

Creating and Configuring Project Resources

This chapter provides information about project resources you can create to support your Service Bus services, and provides links to additional resources. Project resources can be shared and re-used among services throughout a session. Local resources include things like authentication accounts, JAR files, MQ connections, email or JMS alert destinations, mappings, and so on. Several of these resources can be referenced from proxy and business services, and need to be created before you can configure the proxy or business services that use them.

This chapter includes the following topics:

- [Introduction to Service Bus Project Resources](#)
- [Working with Service Accounts](#)
- [Working with Service Key Providers](#)
- [Working with Alert Destinations](#)
- [Working with XML Schemas](#)
- [Working with XML Documents](#)
- [Working with JAR Files](#)

The following topics provide information and instructions for additional local resource types:

- [Working with WSDL Documents](#)
- [WADL Documents for REST Services in Service Bus](#)
- [Transforming Data with XQuery](#)
- [Transforming Data with XSLT](#)
- [Mapping Data with Cross-References](#)
- [Mapping Data with Domain Value Maps](#)
- [Defining Data Structures with Message Format Language](#)
- [Working with JCA Binding Resources](#)
- [Working with JavaScript Resources](#)
- [Working with MQ Connections](#)

- "Configuring Throttling for a Group of Business Services" in Administering Oracle Service Bus

Introduction to Service Bus Project Resources

Service Bus project resources refer to those resources that can be referenced by several components within an application or session. They are also known as local resources, and can include things like service accounts, which define authentication information for remote servers; XML documents and schemas; JAR files; XSLT and XQuery mappings; WSDL and WADL files; and so on. Certain Service Bus components require access to specific project resources, and those project resources must be created before you can create the components that rely on them. For example, if you create a proxy service with an email transport, you must first create the service account that defines the login information for the email server. In order to create an MQ proxy or business service, you must first create the MQ connection resource.

Project Resources and Sessions in the Oracle Service Bus Console

When you create, modify, or delete project resources in the Oracle Service Bus Console, you must be in an active session. If you discard the session, any project resources you created and the associated data are also discarded. When you activate a session after creating or modifying resources, Service Bus makes that information available to the runtime.

Working with Service Accounts

A service account provides a user name and password that proxy services and business services use for outbound authentication or authentication to a local or remote resource, such as an FTP server or a JMS server. The user names and passwords that you define for Service Bus access are used for inbound authentication and for authenticating administrative requests. For example, if a business service is required to supply a user name and password for transport-level authentication with a web service, you create a service account that specifies the user name and password, then you configure the business service to include the service account credentials in its outbound requests.

Service Account Authentication Types

You can use the same service account for multiple business services and proxy services. To specify the user name and password that a service account provides, you can define any of the following types:

- [Static](#)
- [User Name and Password Pass-Through](#)
- [User Mapping Authentication](#)

Static

With a static type of authentication, you save a user name and password with the service account configuration. The service account encodes this user name and password in the outbound request. Use this type of authentication when the login information does not need to change for different messages.

User Name and Password Pass-Through

A pass-through service account provides the user names and passwords that it receives from incoming client requests. For example, if an inbound HTTP basic request contains "pat" and "patspassword" as the user name and password, the service account encodes "pat" and "patspassword" in the outbound request.

Because this type requires that client requests include clear-text user names and passwords, it is applicable only for client requests that use either the HTTP basic protocol, a Web Services Security Username Token authentication with a clear-text password, or a custom user name and password token.

Oracle recommends that you use this technique only when Service Bus and the endpoint belong to the same authentication domain. For example, use this technique when you are routing messages within a single organization and both Service Bus and the message consumer authenticate against a common LDAP server.

The following restrictions apply to this technique:

- It cannot be used in outbound requests that authenticate Service Bus to a local or remote server or system resource, such as an FTP server or a JMS server.
- It cannot be used with the `fn-bea:lookupBasicCredentials` XQuery function. For more information, see [Service Bus XQuery Functions](#).

Note:

If your proxy is an active WSS intermediary, you can use WS-Security to encrypt a WS-Security Username Token or custom user name and password. In this instance, user name and password pass-through works because the proxy first decrypts the request and then has access to the clear-text user name and password.

User Mapping Authentication

A service account that uses mapping authentication maps the user name from one or more authenticated clients to user names and passwords that you specify. The mapping authentication type requires you to correlate (map) the user name obtained by authenticating an inbound request from a client (the local user name) to a user name and password that you specify (the remote user name and password). When the service account receives a request from an authenticated client that has been mapped, it provides the appropriate remote user name and password for the business service or proxy service outbound request.

If the client authenticates at both the transport level and message level, the service account maps the message level user name to the remote user name and password. You can also map an anonymous user name to a remote user name and password.

The following restrictions apply to mapping authentication:

- It cannot be used in outbound requests that authenticate Service Bus to a local or remote server or system resource, such as an FTP server or a JMS server.
- It cannot be used with the `fn-bea:lookupBasicCredentials` XQuery function. For more information, see [Service Bus XQuery Functions](#).

How to Create Service Accounts

Use service accounts to provide authentication information to proxy and business services for outbound authentication or for resource authentication, such as FTP and JMS servers. Service accounts can define authentication information in the following ways:

- Providing a static user name and password.
- Passing the incoming user name and password through to the server.
- Mapping the incoming user name and password to a user name and password you specify.

For more information about these authentication types, see [Service Account Authentication Types](#). For more information about service account properties, see the online help provided with Service Bus.

Creating a Service Account that Passes Though Authentication Information

To create a service account that passes through authentication information:

1. Do one of the following:
 - For JDeveloper: In the Application Navigator, right-click the project or folder to contain the new service account, point to **New**, and select **Service Account**.
 - For Oracle Service Bus Console: In the Project Navigator, right-click the project or folder to contain the new service account, point to **Create**, and select **Service Account**.
2. Enter a unique name for this service account, and an optional description.
3. Click **Create** or **Finish**.

The Service Account Definition Editor appears. By default, the type is Pass Through.

4. Click **Save**.

The service account is created and saved in the current session.

Creating a Service Account with a Static Password

To create a service account with a static password:

1. Create a service account, as described above.
2. On the Service Account Definition Editor, select **Static** for the type.

Tip:

If you are working in the Oracle Service Bus Console, you can also select **Static** for the type on the Create Service Account dialog when you first create the resource.

User name and password fields appear in the editor.

3. Enter the user name and password, and then confirm the password you entered.

4. In the toolbar, click **Save**.

Creating a Service Account that Maps Incoming Passwords

To create a service account that maps incoming passwords:

1. Create a service account, as described above.
2. On the Service Account Definition Editor, select **Mapping** for the type.

Tip:

If you are working in the Oracle Service Bus Console, you can also select **Mapping** for the type on the Create Service Account dialog when you first create the resource.

Mapping tables appear in the editor.

3. In the Remote Users table, do the following:
 - a. Click the **Add** icon above the table.
 - b. In the new row that appears, enter the user name and password that you want to send in outbound requests.
 - c. Repeat the above steps for each remote user to add.
 - d. To remove a remote user, select the row in the table and click the **Delete** icon.
4. To map remote users to local users, do the following in the Local Users table:
 - a. Click the **Add** icon above the table.
 - b. In the **Local User Name** column, enter the name that identifies a client that has been authenticated on its inbound request.
 - c. From the **Remote User Name** list of options, select the user name to send in outbound requests for the authenticated user you specified in the **Local User Name** field.

The list of options is populated from the values you created in the Remote Users table.
 - d. Repeat the above steps for each local user to add.

Note:

If you have not already added these users in Fusion Middleware Control, do so before you use this mapping in a runtime environment. Otherwise, the mapping will never match an authenticated user and will never be used. For more information about adding users, see "Creating Oracle Service Bus Users" in *Administering Oracle Service Bus*.

- e. To remove a local user, select the row in the table and click the **Delete** icon.
5. To map anonymous requests to a specific remote user account, select **Map Anonymous Requests to Remote User**, and then select the user name from the list of options.

This list of options is also populated from the values you created in the Remote Users table.

6. In the toolbar, click **Save**.

How to Edit Service Accounts

Once you create a service account you can modify its description and authentication type, including updating static login credentials, and adding and removing remote and local users for mapped authentication.

To edit a service account:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the service account to edit.
2. Right-click the service account name, and select **Open**.
3. Make any of the following changes:
 - Update the description.
 - Change the authentication type. Make sure to reconfigure the authentication information, as described in [How to Create Service Accounts](#).
 - For mapped authentication types: Add or remove remote and local users, modify remote user passwords, modify local user mappings, and update anonymous user mappings.

For more information about these tasks, see [Creating a Service Account that Maps Incoming Passwords](#) and the online help.

You cannot change the service account name.

4. When you are done making changes, click **Save**.
5. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Note:

If the service account that you modified authenticates with a WebLogic JMS server, the JMS server might not recognize your modification for up to 60 seconds. By default, WebLogic Server JMS checks permissions for each destination every 60 seconds. To change this behavior, modify the WebLogic Server startup command by setting the following system property to the frequency (in seconds) that you want WebLogic Server JMS to check permissions: `weblogic.jms.securityCheckInterval`.

A value of 0 (zero) for this property ensures that a permissions check is performed for every `send`, `receive`, and `getEnumeration` action on a JMS resource.

How to Delete Service Accounts

When you delete a service account, the user name, password, or local-user to remote-user mapping data that the service account defines are also deleted. You can delete the service account even if it is referenced by other resources, though this might result in conflicts due to unresolved references to the deleted resource.

Before you Begin:

If any business service or proxy service is configured to use the service account, remove the service account from the business service or proxy service. In the Oracle Service Bus Console, open the service account in the Service Account Definition Editor and click the **References** icon in the upper right to find out whether any services are using it. In JDeveloper, right-click the service account and select **Explore Dependencies**.

To delete a service account:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the service account to delete.
2. Right-click the service account, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the service account. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Working with Service Key Providers

A service key provider contains Public Key Infrastructure (PKI) credentials that proxy services use for decrypting inbound SOAP messages and for outbound authentication and digital signatures. A PKI credential is a private key paired with a certificate that can be used for digital signatures and encryption (for Web Services Security) and for outbound SSL authentication. The certificate contains the public key that corresponds to the private key.

Note:

- To use a service key provider, you must configure a PKI credential mapping provider. For information on doing this, see [Configuring the Oracle WebLogic Security Framework: Main Steps](#).
 - In earlier versions of Service Bus, service key providers were called proxy service providers.
-
-

A single service key provider can contain all of the following PKI credentials:

- A key-pair for digital encryption

Proxy services use this key-pair to decrypt inbound SOAP messages that have been encrypted to conform with a Web Services Policy statement. If you want the service key provider to support digital encryption, the key store that is associated with the PKI credential mapper must contain at least one X.509 certificate that supports encryption.

- A key-pair for digital signatures

Proxy services use this key-pair when its endpoint is a web service and the web service requires clients to sign one or more parts of a SOAP envelope.

- A key-pair for SSL client authentication (two-way SSL)

Proxy services use this key-pair to authenticate when acting as a client during an outbound TLS/SSL (Secure Sockets Layer) connection; that is, when routing a message to an HTTPS business service or proxy service that requires client certificate authentication.

You can use the same service key provider for multiple proxy services.

How to Create Service Key Providers

When you associate an encryption key service key provider with a proxy service, Service Bus embeds the X.509 certificate into the proxy service's WSDL file. The proxy service then uses this certificate to encrypt the messages that it sends to its endpoint. The proxy service uses the private key in the PKI credential to decrypt the messages that the endpoint returns.

To create a service key provider:

1. Do one of the following:
 - For JDeveloper: In the Application Navigator, right-click the project or folder to contain the new service key provider, point to **New**, and select **Service Key Provider**.
 - For Oracle Service Bus Console: In the Project Navigator, right-click the project or folder to contain the new service key provider, point to **Create**, and select **Service Key Provider**.
2. Enter a unique name for this service key provider, and an optional description.
3. Click **Create** or **Finish**.

The Service Key Provider Definition Editor appears.

4. To configure an encryption key, do the following:
 - a. Next to **Encryption Key**, click the **Browse** icon.

The Select an alias for Encryption Key window displays the key aliases from the key store that your realm's PKI credential mapper uses.
 - b. Enter the password you use to secure access to the key store. (You set this password when you create the keystore.)
 - c. Select a key alias that maps to an X.509 certificate and that supports encryption.
 - d. Click **Submit**.
5. To configure a digital signature key, do the following:
 - a. Next to **Digital Signature Key**, click the **Browse** icon.

The Select an alias for Digital Signature Key window displays the key aliases from the key store that your realm's PKI credential mapper uses.
 - b. Enter the password you use to secure access to the key store. (You set this password when you create the keystore.)
 - c. Select a key alias.

- d. Click **Submit**.
6. To configure an SSL client authentication key for two-way SSL, do the following:
 - a. Next to **SSL Client Authentication Key**, click the **Browse** icon.
The Select an alias for SSL Client Authentication Key window displays the key aliases from the key store that your realm's PKI credential mapper uses.
 - b. Enter the password you use to secure access to the key store. (You set the password when you create the keystore.)
 - c. Select a key alias.
 - d. Click **Submit**.
7. In the toolbar, click **Save**.
8. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Edit Service Key Providers

Once you create a service key provider, you can reconfigure the key information.

To edit a service key provider:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the service key provider to edit.
2. Right-click the service account name, and select **Open**.
3. To make a change to the fields, click the icons to the right of the fields to select, remove, or edit a key.

For information about the fields you can edit, see [How to Create Service Key Providers](#) and the online help provided with Service Bus.

4. When you are done making changes, click **Save** in the toolbar.
5. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Delete Service Key Providers

When you delete a service key provider, Service Bus also deletes the associated alias to key-pair bindings from the PKI credential mapping provider. Service Bus does not delete the associated key-certificate pair from the key store. You can delete the service key provider even if it is referenced by other resources, though this might result in conflicts due to unresolved references to the deleted resource.

Before you Begin:

If any proxy service is configured to use the service key provider, remove the service key provider from the proxy service. In the Oracle Service Bus Console, open the service key provider in the Service Key Provider Definition Editor and click the **References** icon in the upper right to find out if it has any references. In JDeveloper, right-click the service key provider and select **Explore Dependencies**.

To delete a service key provider:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the service key provider to delete.
2. Right-click the name of the service key provider, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the service key provider. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Working with Alert Destinations

An alert destination resource defines a list of recipients that can receive alert notifications from Service Bus. You can configure each alert destination resource to include a set of recipients according to a given context and then associate the resource with the alerts you define. Alert destinations give you the flexibility to specify whether alerts are sent to SNMP traps, are collected for reporting, are logged to the local server's alert log, or sent to e-mail recipients or JMS destinations.

When you configure an Alert action in a pipeline, or an SLA alert rule for a service, the configuration includes specifying an alert destination, which defines who gets notified when alerts are generated. In the case of email and JMS destinations, a destination resource can include a list of email addresses or JMS URIs, respectively. You can reuse alert destinations across alert configurations.

Alert Destination Types

For each alert destination, you can specify that the alerts be sent to multiple types of destinations, as described in the following sections.

- [Email](#)
- [SNMP Traps](#)
- [Reporting](#)
- [Alert Logging](#)
- [JMS](#)

Email

Alert notifications can be sent to multiple emails addresses. To configure an email alert destination, you need an SMTP server global resource or a JavaMail session in Oracle WebLogic Server. When an alert is delivered, email metadata consisting of the details about the alert is prefixed to the details of the payload. For information about SMTP server resources, see [Working with SMTP Server Resources](#). For information about configuring JavaMail sessions, see "Configure Access to JavaMail" in the *Oracle WebLogic Server Administration Console Online Help*.

SNMP Traps

Simple Network Management Protocol (SNMP) traps allow any third-party software to monitor service level agreements within Service Bus. With SNMP notifications

enabled, Web Services Management (WSM) and Enterprise Service Management (ESM) tools can monitor SLA violations and pipeline alerts.

SNMP is an application-layer protocol which allows the exchange of information on the management of a resource across a network. It enables you to monitor a resource and, if required, take some action based on the data obtained from the resource. Service Bus supports SNMP version 1 and 2. SNMP includes the following components:

- Managed Resource
- Management Information Base (MIB)
- SNMP Agent
- SNMP Manager
- Network Management System (NMS)

Reporting

The Reporting destination lets you send notifications of pipeline alerts and SLA alerts to a custom reporting provider that can be developed using the reporting APIs provided with Service Bus. This allows third parties to receive and process alerts in custom Java code.

Alert Logging

Each alert destination lets you configure whether or not the alerts sent to that destination are logged. Logged alerts are sent to the local alert log. Each Service Bus server has its own alert log. In a cluster, the Admin Server collects the alert logs from all Managed Servers and aggregates the alerts for logging.

JMS

Alert notifications can be sent to one or more Java Messaging Service (JMS) queues or topics. You must configure a JNDI URL for the JMS destination for alerts, create a JMS connection factory and a queue or topic, and target them to the appropriate JMS server in the Oracle WebLogic Server Administration Console. For information, see "Methods for Configuring JMS System Resources" in *Administering JMS Resources for Oracle WebLogic Server*. When you define the JMS alert destination you can either use a destination queue or a destination topic. The message type can be bytes or text. For more information about how to configure JMS alert destination see "Adding JMS Destinations."

How To Create Alert Destinations

Alerts are aggregated at runtime, and you can view them on the Service Bus Dashboard in Fusion Middleware Control.

To create an alert destination:

1. Do one of the following:
 - For JDeveloper: In the Application Navigator, right-click the project or folder to contain the new alert destination, point to **New**, and select **Alert Destination**.
 - For Oracle Service Bus Console: In the Project Navigator, right-click the project or folder to contain the new alert destination, point to **Create**, and select **Alert Destination**.

2. Enter a unique name for this alert destination, and an optional description.
3. Click **Create** or **Finish**.

The Alert Destination Definition Editor appears.
4. Select any of the following destinations to include them in this alert destination resource. In JDeveloper, select **Yes** to select a destination type.
 - **SNMP Trap**: Alerts are sent as SNMP traps, and can be processed by any third-party enterprise monitoring systems.
 - **Reporting**: Alerts are sent to the Service Bus reporting module and can be captured using a custom reporting provider developed using the reporting APIs provided with Service Bus. This allows third-parties to receive and process alerts in custom Java code.
 - **Alert Logging**: Alerts sent to this alert destination are logged to the alert log.
5. To add email recipients to the alert destination resource definition, see [How to Define Email Recipients for an Alert Destination](#).
6. To add JMS destinations to the alert destination resource definition, see [How to Define JMS Recipients for an Alert Destination](#).
7. Click **Save**.

How to Define Email Recipients for an Alert Destination

Before you add an email destination, you must configure an SMTP server (see [How to Create SMTP Server Resources](#)), or a JavaMail session in WebLogic Server. If there are no SMTP server resources or JavaMail sessions available, configured, you cannot configure an email recipient.

To add or update email recipients in an alert destination:

1. Create an alert destination, as described in [How To Create Alert Destinations](#).
2. Above the email Recipients table, click **Add**.

In JDeveloper, email configuration fields appear in the bottom of the page. In the Oracle Service Bus Console, the Add email Recipients dialog appears.

3. In the **Mail Recipients** field, enter an email recipient in the format `mailto:username@hostname`.

To specify multiple email recipients, enter the user names and hostnames in a comma-separated list. For example,
`mailto:username@hostname[,username_1@hostname_1] . . .
[,username_n@hostname_n]`

Only the first mail recipient needs to be prefixed with the text "mailto:".

4. To send messages over secure sockets layer (SSL), select **SSL Required**.
5. Do only one of the following:
 - To use an SMTP server for outgoing mail, click in the **SMTP Server** field and select the name of the SMTP server to use.

- To use a Java Mail session, click in the **Mail Session** field and select an available mail session.
6. In the **From Name** field, enter the sender's name for the alert notification.
 7. In the **From Address** field, enter the sender's email address.
This field is required if a value for **From Name** is specified.
 8. In the **Reply To Name** field, enter a name to which replies are addressed.
 9. In the **Reply To Address** field, enter an email address to which replies are sent.
This field is required if a value for **Reply To Name** is specified.
 10. In the **Connection Timeout** field, enter the number of milliseconds a connection must wait for a response from the server before timing out.
 11. In the **Socket I/O Timeout** field, enter the number of milliseconds for a socket I/O timeout when waiting for a response from the server.
 12. In the **Request Encoding** field, enter a character set encoding value.
The default encoding value is `iso-8859-1`.
 13. If you are using Oracle Service Bus Console, click **OK** to close the dialog.
 14. To make changes to an email recipient, select that row in the table and click **Edit** to the upper right of the table. Modify any of the above fields.
 15. To delete an email recipient, select that row in the table and click **Delete** to the upper right of the table.
 16. Click **Save**.

How to Define JMS Recipients for an Alert Destination

While WebLogic Server allows forward slashes in JNDI names, such as "myqueues/myqueue", JNDI names with forward slashes interfere with the URI format required by Service Bus, and you cannot use those names. To work around this issue, define a JMS foreign server and reference that foreign server in the URI. For more information, see *Configure Foreign Servers* in the *Oracle WebLogic Server Administration Console Online Help*.

To add or update JMS destinations in an alert destination:

1. Create an alert destination, as described in [How To Create Alert Destinations](#).
2. Above the JMS Destinations table, click **Add**.
In JDeveloper, JMS configuration fields appear in the bottom of the page. In the Oracle Service Bus Console, the Add JMS Destination dialog appears.
3. In the **Destination URI** field, enter a JMS destination URI in the format `jms://host:port/factoryJndiName/destJndiName`.
4. In the **Destination Type** field, select **Queue** or **Topic**.
5. In the **Message Type** field, select **Bytes** or **Text**.
6. In the **Request Encoding** field, enter a character set encoding value.

The default encoding value is UTF-8.

7. If you are using Oracle Service Bus Console, click **OK** to close the dialog.
8. To make changes to a JMS destination, select that row in the table and click **Edit** to the upper right of the table. Modify any of the above fields.
9. To delete a JMS destination, select that row in the table and click **Delete** to the upper right of the table.
10. Click **Save**.

How to Edit Alert Destinations

Once you create an alert destination you can modify its description, and add, update, or remove email recipients and JMS destinations.

To edit an alert destination:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the alert destination to edit.
2. Right-click the alert destination name, and select **Open**.
3. Modify the description or any of the configuration details:
 - To change the type of destinations to which alerts are sent, select or deselect any of the following types: SNMP Trap, Reporting, and Alert Logging.
In JDeveloper, select **Yes** to select a type and select **No** to deselect it.
 - To add, update, or delete email recipients, see [How to Define Email Recipients for an Alert Destination](#).
 - To add, update, or delete JMS destinations, see [How to Define JMS Recipients for an Alert Destination](#).
4. When you are done making changes, click **Save**.
5. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Delete Alert Destinations

When you delete alert destinations, you need to update any alert actions or rules that reference the resource. To delete specific email recipients or JMS destinations from an alert destination, see [How to Define Email Recipients for an Alert Destination](#) or [How to Define JMS Recipients for an Alert Destination](#).

Before you Begin:

If the alert destination has any references, remove them before deleting it. In the Oracle Service Bus Console, open the alert destination in the Alert Destination Definition Editor and click the **References** icon in the upper right to find out whether there are any references. In JDeveloper, right-click the alert destination and select **Explore Dependencies**.

To delete an alert destination:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the alert destination to delete.

2. Right-click the alert destination, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the alert destination. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Working with SNMP

SNMP is an application-layer protocol that allows the exchange of information on the management of a resource across a network. SNMP lets you to monitor a resource and, if required, take an action based on the data obtained from the resource. For more information about SNMP, see *Monitoring Oracle WebLogic Server with SNMP*.

Guidelines for Working with SNMP Agents for Service Bus

You can create and target SNMP agents in an existing Oracle WebLogic domain to trap SNMP messages generated by Service Bus. For instructions on creating and targeting SNMP agents, see the following topics in the *Oracle WebLogic Server Administration Console Online Help*:

- Create SNMP agents
- Target SNMP agents
- Create trap destinations

You can create SNMP agents that are either domain-scoped or server-scoped. Domain-scoped agents, which are targeted to the domain rather than to an individual server, are for backward compatibility and are being deprecated. Oracle recommends that you create server-scoped agents.

When creating and targeting an SNMP server-scoped agent for Service Bus, use the following guidelines:

- **Targeting the Agent:** When targeting an SNMP agent to Service Bus, target only the Service Bus Admin Server. Only agents targeted to the Admin Server receive alerts from Service Bus. Agents targeted to Managed Servers do not receive SNMP messages.
- **Creating a Trap Destination:** Enter the following settings for the destination:
 - **Name:** alsbDestination-0
 - **Community:** weblogic
 - **Host and Port:** Set the values to point to the host and port where the SNMP manager is listening for these alerts, such as localhost and 163.

How to Start Listening for Traps

You can run a command-line utility to listen for the traps generated within Service Bus.

To start listening for traps:

1. In a command window, change directories to `WL_ORACLE_HOME/server/bin`, and run the following command:

```
setWLSEnv.cmd(.sh)
```

2. Run the following Java command to start the WebLogic Server SNMP command line utility, which listens for traps and prints them on the server console (using 163 as the listen port for traps):

```
java weblogic.diagnostics.snmp.cmdline.Manager SnmpTrapMonitor -p 163
```

From then on, the generated traps should reach the running command line utility.

Note:

On Solaris, port numbers 0 to 1023 are reserved for root login. If you want to use ports 161 and 163 (as used in this procedure), you may have to start the server and command line utility using root login. To avoid this problem, and to avoid using the root login, specify port numbers above 1023 for both the SNMP agent and the SNMP manager command.

Working with XML Schemas

Schemas describe types for primitive or structured data. XML schemas are an XML vocabulary that describe the rules XML business data must follow. XML schemas specify the structure of documents, and the data type of each element and attribute contained in the document.

You use XML schemas as references for WSDL resources and to validate an element specified with an XPath expression in a pipeline. For more information, see [Adding Validate Actions in the Console](#).

How to Create XML Schemas

XML schemas are a standard feature in JDeveloper. For information about the editors and tools you use to create XML schemas, see *Developing Applications Using XML in Developing Applications with Oracle JDeveloper*.

If you are using the Oracle Service Bus Console, you can create XML schemas by importing them or by creating an XML schema resource. For more information on importing, see [Importing and Exporting Resources and Configurations](#). Use the following procedure to manually create XML schema resources.

To create an XML schema in the Oracle Service Bus Console:

1. In the Project Navigator, right-click the project or folder to contain the new XML schema, point to **Create**, and select **Schema**.

The Create Schema dialog appears.

2. Do one of the following:

- To create the resource from an existing schema file, click **Browse** next to the **File Upload** field and then navigate to and select the file to use.

The **Resource Name** field is automatically populated with the file name minus the file extension. You can change this name.

- To create a new XML schema, enter a unique name for the XML schema resource.

3. Optionally, enter a brief description of the resource.
4. Click **Create**.

The XML schema elements, if defined, appear in the Schema Definition Editor.
5. To modify the schema, do the following:
 - a. Click **Edit Source** in the toolbar.

The Edit Source dialog appears.
 - b. To browse to and select a new schema file to upload, click **Browse**.
 - c. To modify the contents of the file, update the code directly in the Contents section of the dialog.
 - d. Click **Save**.
6. In the Schema Definition Editor toolbar, click **Save**.
7. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Edit XML Schemas

XML schemas are a standard feature in JDeveloper. For information about editing XML schemas, see Developing Applications Using XML in *Developing Applications with Oracle JDeveloper*.

If you are using the Oracle Service Bus Console, use the following procedure to edit XML schemas.

To edit an XML schema in the Oracle Service Bus Console:

1. In the Project Navigator, expand the project and folders containing the XML schema to edit.
2. Right-click the XML schema name, and select **Open**.
3. Click **Edit Source** in the toolbar.

The Edit Source dialog appears.
4. To browse to and select a new XML schema file to upload, click **Browse**.
5. To modify the contents of the file, update the code directly in the Contents section of the dialog.
6. Click **Save**.
7. In the Schema Definition Editor toolbar, click **Save**.
8. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Delete XML Schemas

If any resources reference the XML schema you want to delete, remove those references before deleting the XML schema. In the Oracle Service Bus Console, open the XML schema in the Schema Definition Editor and click the **References** icon in the upper right to find out if it has any references. In JDeveloper, right-click the XML schema and select **Explore Dependencies**.

You can delete the XML schema even if it is referenced by other resources, though this might result in conflicts due to unresolved references to the deleted resource.

To delete an XML schema:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the XML schema to delete.
2. Right-click the name of the schema, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the schema. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Working with XML Documents

XML document resources store XML files for use in proxy or business service configurations. For example, you can create XML document resources for TopLink mapping files needed in JCA proxy or business services that communicate with JCA-compliant systems.

For more information about JCA services, see [Using the JCA Transport and JCA Adapters](#).

How to Create XML Documents

XML documents are a standard feature in JDeveloper. For information about the editors and tools you use to create XML files, see "Developing Applications Using XML" in *Developing Applications with Oracle JDeveloper*.

If you are using the Oracle Service Bus Console, the easiest way to create XML documents is to use the import feature. For example, if you import JCA resources (JCA file, associated WSDL file, and TopLink mapping file), Service Bus automatically generates XML document resources out of mapping files and maintains the dependencies among resource files. For more information on importing, see [Importing and Exporting Resources and Configurations](#).

If you do not bulk import, use the following procedure to manually create XML documents.

To create XML documents in the Oracle Service Bus Console:

1. In the Project Navigator, right-click the project or folder to contain the new XML document, point to **Create**, and select **XML Document**.

The Create XML Document dialog appears.

2. Do one of the following:
 - To create the resource from an existing XML file, click **Browse** next to the **File Upload** field and then navigate to and select the XML file to use.
The **Resource Name** field is automatically populated with the file name minus the file extension. You can change this name.
 - To create a new XML document for the resource, enter a unique name for the XML document resource.

3. Optionally, enter a brief description of the resource.
4. Click **Create**.

The XML document appears in the XML Document Definition Editor.
5. To modify the XML code, do the following:
 - a. Click **Edit Source** in the toolbar.

The Edit Source dialog appears.
 - b. To browse to and select a new XML file to upload, click **Browse**.
 - c. To modify the contents of the file, update the code directly in the Contents section of the dialog.
 - d. Click **Save**.
6. In the XML Document Definition Editor toolbar, click **Save**.
7. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Edit XML Documents

XML documents are a standard feature in JDeveloper. For information about editing XML files, see Developing Applications Using XML in *Developing Applications with Oracle JDeveloper*.

If you are using the Oracle Service Bus Console, use the following procedure to edit XML files.

To edit an XML document in the Oracle Service Bus Console:

1. In the Project Navigator, expand the project and folders containing the XML document to edit.
2. Right-click the XML document name, and select **Open**.
3. Click **Edit Source** in the toolbar.

The Edit Source dialog appears.
4. To browse to and select a new XML file to upload, click **Browse**.
5. To modify the contents of the file, update the code directly in the Contents section of the dialog.
6. Click **Save**.
7. In the XML Document Definition Editor toolbar, click **Save**.
8. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Delete XML Documents

If any resources, such as JCA bindings, reference the XML document you want to delete, remove those references before deleting the XML document. In the Oracle Service Bus Console, open the XML document in the XML Document Definition Editor and click the **References** icon in the upper right to find out if it has any references. In JDeveloper, right-click the XML document and select **Explore Dependencies**.

To delete an XML document:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the XML document to delete.
2. Right-click the name of the document, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the XML document. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Working with JAR Files

A JAR (Java Archive) is a zipped file that contains a set of Java classes. JAR files stored the compiled Java classes and associated metadata that can constitute a program. A JAR file acts like a callable program library for Java code elements so a single compilation link provides access to multiple elements, rather than requiring bindings for each element individually.

To use JAR files in a Service Bus project, you upload them to JAR resources. JAR files in Service Bus are used in:

- Java callout actions
- EJB-based business services
- JEJB services
- Tuxedo-based proxy and business services

How to Add JAR Files

JAR file integration is a standard feature in JDeveloper. For information about adding JAR files and libraries to your Service Bus projects, see *How to Manage Libraries in Developing Applications with Oracle JDeveloper*.

If you are using the Oracle Service Bus Console, you can add JAR files by either importing them into a Service Bus project or uploading them into archive resources. For more information on importing, see [Importing and Exporting Resources and Configurations](#). Use the procedure below to upload a JAR file into an archive resource.

To add a JAR file using the Oracle Service Bus Console:

1. In the Project Navigator, right-click the project or folder to contain the new JAR file, point to **Create**, and select **Archive**.
The Create Archive dialog appears.
2. Click **Browse** next to the **File Upload** field and then navigate to and select the JAR file to upload.
The **Resource Name** field is automatically populated with the file name minus the file extension. You can change this name.
3. Optionally, enter a brief description of the resource.

4. Click **Create**.

The configuration details and dependencies for the JAR file appear in the Archive Definition Editor.

5. To add dependencies, do the following:

- a. Click **Add** above the dependencies table.

A new row appears in the table.

- b. Click in the new row in the name column, and then click **Browse**.

The Search and Select dialog appears.

- c. Enter a file name or the path to the file, and click **Search**.

- d. In the results list, select the name of the file containing the dependency, and then click **OK**.

6. Repeat the above steps for each dependency to add. Use the up and down arrows above the Dependencies table to re-order the list of dependencies.

7. In the Archive Definition Editor toolbar, click **Save**.

8. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Update a JAR File

Updating an archive resource essentially involves pointing the resource to a new version of the JAR file or to a different JAR file, as you cannot edit a JAR using Service Bus. JAR file integration is a standard feature in JDeveloper. For information about updating JAR files, see How to Manage Libraries in *Developing Applications with Oracle JDeveloper*.

If you are using the Oracle Service Bus Console, use the following procedure to update a JAR file in an archive resource.

Caution:

If you update the JAR file used by an EJB business service by updating to a newer version of the file, you must redeploy the EJB, edit any EJB service that uses the JAR file, and reselect the JAR resource, save, and activate. This repackages the EJB business service to use the new JAR.

Java callout actions and Tuxedo-based services automatically pick up the new JAR file.

To update a JAR file in the Oracle Service Bus Console:

1. In the Project Navigator, expand the project and folders containing the archive resource to edit.
2. Right-click the archive resource name, and select **Open**.
3. To update the JAR file, do the following:
 - a. Click **Upload New JAR File** in the toolbar.
The Upload New JAR File dialog appears.

archive resource in the Archive Definition Editor and click the **References** icon in the upper right to find out whether it has any references. In JDeveloper, right-click the JAR file and select **Explore Dependencies**.

You can delete the JAR file even if it is referenced by other resources, though this might result in conflicts due to unresolved references to the deleted resource.

To delete a JAR file:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the archive resource for the JAR file to delete.
2. Right-click the name of the resource, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the JAR file. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Creating and Configuring System Resources

This chapter describes how to add and configure system resources, such as JNDI providers, SMTP servers, and proxy servers. System resources can be reused by Service Bus services in all projects in the session.

This chapter includes the following sections:

- [Working with JNDI Provider Resources](#)
- [Working with SMTP Server Resources](#)
- [Working with Proxy Server Resources](#)

UDDI registries are also system resources in Service Bus. You can learn more about them in [Working with UDDI Registries](#).

In the Oracle Service Bus Console, you create system resources in the System project, which by default includes a folder for each type of system resource. In JDeveloper, you create the resources in the **Service Bus System Resources** folder in the Application Resources panel. In the file system, these resources are located in the `System` subfolder under the JDeveloper application folder.

Working with JNDI Provider Resources

JNDI provider resources perform the JNDI lookups for Service Bus projects, providing the protocols and security credentials required for accessing remote servers. You can use any protocol for the JNDI provider, including HTTP, HTTPS, t3, t3s, IIOP, and IIOPS. Service Bus supports several initial context factory classes for the JNDI lookup, depending on the application server you are using. When connecting to JBoss Application Server using IIOP or IIOPS, use the `com.sun.jndi.cosnaming.CNCTXFactory` class (IIOP/S is not supported in JBoss 7.0.0 or later).

Classpath Requirements for JBoss Application Server

When using JBoss initial context factory classes, make sure to include the class and any dependent classes in the Service Bus server classpath. The following JAR files are required in the server post-classpath or `domain_name/lib` directory. These files are located in your JBoss installation in the `/client` directory.

- `jbossall-client.jar` for JBoss 4.x through 6.x
- `jboss-client.jar` for JBoss 7.x
- `log4j.jar`

Note:

If you do not want to add the client JAR file to the classpath, you can add the following files individually: `jboss-client.jar`, `jboss-common-core.jar`, `jboss-integration.jar`, `jboss-logging-spi.jar`, `jboss-remoting.jar`, `jboss-security-spi.jar`, `jboss-serialization.jar`, `jbossx-client.jar`, and `jnp-client.jar`.

In addition, add `jboss-ejb-client.properties` to the server classpath or library with the following properties defined:

- For unsecure connections:
 - `remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false`
 - `remote.connections=<name_of_connection>`
 - `remote.connection.conn1.host=<hostname>`
 - `remote.connection.conn1.port=<port>`
- For secure connections:
 - `remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false`
 - `remote.connections=<name_of_connection>`
 - `remote.connection.conn2.host=<hostname>`
 - `remote.connection.conn2.port=<port>`
 - `remote.connection.conn2.connect.options.org.xnio.Options.SASL_DISALLOWED_MECHANISMS=JBASS-LOCAL-USER`
 - `remote.connection.conn2.username=<username>`
 - `remote.connection.conn2.password=<password>`

When using the IIOP protocol, include the following JAR files in the Oracle WebLogic Server classpath. These files are located in your JBoss installation in `/server/all/lib`, except `logkit.jar`, which is located in the `/client` directory.

- `jacorb.jar`
- `avalon-framework.jar`
- `logkit.jar`

About JBoss Initial Context Factory Environment Properties

You can invoke EJBs deployed in JBoss Application Server using the EJB and JEJB transports. Service Bus supports the following initial context factory implementations for JBoss. You can specify environment properties for each of the classes for JBoss versions 4.x through 6.x.

- `org.jnp.interfaces.NamingContextFactory`

- `org.jboss.naming.HttpNamingContextFactory`
- `org.jboss.security.jndi.LoginInitialContextFactory`

For JBoss 7.x, you can specify environment properties for `org.jboss.naming.remote.client.InitialContextFactory`.

The following table lists the environment properties and indicates whether they are supported for each factory class. For more information about these classes and the properties you can set, see "The Naming InitialContext Factories" at <http://docs.jboss.org/jbossas/jboss4guide/r1/html/ch3.chapter.html>

Table 7-1 JBoss Initial Context Factory Environment Parameters

Parameter	NamingContext Factory (4.x-6.x)	HttpNaming ContextFactory (4.x-6.x)	LoginInitial ContextFactory (4.x-6.x)	InitialContext Factory (7.x only)
<code>java.naming.factory.initial</code>	Yes	Yes	Yes	Yes
<code>java.naming.provider.url</code>	Yes	Yes	Yes	Yes
<code>java.naming.factory.url.pkgs</code>	Yes	Yes	Yes	Yes
<code>jnp.socketFactory</code>	Yes	No	No	Yes
<code>jnp.timeout</code>	Yes	No	No	Yes
<code>jnp.sotimeout</code>	Yes	No	No	Yes
<code>java.naming.security.principal</code>	No	No	Yes	Yes
<code>java.naming.security.credentials</code>	No	No	Yes	Yes
<code>java.naming.security.protocol</code>	No	No	Yes	Yes

In addition, the following initial context properties may be required when looking up EJBs deployed on a JBoss cluster.

- `jnp.partitionName`
- `jnp.discoveryGroup`
- `jnp.discoveryPort`
- `jnp.discoveryTimeout`
- `jnp.disableDiscovery`

How to View JNDI Provider Resources in the Oracle Service Bus Console

The Folders Definition Editor for JNDI providers lists all the JNDI provider resources you have created in the current session. Use this page to quickly find and access the JNDI provider resources you have defined.

To view JNDI providers in the console:

1. Expand the **System** project, right-click **JNDI Providers**, and then select **Open**.
The Folder Definition Editor appears with a list of existing JNDI provider resources.

2. To locate specific JNDI providers, do the following:
 - a. If the query fields are not visible above the JNDI Providers table, click **Query by Example** in the table toolbar.
 - b. Enter the name of the JNDI provider resource you want to find above the **Name** column, and press **Enter**.

You can enter wildcard characters (? for a single character; * for multiple characters) to perform a more general search.
 - c. To view all JNDI providers again, clear the query fields and press **Enter**.
3. To view the configuration for a JNDI provider, click the resource name in the JNDI Providers table.
4. To delete a JNDI provider resource, select the name of the resource in the table and click **Delete**. See [How to Delete JNDI Provider Resources](#).

How to Create a JNDI Provider Resource

When you create a JNDI provider resource, you specify connection information for the remote server, including the URL, the initial context factory, security credentials, and, optionally, environment properties for JBoss context factories.

To create a JNDI provider resource:

1. Do one of the following:
 - If you are using JDeveloper, expand the Application Resources panel, right-click **Service Bus System Resources**, point to **New**, and then select **JNDI Provider**.

Tip:

To create JNDI provider resources directly in a project, making it a project-level resource, right-click the project, point to **New**, and then select **JNDI Provider**.
 - If you are using Oracle Service Bus Console, expand the **System** project, right-click **JNDI Providers**, point to **Create**, and then select **Create JNDI Provider**.

The Create JNDI Service dialog appears.

2. Enter a name and optional description for the resource, and then click **Finish** or **Create**.

The JNDI Provider Definition Editor appears.

3. Enter the URL for the JNDI provider in the format `protocol://hostname:port`.

For clusters, use a comma-separated list of Managed Servers, in the format: `protocol://hostname:ms1port, hostname:ms2port`

4. In the **Initial Context Factory** field, select the initial context factory class name for the JNDI provider you are creating.
5. In the **JNDI Request Timeout** field, enter the JNDI request timeout in milliseconds.

6. Select **JNDI Cache** to enable local caching of JNDI objects.
7. To add environment properties for JBoss initial context factories, click the **Add** icon above the Environment Parameters table, and enter the parameter name and value in the new line that appears.

For information about the parameters, see [About JBoss Initial Context Factory Environment Properties](#).
8. If access to the target JNDI provider requires a user name and password, enter a user name in the **User Name** field, and the associated password in the **Password** and **Confirm Password** fields.
9. Click **Save**.
10. If you are using the Oracle Service Bus Console, do any of the following in the toolbar in the upper right:
 - To reset the JNDI context to discard the JNDI connection and locally cached objects, click the **Reset** icon.
 - To test whether the JNDI provider can successfully establish a connection and obtain a JNDI context, click the **Test** icon.
11. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Edit JNDI Provider Resources

Once you create a JNDI provider resource, you can modify its description and most of the JNDI properties.

To edit a JNDI provider resource:

1. Expand the project and folders containing the resource to edit. This can be any of the following locations:
 - In JDeveloper, the **Service Bus System Resources** folder in the Application Resources panel.
 - In JDeveloper, the Service Bus project or folder in which the JNDI provider resource is located in the Application Navigator.
 - In Oracle Service Bus Console, the **JNDI Providers** folder in the **System** project.
2. Right-click the JNDI provider name, and select **Open**.
The JNDI Provider Definition Editor appears.
3. Modify any of the fields described in [How to Create a JNDI Provider Resource](#). The online help describes these fields in greater detail.

Note:

The **New Password** field is only editable if the JNDI provider was *not* configured with a user name and password.

4. When you are done making changes, click **Save**.

5. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Delete JNDI Provider Resources

When you delete a JNDI provider resource, any references to the resource from other Service Bus resources are broken. To find out whether any resources reference a JNDI provider, open the JNDI provider resource in the JNDI Provider Definition Editor and click the **References** icon in the upper right. In JDeveloper, right-click the JNDI provider and select **Explore Dependencies**.

To delete a JNDI provider resource:

1. Expand the project and folders containing the resource to edit. This can be any of the following locations:
 - In JDeveloper, the **Service Bus System Resources** folder in the Application Resources panel.
 - In JDeveloper, the Service Bus project or folder in which the JNDI provider resource is located in the Application Navigator.
 - In Oracle Service Bus Console, the **JNDI Providers** folder in the **System** project.
2. Right-click the JNDI provider resource, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the resource. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Working with SMTP Server Resources

SMTP server resources define connection properties for SMTP servers, and are used while configuring alert destination resources and business services based on the email transport. The SMTP server global resource captures the address of the SMTP server, port number, and if required, the authentication credentials. The authentication credentials are stored inline and are not stored as a service account. For information about alert destinations, see [Working with Alert Destinations](#).

How to View SMTP Server Resources in the Oracle Service Bus Console

The Folders Definition Editor for SMTP servers lists all the SMTP server resources you have created in the current session. Use this page to quickly find and access the SMTP server resources you have defined.

To view SMTP servers in the console:

1. Expand the **System** project, right-click **SMTP Servers**, and then select **Open**.
The Folder Definition Editor appears with a list of existing SMTP server resources.
2. To locate specific SMTP server resources, do the following:
 - a. If the query fields are not visible above the SMTP Servers table, click **Query by Example** in the table toolbar.

- b. Enter the name of the SMTP server resource you want to find above the **Name** column, and press **Enter**.
You can enter wildcard characters (? for a single character; * for multiple characters) to perform a more general search.
 - c. To view all SMTP servers again, clear the query fields and press **Enter**.
3. To view the configuration for an SMTP server, click the resource name in the SMTP Servers table.
 4. To delete an SMTP server resource, select the name of the resource in the table and click **Delete**. See [How to Delete SMTP Server Resources](#) .

How to Create SMTP Server Resources

When you create an SMTP server resource, you specify connection information for the server, including the URL and port number. Security information is only required if the server requires authentication.

To create an SMTP server resource:

1. Do one of the following:
 - If you are using JDeveloper, expand the Application Resources panel, right-click **Service Bus System Resources**, point to **New**, and then select **SMTP Server**.

Note:

To create SMTP server resources directly in a project, making it a project-level resource, right-click the project, point to **New**, and then select **SMTP Server**.

- If you are using Oracle Service Bus Console, expand the **System** project, right-click **SMTP Servers**, point to **Create**, and then select **Create SMTP Server**.

The Create SMTP Server dialog appears.

2. Enter a name and optional description for the resource.
3. If you are using the console, enter the SMTP server URL and port number.
4. Click **Finish** or **Create**.

The SMTP Server Definition Editor appears.

5. If you are using JDeveloper, enter the URL and port number to access the SMTP server.
6. If access to the SMTP server requires authentication, enter a user name in the **User Name** field, and the associated password in the **Password** and **Confirm Password** fields.
7. Click **Save**.
8. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Configure a Default SMTP Server

In the Oracle Service Bus Console, you can designate one of the configured SMTP servers as the default server for the domain.

To configure a default SMTP server:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.
2. In the Project Navigator, expand **System > SMTP Servers**.
3. Right-click the name of the SMTP server you want to be the default server, and then select **Set as Default**.
4. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Edit SMTP Server Resources

Once you create an SMTP server resource, you can modify its description and most of the server properties.

To edit an SMTP server resource:

1. Expand the project and folders containing the resource to edit. This can be any of the following locations:
 - In JDeveloper, the **Service Bus System Resources** folder in the Application Resources panel.
 - In JDeveloper, the Service Bus project or folder in which the SMTP server resource is located in the Application Navigator.
 - In Oracle Service Bus Console, the **SMTP Servers** folder in the **System** project.
2. Right-click the SMTP server name, and select **Open**.

The SMTP Server Definition Editor appears.

3. Modify any of the fields described in [How to Create SMTP Server Resources](#). The online help describes these fields in greater detail.
4. When you are done making changes, click **Save**.
5. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Delete SMTP Server Resources

When you delete an SMTP server resource, any references to the resource from other Service Bus resources are broken. To find out whether any resources reference an SMTP server, open the SMTP server resource in the SMTP Server Definition Editor and click the **References** icon in the upper right. In JDeveloper, right-click the SMTP server and select **Explore Dependencies**.

To delete an SMTP server resource:

1. Expand the project and folders containing the resource to edit. This can be any of the following locations:

- In JDeveloper, the **Service Bus System Resources** folder in the Application Resources panel.
 - In JDeveloper, the Service Bus project or folder in which the SMTP server resource is located in the Application Navigator.
 - In Oracle Service Bus Console, the **SMTP Servers** folder in the **System** project.
2. Right-click the SMTP server resource, and select **Delete**.
 3. If you are using JDeveloper, a confirmation dialog displays the number of references for the resource. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
 4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Working with Proxy Server Resources

Proxy server resources are used while configuring the HTTP transport for business services. For more information, see [Using HTTP and Poller Transports](#).

- [Using Proxy Servers with SSL](#)
- [How to View Proxy Server Resources in Oracle Service Bus Console](#)
- [How to Create Proxy Server Resources](#)
- [How to Edit Proxy Server Resources](#)
- [How to Delete Proxy Server Resources](#)

Using Proxy Servers with SSL

When configuring a proxy server, you can specify the clear text or SSL port number for the server, in addition to the host name or IP address. Note that the port specified in the proxy server resource configuration corresponds to the actual physical port of the web proxy server and not the port of the back end service. The endpoint configuration on the business service, however, corresponds to the actual endpoint of the back end service. This is true even in the case of SSL/TLS Tunneling using the HTTP CONNECT command.

While some web proxy servers support HTTP CONNECT using both clear text and SSL sockets, other servers support it using only clear text sockets. For example, Oracle iPlanet Web Proxy Server 4.0 supports HTTP CONNECT using clear text and SSL sockets while Apache Web Server 2.2 in proxy server mode supports it using only clear sockets.

Typically, for SSL/TLS Tunneling support, you do not need to install server or Certificate Authority (CA) certificates on the physical proxy server. Instead, the web proxy server uses the certificate received from the caller (Service Bus, in this case) to open a secure socket. Similarly, when the server hosting the business application requires a client certificate, the certificate of the server hosting the proxy service (Service Bus) is used for authentication.

How to View Proxy Server Resources in Oracle Service Bus Console

The Folders Definition Editor for proxy servers lists all the proxy server resources you have created in the current session. Use this page to quickly find and access the proxy server resources you have defined.

To view proxy servers in the console:

1. Expand the **System** project, right-click **Proxy Servers**, and then select **Open**.
The Folder Definition Editor appears with a list of existing proxy server resources.
2. To locate specific proxy server resources, do the following:
 - a. If the query fields are not visible above the Proxy Servers table, click **Query by Example** in the table toolbar.
 - b. Enter the name of the proxy server resource you want to find above the **Name** column, and press **Enter**.
You can enter wildcard characters (? for a single character; * for multiple characters) to perform a more general search.
 - c. To view all proxy servers again, clear the query fields and press **Enter**.
3. To view the configuration for a proxy server, click the resource name in the Proxy Servers table.
4. To delete a proxy server resource, select the name of the resource in the table and click **Delete**. See [How to Delete Proxy Server Resources](#).

How to Create Proxy Server Resources

When you create a proxy server resource, you specify connection information for the server, including the server name and port number. Security information is only required if the server requires authentication.

You can configure multiple proxy servers for each proxy server resource. In this case, Service Bus can perform load balancing and offer fault tolerance features for the resource.

To create a proxy server resource:

1. Do one of the following:
 - If you are using JDeveloper, expand the Application Resources panel, right-click **Service Bus System Resources**, point to **New**, and then select **Proxy Server**.

Note:

To create proxy server resources directly in a project, making it a project-level resource, right-click the project, point to **New**, and then select **Proxy Server**.

- If you are using Oracle Service Bus Console, expand the **System** project, right-click **Proxy Servers**, point to **Create**, and then select **Create Proxy Server**.

The Create Proxy Server dialog appears.

2. Enter a name and optional description for the resource.
3. Click **Finish** or **Create**.
The Proxy Server Definition Editor appears.
4. To define proxy servers, do the following in the Host-Port Parameters table:
 - a. Click the **Add** icon above the table.
A new row appears in the table.
 - b. In the **Host** field, enter the host name or IP address of the proxy server.
 - c. In the **Port** field, enter the proxy server's port number.
 - d. To remove a host name and port number from the list, select the row in the table containing the server to delete and click the **Delete** icon.
5. If access to the proxy server requires authentication, enter a user name in the **User Name** field, and the associated password in the **Password** and **Confirm Password** fields.
6. Click **Save**.
7. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Edit Proxy Server Resources

Once you create a proxy server resource, you can modify its description and security options, and you can modify, add, and delete proxy servers from the host list.

To edit a proxy server resource:

1. Expand the project and folders containing the resource to edit. This can be any of the following locations:
 - In JDeveloper, the **Service Bus System Resources** folder in the Application Resources panel.
 - In JDeveloper, the Service Bus project or folder in which the proxy server resource is located in the Application Navigator.
 - In Oracle Service Bus Console, the **Proxy Servers** folder in the **System** project.
2. Right-click the proxy server name, and select **Open**.
The Proxy Server Definition Editor appears.
3. Modify any of the fields described in [How to Create Proxy Server Resources](#). The online help describes these fields in greater detail.
4. When you are done making changes, click **Save**.
5. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Delete Proxy Server Resources

When you delete a proxy server resource, any references to the resource from other Service Bus resources are broken. To find out whether any resources reference a proxy server, open the proxy server resource in the Proxy Server Definition Editor and click the **References** icon in the upper right. In JDeveloper, right-click the proxy server and select **Explore Dependencies**.

To delete a proxy server resource:

1. Expand the project and folders containing the resource to edit. This can be any of the following locations:
 - In JDeveloper, the **Service Bus System Resources** folder in the Application Resources panel.
 - In JDeveloper, the Service Bus project or folder in which the proxy server resource is located in the Application Navigator.
 - In Oracle Service Bus Console, the **Proxy Servers** folder in the **System** project.
2. Right-click the proxy server resource, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the resource. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Creating and Configuring Proxy Services

This chapter describes how to create, configure, and manage proxy services using the Oracle Service Bus Console and JDeveloper. Service Bus proxy services, along with business services, provide the means for managing services, transforming messages, and routing messages through the enterprise.

This chapter includes the following sections:

- [Introduction to Proxy Services](#)
- [Securing Proxy Services](#)
- [Service Level Agreement Alert Rules](#)
- [Web Services Interoperability Compliance](#)
- [Creating Proxy Services](#)
- [Configuring Proxy Services](#)
- [Deleting Proxy Services](#)
- [Consuming Proxy Services in JDeveloper with WSIL](#)

Introduction to Proxy Services

Proxy services provide the interface that service consumers use to connect with back-end services through Service Bus. They are definitions of intermediary web services that Service Bus hosts locally. Service Bus uses proxy services to route messages between business services (such as enterprise web services and databases) and service clients (such as presentation applications or other business services).

Proxy services define the interfaces in terms of Web Services Description Language (WSDL) or Web Application Definition Language (WADL) and the type of transport used. A proxy service defines the communication interface, the type of transport, transport settings, security settings, and the associated message processing logic. It then uses a message flow definition, or pipeline, to transform and route messages to one or more business services. The pipeline defines the logic that determines how messages are handled as they flow through Service Bus. If a proxy service is based on a WSDL document, the configuration also includes a WSDL port or a WSDL binding.

You can base proxy services on existing WSDL and WADL documents, including those imported from a UDDI registry, SOA Oracle Metadata Services (MDS) Repository, an application server, or the file system. Service Bus also supports proxy services that use the REST binding (see [Creating REST Services with Oracle Service Bus](#)). These proxy services are based on WADL documents and can only be created using the Service Bus Overview Editor in JDeveloper.

Proxy Service Definitions

Each proxy service is defined by whether it is based on a WSDL web service or a Service Bus transport. A WSDL-based service is a SOAP or XML proxy service whose interface is described by a WSDL document. A transport-typed service is a proxy service based on a Service Bus transport, including the JCA transport, which provides support for configuring proxy services for Oracle JCA-compliant adapters. It also includes REST proxy services, which use the HTTP transport. Each type of proxy service supports transport protocols specific to its definition. Service Bus supports several standard transport protocols as well as custom transports.

You can use either the Create Proxy Service wizard or the Service Bus Overview Editor in JDeveloper to create proxy services with either a WSDL-based or transport-typed service. Using the Service Bus Overview Editor, you can also generate proxy services directly from a JCA adapter to create a proxy service already configured for that adapter type. Both the wizard and the editor let you expose business services and pipelines as proxy services.

Service Types and Protocols for Proxy Services

Service Bus supports various service types ranging from conventional web services (using XML or SOAP bindings in WSDL files) to non-XML (generic) services. When you create a transport-typed proxy service, you also need to further define the service by specifying and configuring the service type. The service types you can select are restricted based on the transport used to communicate with the service endpoint. For information about the transports supported with each service type, see [Transports, Adapters, and Bindings](#).

A proxy service can have one of the following service types, identified by the types of messages it processes:

- **WSDL Based Service:** This service type is generated from an existing WSDL document or one that you create at the same time you create the proxy service. When creating a WSDL-based service, you need to specify the port or binding to use.
- **Messaging Service:** This service type can receive messages of one data type and respond with messages of a different data type. Supported data types include XML, Message Format Language (MFL), text, untyped, binary, Java, and attachments where the interface is not described by WSDL
- **Any SOAP Service:** This service type exchanges SOAP messages. SOAP messages are constructed by wrapping the contents of the header and body variables inside a `<soap:Envelope>` element. If the body variable contains a piece of reference XML, it is sent as is; that is, the referenced content is not substituted into the message. If attachments are defined in the attachments variable, a MIME package is created from the main message and the attachment data. Content handling for each attachment part is similar to how it is handled for messaging services.
- **Any XML Service (non-SOAP):** With this service type, messages to XML-based services are XML, but can be of any type the proxy service configuration allows. In messages that include attachments, their content is a MIME package that includes the primary XML payload as one of its parts (typically the first part or the one identified by the top-level content-type header).

- **REST Service:** This type of service is based on the REST binding, and can be generated from an existing WADL or one that you create at the same time you create the proxy service. This type of service can only be created in the Service Bus Overview Editor in JDeveloper, and is not an option on the Create Proxy Service wizard. For more information, see [Creating REST Services with Oracle Service Bus](#).

When to Use SOAP or Any XML Service Types

If you want to expose one port to clients for a variety of enterprise applications, use **Any SOAP** or **Any XML** service types. For Any SOAP, you must specify if it is SOAP 1.1 or SOAP 1.2.

When to Use the Messaging Service Type

If one of the request or response messages is non-XML, you must use the messaging service type. Service Bus does not automatically perform "misunderstand" SOAP header checking. However, you can use XQuery conditional expressions and validate actions to explicitly perform this type of check in the pipeline. For more information on the validate action, see [Adding Validate Actions in the Console](#). For more information on conditional XQuery expressions, see [Working With Expression Editors in Oracle Service Bus Console](#).

Binding Definitions and Runtime Variables for Proxy Service Types

No matter its definition or type, each proxy service type is modeled following the same pattern. Each service type must define these characteristics:

- Binding definition
- Runtime configuration
- Runtime variables (`$operation`, `$body`, `$header`, `$attachments`)

WSDL Service Type

Runtime Variables

- For SOAP-based WSDL services, the variables are similar to Any SOAP service types except `$operation` is initialized based on the operation selection algorithm.
- For XML-based WSDL services, the variables are similar to Any XML service types except the `$operation` is initialized based on the operation selection algorithm.

Messaging Service Type

Binding Definition

The binding definition for messaging services consists of configuring the content type of the messages that are exchanged. The content type for the response does not need to be the same as for the request; therefore, the response is configured separately. For example, the service could accept an MFL message and return an XML acknowledgment receipt. The response could also be set to None.

By definition, messaging-based services do not have a WSDL definition. It is not possible to request a WSDL document for those services. Following are the content types to choose from for the request and response:

- None

- Binary
- Text
- MFL
- XML
- Java

Note:

- If you are using a Reply action in the pipeline to propagate success or failure messages from a service to the calling client, select an option other than None. The None option blocks the reply.
 - Email, File, FTP, and SFTP transport proxy services with a messaging service type support one-way messaging *only*, so there is no response message. The content type for the response message should be None.
-

Runtime Variables

This service type is message-based. There is no concept of multiple operations as there is for web services. Therefore, the `$operation` variable is left empty. The `$body` variable holds the incoming message wrapped in a `<soap:Body>` element. For SOAP 1.1 proxy services, `$body` uses the SOAP 1.1 namespace `Body`; for SOAP 1.2 proxy services, it uses SOAP 1.2 namespace `Body`. The `$header` variable is not applicable to this service type, and is set to its default value. The `$attachments` variable contains message attachments if there are any.

To learn more about the message context variables, see [Message-Related Variables](#) and [Constructing Messages to Dispatch](#).

Any SOAP Service**Binding Definition**

The only information this service type defines is that the service is receiving or sending SOAP messages, regardless of their WSDL binding definition. The binding configuration for this type is empty, so the combination of this type and the content-type of the message is sufficient to determine whether or not there are attachments to the message. By definition, "any" services (SOAP or XML) do not have any WSDL definition. It is not possible to generate or view a WSDL document for those services.

Runtime Variables

The `$body` and `$header` variables respectively hold the `<soap:Body>` and `<soap:Header>` of the incoming SOAP message. (For SOAP 1.1 proxies, `$body` and `$header` use SOAP 1.1 namespace `Body` and namespace; for SOAP 1.2 proxies, they use SOAP 1.2 namespace `Body` and namespace.) The `$attachments` variable contains the SOAP message attachments, if any. The `$operation` variable is not applicable to this service type because you do not define a port type.

To learn more about the message context variables, see [Message-Related Variables](#) and [Constructing Messages to Dispatch](#).

Any XML Service

Binding Definition

The only information this service type defines is that the service is receiving or sending XML messages, regardless of their WSDL binding definition. The binding configuration for this type is empty, so the combination of this type and the content-type of the message is sufficient to determine whether or not there are attachments to the message.

As per their definition, "any" services (SOAP or XML) do not have a WSDL definition. It is not possible to request a WSDL document for those services.

Runtime Variables

The `$body` variable holds the incoming XML message wrapped in a `<soap:Body>` element. (For SOAP 1.1 proxies, `$body` and `$header` use SOAP 1.1 namespace `Body` and namespace; for SOAP 1.2 proxies, they use SOAP 1.2 namespace `Body` and namespace.) The `$attachments` variable contains message attachments, if there are any. The `$header` variable is not applicable to this service type and is set to its default value. The `$operation` variable is not applicable to this service type because you do not define a port type.

To learn more about the message context variables, see [Message-Related Variables](#) and [Constructing Messages to Dispatch](#).

Proxy Service Transport Protocol Configuration

Much of the configuration for proxy services involves the transport protocol. Transports are the communication layer between the external systems and the proxy services, acting as an entry point into Service Bus. The available transport protocols for a proxy service vary depending on the service type you are creating. Each transport protocol has its own configuration requirements. For more information about transport protocols and their configuration requirements, see [Working with JCA Adapters, Transports, and Bindings](#) and click the link to the specific protocol you are interested in.

Based on the transport and WSDL file, the transport mode is automatically selected, but you can overwrite it in the `$inbound` or `$outbound` variable.

Securing Proxy Services

You can secure proxy services through multiple methods, including Oracle Web Services Manager (WSM) policies, authentication mappings, and service key providers. A service provider is required if the proxy service routes messages to HTTPS services that require client certificate authentication and may be required in some message-level security scenarios. A service account can be created to provide authentication when connecting to a business service. It acts as an alias resource for the required user name and password pair. WebLogic Server can be used to directly manage security credentials for a business service requiring credential-level validation.

For more information about securing proxy services, see [Securing Business and Proxy Services](#).

Service Level Agreement Alert Rules

Service Level Agreement (SLA) alert rules define conditions under which an alert is generated. These conditions are typically indicators of the overall health of the Service Bus application or of a specific service component. For information about defining SLA alert rules for a proxy service, see "Creating Service Level Agreement Alert Rules" in *Administering Oracle Service Bus*.

Web Services Interoperability Compliance

You can configure a proxy service to enforce WS-I compliance (for SOAP 1.1 services only) and select the selection algorithm to use to determine the operation called by this proxy service. This option is only available for SOAP or XML services defined from a WSDL file. The WSDL specification defines a default algorithm to compute which operation is called based on the type of the SOAP message received. However, there are cases (for example, performance issues, signature/encryption issues, or the default algorithm is not applicable) when you may need to select the operation based on other means.

Service Bus provides additional algorithms. Each of them follows the same pattern and are based on the evaluation of an expression to get a value that is then used to lookup the corresponding operation in a static table.

Service Bus is generally very forgiving if an inbound message is either missing data such that the operation cannot be determined, or has data that does not correspond to a valid operation. Both of these conditions result in `$operation` being empty. Rather than reject all such messages, Service Bus does not initialize the operation variable in the context but otherwise continues to process the message. However, security requirements are enforced if the proxy service is WSDL-based and at least one of the following conditions is true:

- The WSDL file has a WS-Security policy and the proxy is an active intermediary.
- The proxy has message-level custom authentication (either custom token or user name/password).

If these conditions are met, then there is a runtime check to make sure the operation selection algorithm returns a valid operation name. If the operation selection returns null or an operation that is not in the WSDL file, then the message is rejected and an error is raised.

Creating Proxy Services

This section describes how to create proxy services using JDeveloper or the Oracle Service Bus Console. For information about creating Service Bus applications and projects, see [Creating Service Bus Applications and Projects in JDeveloper](#) or, for the console, [Create New Projects and Folders for Resources](#). For information about working with projects, applications, and other components in JDeveloper, see the *User's Guide for Oracle JDeveloper*

You can use various methods to create a proxy service, including the Create Proxy Service wizard, the Service Bus Overview Editor, generating it from an existing service or JCA resource, and exposing a pipeline as a proxy service when you create the pipeline. When you create a proxy service, the Create Proxy Service wizard provides a series of pages where you can configure certain proxy service properties. This section describes how to use the Create Proxy Service wizard to create proxy services. For

information on using the Service Bus Overview Editor, see [Developing Oracle Service Bus Applications in JDeveloper](#).

Before You Begin:

If you are using any system resources, such as SMTP servers, MQ connections, or UDDI servers, make sure to create those resources before beginning to create a proxy service. Configuring the proxy service includes specifying or selecting those resources, and you will not be able to complete the proxy service configuration until the required resources exist in Service Bus.

If you are working in JDeveloper, create or open the application and project to which you want to add the proxy service. If you are working in the Oracle Service Bus Console, make sure you are in an active session and the project to which you want to add the proxy service exists.

How to Create a Proxy Service

When you create a proxy service, you need to specify certain information, such as the service type and whether to use a WSDL file or Service Bus transport. The following topics provide additional information about creating proxy services:

- For information about proxy service types, see [Proxy Service Definitions](#)
- For information about WSDL files, see [Working with WSDL Documents](#).
- For information about the different transports you can use, see [Working with JCA Adapters, Transports, and Bindings](#).

To create a proxy service:

1. In the Project or Application Navigator, right-click the project, point to **New or Create**, and then select **Proxy Service**.

The Create Proxy Service wizard appears.

2. Enter a name for the service and, optionally, a description.

Note:

- JDeveloper lets you modify the location where the proxy service file is stored on the server. The file should be stored under the application and project folders, which is the default location.
 - For naming requirements, see [Naming Guidelines for Service Bus Components](#).
-
-

3. Do one of the following:

- To generate the proxy service by configuring its transport, select **Transport**, and then select the type of transport from the list.
- To generate the proxy service configuration from an existing WSDL file, select **WSDL** and then enter the name of the WSDL file or click the **Search** icon to search for a WSDL resource. Once you specify the WSDL file, select the port or binding to use from the **Port/Binding** field.

4. If you are working in JDeveloper, enter a new name for the pipeline generated for the proxy service (optional). If you do not want to generate a pipeline, clear the **Generate Pipeline** check box.

Note:

This option is disabled for the JEJB transport.

5. Click **Next**.

The fields on the remaining pages in the wizard depend on your selections from the first page. The options described in the following steps might not be available for all configurations.

6. If you chose WSDL definition on the first page, skip to [step 9](#).
7. If you chose a Transport definition on the first page, select one of the following service types:
 - **WSDL**: If you select this option, enter the WSDL file name or click **Choose a WSDL** to browse to and select a WSDL file to use. Select the port or binding type from the list of options.
 - **Any SOAP**: If you select this option, select the SOAP version to use.
 - **Any XML**: This option requires no additional configuration.
 - **Messaging**: If you select this option, select the data type for the request message and optionally the response message. If you select MFL or XML, you must also select the schema file.

8. Click **Next**.

The Transport page appears.

9. If the protocol you want to use is not already selected, select a new protocol from the list.
10. Specify the endpoint URI. For required URI formats, see the online help provided with Service Bus.
11. Click **Finish** or **Create**.

The Proxy Service Definition Editor displays the general configuration of the new proxy service.

12. Configure the proxy service, as described in [Configuring Proxy Services](#).
13. After you create a proxy service, you need to configure the message flow in a pipeline or split-join. To learn more, see [Working with Pipelines in Oracle Service Bus Console](#) and [Improving Service Performance with Split-Join](#).

How to Generate a Proxy Service from a JCA Binding Resource

With Service Bus, you can generate proxy services from inbound JCA files. JCA services, which use the Service Bus JCA transport, communicate with Enterprise Information Systems (EIS) through a JCA adapter framework and JCA-compliant

adapters. For more information on JCA binding resources, see [Using the JCA Transport and JCA Adapters](#).

Before You Begin:

Create the JCA file, its associated abstract WSDL file, and any other required resources, such as a TopLink mapping file in JDeveloper. For more information, see [Using the JCA Transport and JCA Adapters](#) and *Understanding Technology Adapters*.

Note:

- If you select an outbound JCA binding instead of an inbound one, the option to generate a proxy service is not available.
 - In JDeveloper, you can also generate a proxy service when you create the JCA adapter if you create it from the Service Bus Overview Editor. For more information, see [How to Create a Proxy Service](#).
-
-

Generating a Proxy Service from a JCA Binding in JDeveloper

To generate a proxy service from a JCA binding in JDeveloper:

1. In the Application Navigator, right-click the inbound JCA file, point to **Service Bus**, and then select **Generate Proxy Service**.

The Create Proxy Service wizard appears, configured for the selected JCA binding.

2. On the wizard, keep the default service name and location, or specify new ones. The location must be in the current application's directory structure.

See [Naming Guidelines for Service Bus Components](#) for naming guidance.

3. To generate a corresponding pipeline for the proxy service, leave **Generate Pipeline** selected and specify a name for the pipeline (or accept the default name). Clear the **Generate Pipeline** check box if you do not want to create the pipeline at this time.

4. Click **Next**.

The Type page appears.

5. Select a WSDL binding if necessary, and then click **Next** again.

The Transport page appears.

6. Update the endpoint URI if necessary, and then click **Finish**.

Service Bus generates the proxy service and the concrete WSDL file used by the proxy service.

7. Configure the proxy service, as described in [Configuring Proxy Services](#).

Generating a Proxy Service from a JCA Binding in the Console

Before you begin, import the JCA resource files from JDeveloper to the console so all references to dependencies are maintained. For more information, see [Working with JCA Binding Resources](#). and [Importing and Exporting Resources and Configurations](#).

To generate a proxy service from a JCA binding in the console:

1. In the Project Navigator, right-click the inbound JCA file, and select **Generate WSDL and Service**.

The Generate WSDL and Service dialog appears.

2. Optionally, modify the names of the WSDL file and the service you want to generate, and select a location for these new resources.

See [Naming Guidelines for Service Bus Components](#) for naming guidance.

3. Click **Generate**.

Service Bus generates the service and its corresponding WSDL file.

4. In the Project Navigator, navigate to the new resources, and open the proxy service in the Proxy Service Definition Editor.

5. Configure the proxy service, as described in [Configuring Proxy Services](#).

How to Generate a Proxy Service from an Existing Service in JDeveloper

In JDeveloper, you can generate proxy services from other proxy services, business services, pipelines, and split-joins. The configuration of the proxy service is based on that of the existing service.

To generate a proxy service from another service in JDeveloper:

1. In the Application Navigator, right-click the existing service, point to **Service Bus**, and then select **Generate Proxy Service**.

The Create Proxy Service wizard appears.

2. Configure the name, description, and file location for the service, and then click **Next**.

The Type page appears.

3. If the service is a WSDL service, select the binding to use and then click **Next**.
4. On the Transport page, select the transport protocol and update the endpoint URI. For required URI formats, see the online help provided with Service Bus.
5. Click **Finish**.
6. Configure the proxy service, as described in [Configuring Proxy Services](#).

How to Generate a Proxy Service from a WSDL Document in JDeveloper

You can use an existing WSDL document to generate a proxy service, business service, pipeline, or split-join.

To generate a proxy service from a WSDL document in JDeveloper:

1. In the Application Navigator, right-click the existing WSDL document, point to **Service Bus**, and then select **Generate Proxy Service**.

The Create Proxy Service wizard appears.

2. Configure the name, description, file location, and WSDL binding for the service.
3. To generate a corresponding pipeline for the proxy service, leave **Generate Pipeline** selected and specify a name for the pipeline (or accept the default name). Clear the **Generate Pipeline** check box if you do not want to create the pipeline at this time.
4. Click **Next**.
5. On the Transport page, select the transport protocol and update the endpoint URI. For required URI formats, see the online help provided with Service Bus.
6. Click **Finish**.
7. Configure the proxy service, as described in [Configuring Proxy Services](#).

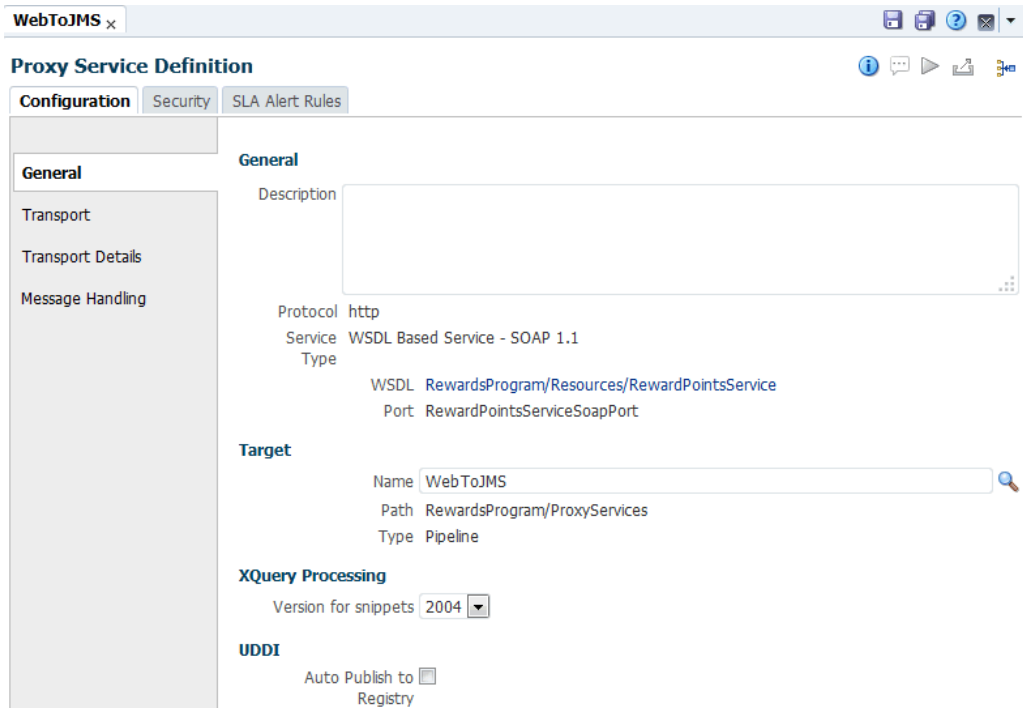
Configuring Proxy Services

Once you create a proxy service, you can edit the configuration, add security policies, modify security settings, and set up SLA alert rules. The information you can modify depends on how the service was originally configured. For a list of all the configurable properties for proxy services, see the online help available for each Proxy Service Definition Editor page.

If you are working in the Oracle Service Bus Console, make sure you are in an active session before performing any of the tasks in this section.

How to Configure General Information for a Proxy Service

The General tab of the Proxy Service Definition Editor displays information about the service such as a description of the service, the transport used by the service, the service type, any WSDL ports or bindings, the XQuery version, and the service invoked by the proxy service. The following figure shows the General tab in the Oracle Service Bus Console.

Figure 8-1 Proxy Service General Configuration Page in the Console

To configure general information for a proxy service:

1. In the Project or Application Navigator, right-click the proxy service to edit, and click **Open**.
2. Click the General tab if it is not already the visible page.
3. Enter or update the description for the service.
4. If the service references any resources, such as a WSDL or MFL document, click the name of the resource to view the document in its own editor.
5. To change the target service for the proxy service, click **Choose a Service Resource** next to the Target Service name. In JDeveloper, browse to and select a new target service. In the console, search for and select a new target resource.
6. To change the XQuery version used, select a new option from the **XQuery Processing** list.
7. If you are working in the Oracle Service Bus Console, select **Auto Publish to Registry** if you want the proxy service to automatically be published to the default UDDI registry,

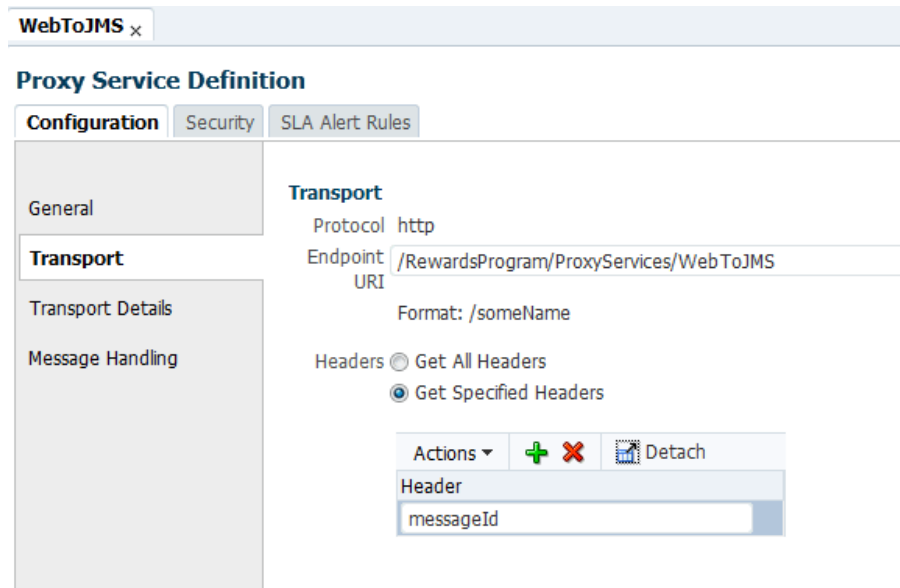
In order to automatically publish proxy services, you must define a default UDDI server. For more information, see [Keeping Services Synchronized](#).

8. When you are done making changes, click **Save All**.
9. If you are working in the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Configure a Proxy Service Transport

Use the Transport and Transport detail page to configure the transport for the proxy service. The available properties vary for each transport. The following figure shows the Transport tab in the Oracle Service Bus Console.

Figure 8-2 Proxy Service Transport Configuration Page in the Console

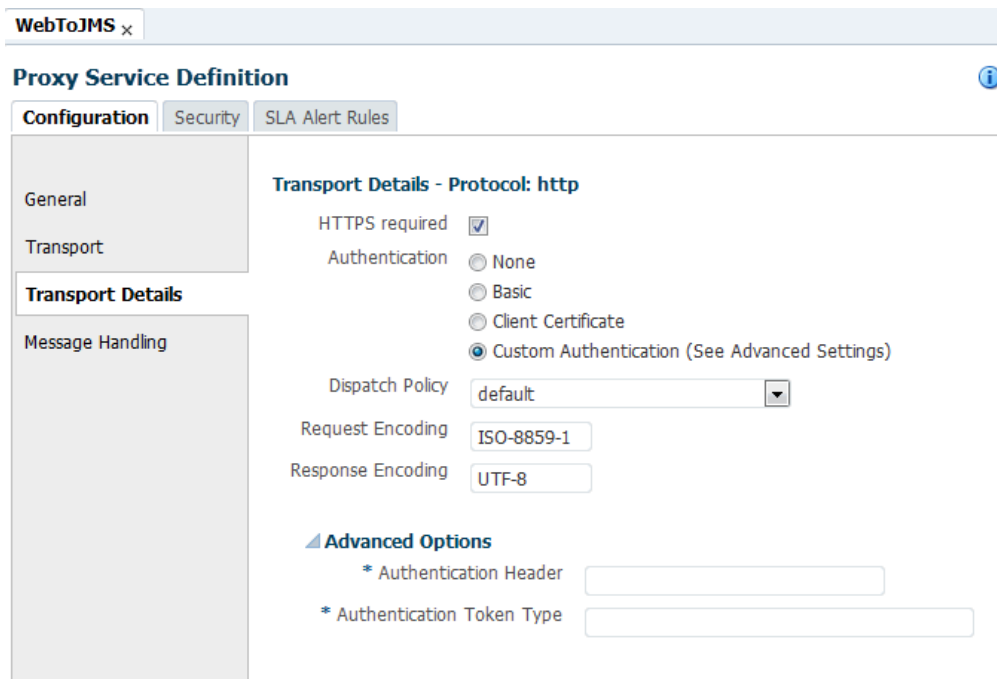


To configure a proxy service transport:

1. In the Project or Application Navigator, right-click the proxy service to edit, and click **Open**.
2. Click the Transport tab, and do any of the following:
 - Update the endpoint URI.
 - To retrieve all headers, select **Get All Headers**.
 - To retrieve a subset of headers, select **Get Specified Headers**, and specify the headers to retrieve in the Headers table. Click **Add** above the table to add headers.
3. Click the Transport Detail tab.

The properties you can configure here are based on the transport for the proxy service. For information about specific transports, see [Working with JCA Adapters, Transports, and Bindings](#). The following figure shows the HTTP transport properties in the console.

Figure 8-3 Proxy Service Transport Details Page in the Console

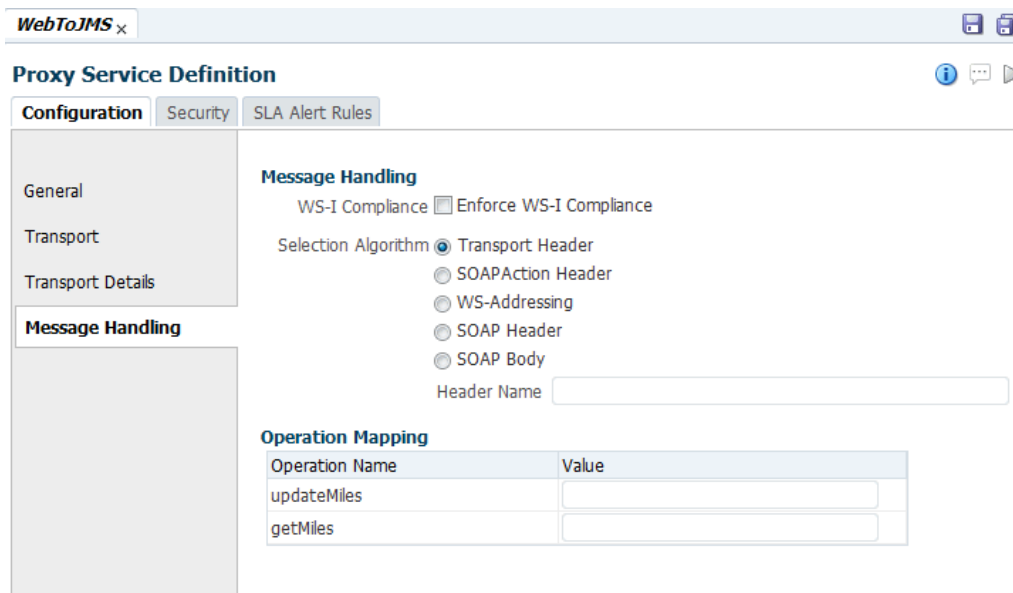


4. When you are done making changes, click **Save All**.
5. If you are working in the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Configure Proxy Service Message Handling

On the Message Handling page, you can configure how the proxy service processes message contents, including checking for WS-I compliance and the XQuery version to use. The following figure shows the Message Handling tab in the Oracle Service Bus Console.

Figure 8-4 Proxy Service Message Handling Page in the Console



To configure message handling for a proxy service:

1. In the Project or Application Navigator, right-click the proxy service to edit, and click **Open**.
2. Click the **Message Handling** tab.
3. To check messages for WS-I compliance, select the check box for **Enforce WS-I Compliance**.
4. Select one of the following selection algorithms:
 - **Transport Header:** Lets you define the transport header that contains the lookup value. If you select this option, you must also specify a header name and the operational values in the new fields that appear.
 - **SOAP Action Header:** Specifies that operation mapping be done automatically from the WSDL file associated with this proxy service.
 - **WS-Addressing:** Specifies that the lookup value is contained by the WS-Addressing Action tag located in the SOAP headers of the SOAP message. If you select this option, you must also specify the operational values in the new fields that appear.
 - **SOAP Header:** Lets you define an XPath expression to be evaluated against the SOAP headers. This allows you to get the lookup value. If you select this option, you must also define an XPath expression and value in the new fields that appear.
 - **SOAP Body Type:** Uses the default algorithm defined by the WSDL specification to compute which operation is called based on the type of the SOAP message received.
 - **Payload Type:** This option is only available for XML services based on a WSDL port or WSDL binding.

For more information about these algorithms, see the online help provided for this page.
5. When you are done making changes, click **Save All**.
6. If you are working in the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Configure Security for a Proxy Service

You can secure proxy services through multiple methods, including Oracle Web Services Manager (OWSM) policies and access control at the transport and message levels. For more information about securing proxy services, see [Securing Proxy Services](#) and [Securing Business and Proxy Services](#).

How to Configure Service Level Agreement Alerts for a Proxy Service

SLA alerts let system administrators know when certain conditions are met that indicate the health of a proxy service. For information about defining SLA alerts, see "Creating Service Level Agreement Alert Rules" in *Administering Oracle Service Bus*.

Deleting Proxy Services

Deleting a proxy service deletes all of the ACLs referenced by the proxy from the repository controlled by Service Bus, as well as from the appropriate authorization provider. You can delete the proxy service even if it is referenced by other resources, though this might result in conflicts due to unresolved references to the deleted service.

How to Delete a Proxy Service

Before deleting a proxy service, check for any dependencies. In the Oracle Service Bus Console, open the proxy service in the Proxy Service Definition Editor and click the **References** icon in the upper right to find out whether any services are using it. In JDeveloper, right-click the proxy service and select **Explore Dependencies**.

To delete a proxy service:

1. In the Project or Application Navigator, right-click the proxy service to delete, and select **Delete**.

A confirmation dialog appears.

2. In JDeveloper, if other resources reference this proxy service the confirmation dialog displays the number of references. Click **Show Usages** to view information about the reference.
3. On the confirmation dialog, click **Yes** to confirm you want to delete the service.

The proxy service is deleted.

4. If you are working in the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Consuming Proxy Services in JDeveloper with WSIL

Service Bus makes its WSDL-based proxy services available through the Web Services Inspection Language (WSIL), letting you consume Service Bus WSDL proxy services in JDeveloper for service orchestration in Oracle SOA Suite.

The Service Bus WSIL servlet automatically registers WSDL-based proxy services deployed in the Service Bus runtime environment. By creating a WSIL connection in JDeveloper, you can access those proxy services through different URL patterns that map to different hierarchy levels, such as project, folder, and individual service. For example, when you connect to the Service Bus WSIL servlet with a project-level URL, you can see all the child folders and WSDL-based proxy services in that project in JDeveloper.

How to Consume Service Bus Proxy Services in JDeveloper with WSIL

The following procedure guides you through the process of creating a WSIL connection in JDeveloper and generating web service references out of Service Bus WSDL proxy services for use in SOA applications.

To consume proxy services in JDeveloper with WSIL:

1. In JDeveloper, open or create a SOA application.

2. Create a new WSIL connection.

In the Resources window, click the **Add** icon, select **IDE Connections**, and select **WSIL**.

On the Create WSIL Connection dialog, do the following:

a. Enter a name for the connection.

See [Naming Guidelines for Service Bus Components](#) for naming guidance.

b. Enter the credentials for one of the following Service Bus roles: Administrator, Deployer, Operator, or Monitor.

c. Enter the URL to the Service Bus WSIL in one of the following formats:

- Domain (gets all projects, folders, and WSDL-based proxy services):

```
http://host:port/sbinspection.wsil
```

- Project (gets all child folders and WSDL-based proxy services):

```
http://host:port/sbinspection.wsil?refpath=project_name
```

- Folder (in a project, gets the folder, all child folders, and WSDL-based proxy services):

```
http://host:port/sbinspection.wsil?refpath=project_name/folder_path
```

For example:

```
http://localhost:7021/sbinspection.wsil?refpath=MortgageBroker/ProxyServices
```

- Proxy Service (gets an individual WSDL-based proxy service):

```
http://host:port/sbinspection.wsil?refpath=project_name/folder_path/wsdL_proxy_service
```

For example:

```
http://localhost:7021/sbinspection.wsil?refpath=MortgageBroker/ProxyServices/loanGateway1
```

In a cluster, the WSIL servlet is deployed on Managed Servers and not the Admin Server. Use a Managed Server host name and port in the URL.

d. Click **Test Connection** to verify the connection is valid.

e. Click **OK**. The WSIL connection appears in the Resources window in the hierarchy determined by the URL you entered.

3. To use a Service Bus WSDL-based proxy service in your SOA application, create a web service reference to it.

- In the Components window, create a new web service. In the Create Web Service window, click the **WSDL URL** browse icon.
- In the SOA Resource Browser, select **Resources**, and select the Service Bus proxy service in the WSIL connection created in the previous step.

When you create the web service reference to a Service Bus WSDL-based proxy service, you can use it as an external reference in your SOA application.

The Service Bus WSIL servlet leverages the SBResource servlet. If the SBResource is undeployed, the WSIL connection is not available.

Creating and Configuring Business Services

This chapter describes how to create, configure, and manage business services using the Oracle Service Bus Console and JDeveloper. Service Bus business services, along with proxy services, provide the means for managing services, transforming messages, and routing messages through the enterprise.

- [Introduction to Business Services](#)
- [Using Proxy Servers](#)
- [Service Level Agreement Alert Rules](#)
- [Security and Security Policies for Business Services](#)
- [Creating Business Services](#)
- [Configuring Business Services](#)
- [Deleting Business Services](#)
- [Improving Performance by Caching Business Service Results](#)

Introduction to Business Services

Business services are Service Bus definitions of the enterprise services with which you want to exchange messages. They define enterprise web services to which Service Bus is a client. Those external web services are implemented in and hosted by external systems, so Service Bus must know what to invoke, how to invoke it, and what to expect as a result. Business services model those interfaces so that Service Bus can invoke the external services.

You define business services using WSDL (Web Services Definition Language) or Web Application Definition Language (WADL), just as you would define a proxy service. A business service configuration includes its interface, transport settings, and security settings. If the business service is based on a WSDL document, the configuration also includes a WSDL port or a WSDL binding. (See [Working with WSDL Documents](#).)

You can base business services on existing WSDL and WADL documents, including those imported from a UDDI registry, SOA Oracle Metadata Services (MDS) Repository, an application server, or the file system. Service Bus also supports business services that use the REST binding (see [Creating REST Services with Oracle Service Bus](#)). These services are based on WADL documents and can only be created using the Service Bus Overview Editor in JDeveloper.

Business Service Definitions

Each business service is defined by whether it is based on a WSDL web service or a Service Bus transport. A WSDL-based service is a SOAP or XML business service whose interface is described by a WSDL document. A transport-typed service is a business service based on a Service Bus transport, including the JCA transport, which provides support for configuring business services for Oracle JCA-compliant adapters. It also includes REST business services, which use the HTTP transport. Each type of business service supports transport protocols specific to its definition. Service Bus supports several standard transport protocols as well as custom transports.

You can use either the Create Business Service wizard or the Service Bus Overview Editor in JDeveloper to create business services with either a WSDL-based or transport-typed service. Using the Service Bus Overview Editor, you can also generate business services directly from a JCA adapter to create a business service already configured for that adapter type. Both the wizard and the editor let you generate business services from proxy services.

Service Types and Protocols for Business Services

Service Bus supports various service types ranging from conventional web services (using XML or SOAP bindings in WSDL files) to non-XML (generic) services. When you create a transport-typed business service, you also need to further define the service by specifying and configuring the service type. The service types you can select are restricted based on the transport used to communicate with the service endpoint. For information about the transports supported with each service type, see [Transports, Adapters, and Bindings](#).

A business service can have one of the following service types, identified by the types of messages it processes:

- **WSDL Based Service:** This service type is generated from an existing WSDL document or one that you create at the same time you create the business service. When creating a WSDL-based service, you need to specify the port or binding to use.
- **Messaging Service:** This service type can receive messages of one data type and respond with messages of a different data type. Supported data types include XML, Message Format Language (MFL), text, untyped, binary, Java, and attachments where the interface is not described by WSDL
- **Any SOAP Service:** This service type exchanges SOAP messages. SOAP messages are constructed by wrapping the contents of the header and body variables inside a `<soap:Envelope>` element. If the body variable contains a piece of reference XML, it is sent as is; that is, the referenced content is not substituted into the message. If attachments are defined in the attachments variable, a MIME package is created from the main message and the attachment data. Content handling for each attachment part is similar to how it is handled for messaging services.
- **Any XML Service (non-SOAP):** With this service type, messages to XML-based services are XML, but can be of any type the business service configuration allows. In messages that include attachments, their content is a MIME package that includes the primary XML payload as one of its parts (typically the first part or the one identified by the top-level content-type header).

- **REST Service:** This type of service is based on the REST binding, and can be generated from an existing WADL or one that you create at the same time you create the business service. This type of service can only be created in the Service Bus Overview Editor in JDeveloper, and is not an option on the Create Business Service wizard. For more information, see [Creating REST Services with Oracle Service Bus](#).

Binding Definitions and Runtime Variables for Business Service Types

Each business service type is modeled following the same pattern, which must be configured for the service. These models are the same as those for proxy services. For more information, see [Binding Definitions and Runtime Variables for Proxy Service Types](#).

Business Service Transport Protocol Configuration

Much of the configuration for business services involves the transport protocol. Transports are the communication layer between the external systems and the business services. The available transport protocols for a business service vary depending on the service type you are creating. Each transport protocol has its own configuration requirements. For more information about transport protocols and their configuration requirements, see [Working with JCA Adapters, Transports, and Bindings](#) and click the link to the specific protocol you are interested in.

Based on the transport and WSDL file or interface, the transport mode is automatically selected, but you can overwrite it using the routing options action for a route or publish action.

You can configure the following parameters for each business service:

- List of weighted endpoint URIs in the format `<string URI, integer weight>`; for example, `<http://www.oracle.com, 100>`. For a random-weighted list, the list should contain at least one element.
- Load-balancing algorithm, which can be round-robin, random, or random-weighted. If you select random-weighted, the weights are applicable for each URI.
- Retry Count
- Retry Iteration Interval
- Retry Application Errors

The transport you select must be able to support the transport mode (that is, request/response, one-way or both) required by the binding definition, and be configured accordingly.

For services exchanging messages in both modes (request/response and one-way), you must configure the binding layer so it can select the transport mode accordingly. This occurs automatically when the service is a concrete type, as it is described in the binding definition. When it is not a concrete type, to configure the binding layer, you must use the routing options action in the pipeline to set the mode for a route or publish.

For a Tuxedo transport-based service, if the service type is XML, an FML32 buffer with an FLD_MBSTRING field from a Tuxedo client will not be transformed to XML. For information about configuring business services based on various transport protocols, see [Working with JCA Adapters, Transports, and Bindings](#).

About the Load Balancing Algorithm

The load balancing algorithm defines the order in which the endpoint URIs are selected at runtime. Service Bus supports the following algorithms:

- **Round-robin:** Dynamically orders the URIs that you define for a business service. If the first one fails, it tries the next one, and so on until the retry count is exhausted. For every new message, there is a new order of URIs.
- **Random:** Randomly orders the list of URIs that you define for a business service. If the first one fails, it tries the next one, and so on until the retry count is exhausted.
- **Random-weighted:** Randomly orders the list of URIs that you define for a business service, but some are retried more than others based on the value you enter in the **Weight** field.
- **None:** Orders the list of URIs that you define for a business service from top to bottom.

About Business Service URI Retries

The retry option for business services specifies the maximum number of times a business service can attempt to access endpoint URIs after an initial failure. For example, consider the behavior of a business service B with endpoint URIs eu1, eu2, and eu3, when the retry count is set to 1, 2, and 4.

When Retry Count = 1: If business service B fails to process a request or is unable to access the endpoint URI eu1, it tries to process the request with eu2 (retry 1). If the retry fails then the business service returns failure. The business service does not retry the third endpoint URI eu3.

When Retry Count = 2: If business service B fails to process a request or is unable to access the endpoint URI eu1, it tries to process the request with eu2 (retry 1). If the retry fails then the business service tries to process the request with eu3 (retry 2). If the retry fails then the business service returns failure.

When Retry Count = 4: If business service B fails to process a request or is unable to access the endpoint URI eu1, it tries to process the request with eu2 (retry 1). If the retry fails then the business service tries to process the request with eu3 (retry 2). Then the business service waits for a interval you have configured for retry iteration interval (in seconds) before trying eu1 (retry 3). If this fails the business service retries eu2 (retry 4). If the retry fails then the business service returns failure.

If the retry count is set to 0, then the business service does not retry after the failure.

Note:

The order in which a business service retries the endpoints is controlled by the load balancing algorithm.

Suppressing Retries in Case of Application Errors

A business service might fail to process a request due to communication or application errors. Communication errors occur due to random network problems. Retrying such requests with another endpoint URI can be successful. Application errors occur when a request is malformed or other errors, and cannot be processed by any of the endpoints. You can turn off retry behavior for the application errors by clearing the

Retry Application Errors option in the Transport Configuration page for a business service, depending on the transport used.

Message Handling for Business Services

Business services include properties that define how the service processes message contents, including MTOM/XOP support, MIME attachments, checking for WS-I compliance, and the XQuery version to use.

XOP/MTOM Support

A business services enabled for XOP/MTOM support can encode outbound messages in MTOM/XOP format. SOAP Message Transmission Optimization Mechanism (MTOM) is a method of sending binary data to and from web services. MTOM uses XML-binary Optimized Packaging (XOP) to transfer the binary data.

Service Bus supports XOP/MTOM using the following transports:

- HTTP/S
- Local
- SB

Binary data in the `$header` and `$body` message context variables can be handled in either of two ways:

- **Include Binary Data by Reference:** (Default) In an outbound response message, replace `xop:Include` elements with `ctx:binary-content` elements when setting up the `$body` message context variable.
- **Include Binary Data by Value:** In an outbound response message, replace `xop:Include` elements with Base64-encoded text versions of corresponding binary data when setting up the `$body` message context variable.

Note that if XOP/MTOM support is enabled for a business service, it is not required that every outbound message be in the MTOM format. Instead, this setting specifies that the business service is capable of handling an MTOM payload. Since Service Bus does not support a combination of MTOM and SwA, the system issues a runtime error when Service Bus attempts to dispatch an outbound request to a business service and the business service is both enabled for MTOM/XOP and the `$attachments` message context variable is not null.

Attachments

Service Bus supports streaming MIME attachments using the HTTP/S transport. This feature lets you store attachments in outbound response messages to a disk file and then process the data in a streaming fashion without buffering the attachment contents in memory. This enables the business service to process large attachments robustly and efficiently.

Note that if you enable XOP/MTOM support and choose the **Include Binary Data by Value** option, a warning appears if you try to select **Page Attachments to Disk**. These two options are not compatible. Note also that payloads that contain attachments must conform to RFC 822. Specifically, lines containing internet headers need to be terminated with CRLF (carriage return line feed).

Web Services Interoperability Compliance

In a business service's message handling properties, you can specify whether the service must conform to the Basic Profile defined by the Web Services Interoperability Organization (WS-I). This option is available for or SOAP 1.1 services only. When a service is marked WS-I compliant, checks are performed against the messages sent to and from that service.

Using Proxy Servers

You can configure business services to route messages through a proxy server by creating a proxy server resource that specifies one or more proxy servers together with the necessary credentials. You can then associate the proxy server resource with a business service. This instructs Service Bus to connect to the business service through the configured proxy server.

Adding multiple proxy servers to a resource enables Service Bus to perform load balancing and offer fault tolerance among the configured proxy servers. The credentials are used when opening a connection to the proxy server. If a particular proxy server is not reachable, Service Bus attempts to use the next proxy server in the configuration. If all proxy servers are unreachable, Service Bus tries to connect to the back end service directly. If that too fails, a fault is raised and sent back to the caller.

For information about proxy server resources, see [Working with Proxy Server Resources](#).

Service Level Agreement Alert Rules

Service Level Agreement (SLA) alert rules define conditions under which an alert is generated. These conditions are typically indicators of the overall health of the Service Bus application or of a specific service component. For information about defining SLA alert rules for a business service, see "Creating Service Level Agreement Alert Rules" in *Administering Oracle Service Bus*.

Security and Security Policies for Business Services

You can secure business services through multiple methods, including Oracle Web Services Manager (OWSM) policies and access control at the transport level. Outbound transport-level security applies to the connections between proxy services and business services. OWSM policies are bound by reference, not inlined in the effective WSDL file. When you secure business services with OWSM, you can also specify policy overrides.

For more information about transport-level security, see [Configuring Transport-Level Security](#). For more information about securing business services, see [Securing Business and Proxy Services](#).

Creating Business Services

This section describes how to create business services using Oracle JDeveloper or the Oracle Service Bus Console. For information about creating Service Bus applications and projects, see [Creating Service Bus Applications and Projects in JDeveloper](#) or [Create New Projects and Folders for Resources](#). For information about working with projects, applications, and other components in JDeveloper, see *Developing Applications with Oracle JDeveloper*.

You can use various methods to create a business service, including generating it from an existing service, a JCA resource, or WSDL document. When you create a business service, the Create Business Service wizard provides a series of pages where you can configure certain business service properties. This section describes how to use the Create Business Service wizard to create business services. For information on using the Service Bus Overview Editor, see [Developing Oracle Service Bus Applications in JDeveloper](#).

Before You Begin:

If you are using any system resources, such as SMTP servers, MQ connections, or UDDI servers, make sure to create those resources before beginning to create a business service. Configuring the business service includes specifying or selecting those resources, and you will not be able to complete the business service configuration until the required resources exist in Service Bus.

If you are working in JDeveloper, create or open the application and project to which you want to add the business service. If you are working in the Oracle Service Bus Console, make sure you are in an active session and the project to which you want to add the business service exists.

How to Create a Business Service

When you create a business service, you need to specify certain information, such as the service type and whether to use a WSDL file or Service Bus transport. The following topics provide additional information about creating business services:

- For information about business service types, see [Business Service Definitions](#).
- For information about WSDL files, see [Working with WSDL Documents](#).
- For information about the different transports you can use, see [Working with JCA Adapters, Transports, and Bindings](#).
- For information about load balancing, see [About the Load Balancing Algorithm](#).

To create a business service:

1. In the Project or Application Navigator, right-click the project, point to **New** or **Create**, and then select **Business Service**.

The Create Business Service wizard appears.

2. Enter a name for the service and, optionally, a description.

Note:

- JDeveloper lets you modify the location where the business service file is stored on the server. The file should be stored under the application and project folders, which is the default location.
 - For naming requirements, see [Naming Guidelines for Service Bus Components](#).
-
-

3. Do one of the following:

- To generate the business service by configuring its transport, select **Transport**, and then select the type of transport from the list.

- To generate the business service configuration from an existing WSDL file, select **WSDL** and then enter the name of the WSDL file or click the **Search** icon to search for a WSDL resource. Once you specify the WSDL file, select the port or binding to use from the **Port/Binding** field.

4. Click **Next**.

The fields on the remaining pages in the wizard depend on your selections from the first page. The options described in the following steps might not be available for all configurations.

5. If you chose WSDL definition on the first page, skip to step 8.
6. If you chose a Transport definition on the first page, select one of the following service types:
- **WSDL**: If you select this option, enter the WSDL file name or click **Choose a WSDL** to browse to and select a WSDL file to use. Select the port or binding type from the list of options.
 - **Any SOAP**: If you select this option, select the SOAP version to use.
 - **Any XML**: This option requires no additional configuration.
 - **Messaging**: If you select this option, select the data type for the request message and optionally the response message. If you select MFL or XML, you must also select the schema file.

7. Click **Next**.

The Transport page appears.

8. If the protocol you want to use is not already selected, select a new protocol from the list.
9. If you are working in the Oracle Service Bus Console, select a load balancing algorithm for the business service.
10. Specify the endpoint URIs. For required URI formats, see the online help provided with Service Bus.

Note:

In the console, you can add multiple URIs to the list. Click **Add** and then modify the new URI that appears. Use the up and down arrows to re-order the URIs. In JDeveloper, you can do this when you configure the business service.

11. Click **Finish** or **Create**.

The Business Service Definition Editor displays the general configuration of the new business service.

12. Configure the business service, as described in [Configuring Business Services](#).

How to Generate a Business Service from a JCA Binding Resource

With Service Bus, you can generate a business service from an outbound JCA binding resource. JCA services, which use the Service Bus JCA transport, communicate with

Enterprise Information Systems (EIS) through a JCA adapter framework and JCA-compliant adapters. For more information on JCA binding resources, see [Using the JCA Transport and JCA Adapters](#).

Before You Begin:

Create the JCA file, its associated abstract WSDL file, and any other required resources, such as a TopLink mapping file in JDeveloper. For more information, see [Using the JCA Transport and JCA Adapters](#) and *Understanding Technology Adapters*.

Note:

- If you select an inbound JCA binding instead of an outbound one, the option to generate a business service is not available.
 - In JDeveloper, you can also generate a business service when you create the JCA adapter if you create it from the Service Bus Overview Editor. For more information, see [How to Create a Business Service](#).
-
-

Generating a Business Service from a JCA Binding in JDeveloper

To generate a business service from a JCA binding in JDeveloper:

1. In the Application Navigator, right-click the outbound JCA file, point to **Service Bus**, and then select **Generate Business Service**.

The Create Business Service wizard appears, configured for the selected JCA binding.

2. On the wizard, keep the default service name and location, or specify new ones. The location must be in the current application's directory structure.

See [Naming Guidelines for Service Bus Components](#) for naming guidance.

3. Click **Next**.

The Type page appears.

4. Select a WSDL binding if necessary, and then click **Next** again.

The Transport page appears.

5. Update the endpoint URI if necessary, and then click **Finish**.

Service Bus generates the business service and the concrete WSDL file that is used by the business service.

6. Configure the business service, as described in [Configuring Business Services](#).

Generating a Business Service from a JCA Binding in the Console

Before you begin, import the JCA resource files from JDeveloper to the console so all references to dependencies are maintained. For more information, see [Working with JCA Binding Resources](#). and [Importing and Exporting Resources and Configurations](#).

To generate a business service from a JCA binding in the console:

1. In the Project Navigator, right-click the outbound JCA file, and select **Generate WSDL and Service**.

The Generate WSDL and Service dialog appears.

2. Optionally, modify the names of the WSDL file and the service you want to generate, and select a location for these new resources.

See [Naming Guidelines for Service Bus Components](#) for naming guidance.

3. Click **Generate**.

Service Bus generates the service and its corresponding WSDL file.

4. In the Project Navigator, navigate to the new resources, and open the business service in the Business Service Definition Editor.

5. Configure the business service, as described in [Configuring Business Services](#).

How to Generate a Business Service from a Proxy Service in JDeveloper

In JDeveloper, you can generate business services from the proxy services you create. The configuration of the business service is based on that of the proxy service.

To generate a business service from a proxy service in JDeveloper:

1. In the Application Navigator, right-click the existing proxy service, point to **Service Bus**, and then select **Generate Business Service**.

The Create Business Service wizard appears.

2. Configure the name, description, and file location for the service, and then click **Next**.

The Type page appears.

3. If the service is a WSDL service, select the binding to use and then click **Next**.
4. On the Transport page, select the transport protocol and update the endpoint URI. For required URI formats, see the online help provided with Service Bus.
5. Click **Finish**.
6. Configure the business service, as described in [Configuring Business Services](#).

How to Generate a Business Service from a WSDL Document in JDeveloper

You can use an existing WSDL document to generate a proxy service, business service, pipeline, or split-join.

To generate a business service from a WSDL document in JDeveloper:

1. In the Application Navigator, right-click the existing WSDL document, point to **Service Bus**, and then select **Generate Business Service**.

The Create Business Service wizard appears.

2. Configure the name, description, file location, and WSDL binding for the service, and then click **Next**.

3. On the Transport page, select the transport protocol and update the endpoint URI. For required URI formats, see the online help provided with Service Bus.
4. Click **Finish**.
5. Configure the business service, as described in [Configuring Business Services](#).

Configuring Business Services

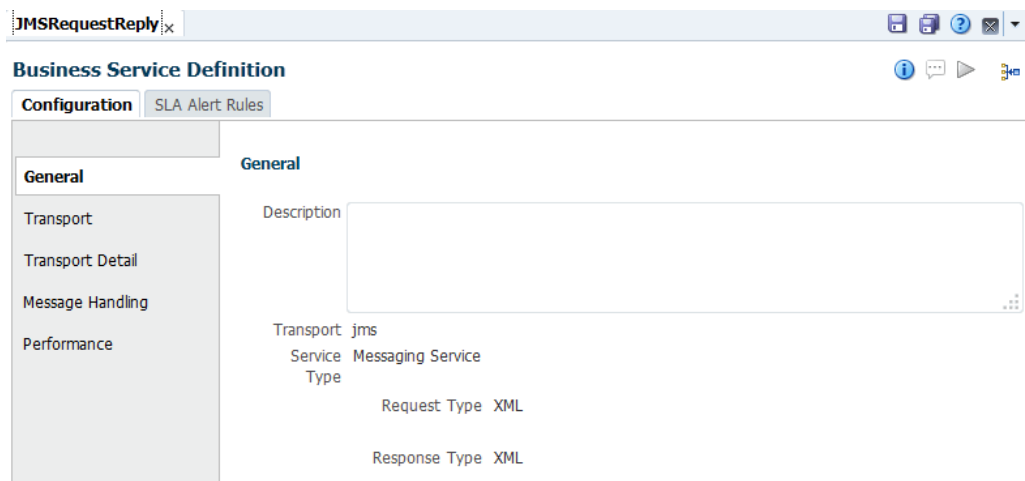
Once you create a business service, you can edit the configuration, add security policies, modify security settings, and set up SLA alert rules. The information you can modify depends on how the service was originally configured. For a list of all the configurable properties for business services, see the online help available for each Business Service Definition Editor page.

If you are working in the Oracle Service Bus Console, make sure you are in an active session before performing any of the tasks in this section.

How to Configure General Information for a Business Service

The General tab of the Business Service Definition Editor displays information about the service such as a description of the service, the transport used by the service, the service type, and any WSDL ports or bindings. You can only modify the description on this page. The following figure shows the General tab in the Oracle Service Bus Console.

Figure 9-1 Business Service General Configuration Page in the Console



To configure general information for a business service:

1. In the Project or Application Navigator, right-click the business service to edit, and click **Open**.
2. Click the General tab if it is not already the visible page.
3. Enter or update the description for the service.
4. If the service references any resources, such as a WSDL or MFL document, click the name of the resource to view the document in its own editor.
5. When you are done making changes, click **Save All**.

- If you are working in the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Configure a Business Service Transport

Use the Transport and Transport detail page to configure the transport for the business service. The available properties vary for each transport. The following figure shows the Transport tab in the Oracle Service Bus Console.

Figure 9-2 Business Service Transport Configuration Page in the Console

The screenshot displays the 'Business Service Definition' page for 'JMSRequestReply'. The 'Configuration' tab is selected, and the 'Transport' sub-tab is active. The configuration includes:

- Protocol:** jms
- Load Balancing Algorithm:** Round Robin
- Endpoint URIs:** A table with one entry: jms://localhost:7021/QConn/RequestQ
- Retries:**
 - Retry Count: 0
 - Retry Iteration Interval: 30 (seconds)
 - Retry Application Errors:

To configure a business service transport:

- In the Project or Application Navigator, right-click the business service to edit, and click **Open**.
- Click the Transport tab, and do any of the following:
 - To change the load-balancing algorithm, select a new algorithm from the list of available options.
For more information, see [About the Load Balancing Algorithm](#).
 - Update or add endpoint URIs. For more information, see the online help provided for this page.
 - In the **Retry Count** field, specify the number of times to retry URI endpoints.
 - In the **Retry Iteration Interval** field, specify the amount of time in seconds to wait after trying all URIs before trying again.
 - Select or clear **Retry Application Errors** to specify whether to retry application errors.
- Click the Transport Detail tab.

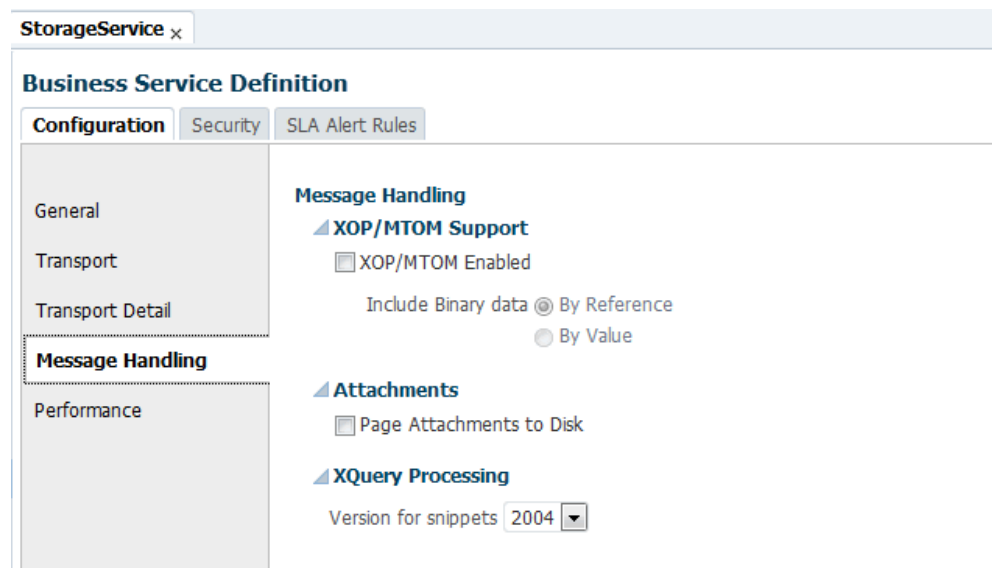
The properties you can configure here are based on the transport for the business service. For information about specific transports, see [Working with JCA Adapters, Transports, and Bindings](#) or the online help for the Transport Detail tab.

4. When you are done making changes, click **Save All**.
5. If you are working in the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Configure Business Service Message Handling

On the Message Handling page, you can configure how the business service processes message contents, including MTOM/XOP support, attachments, checking for WS-I compliance, and the XQuery version to use. The following figure shows the Message Handling tab in the Oracle Service Bus Console.

Figure 9-3 Business Service Message Handling Page in the Console



To configure message handling for a business service:

For information about message handling properties, see [Message Handling for Business Services](#).

1. In the Project or Application Navigator, right-click the business service to edit, and click **Open**.
2. Click the Message Handling tab.
3. To enable XOP/MTOM support, select **XOP/MTOM Enabled**, and select whether to include binary data by reference or value.
4. To specify how MIME attachments are handled, select or clear **Page Attachments to Disk**.
5. To check messages for WS-I compliance, select the check box for **Enforce WS-I Compliance**.
6. Select the version of XQuery to use for processing, either 2004 or 1.0.

Note:

XQuery 1.0 is the recommended version. Support for XQuery 2004 will be deprecated in future releases.

7. When you are done making changes, click **Save All**.
8. If you are working in the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Configure Performance for a Business Service

On the Performance tab, you can configure result caching to improve the performance of the business service. For more information, see [Improving Performance by Caching Business Service Results](#). For instructions, see [How to Configure a Business Service for Result Caching](#).

How to Configure Security for a Business Service

You can secure business services through multiple methods, including Oracle Web Services Manager (WSM) policies and access control at the transport level. For more information about securing business services, see [Security and Security Policies for Business Services](#) and [Securing Business and Proxy Services](#).

How to Configure Service Level Agreement Alerts for a Business Service

SLA alerts let system administrators know when certain conditions are met that indicate the health of a business service. For information about defining SLA alerts, see "Creating Service Level Agreement Alert Rules" in *Administering Oracle Service Bus*.

Deleting Business Services

You can delete a business service even if it is referenced by other resources, though this might result in conflicts due to unresolved references to the deleted resource.

How to Delete a Business Service

Check for dependencies before removing a business service. In the Oracle Service Bus Console, open the business service in the Business Service Definition Editor and click the **References** icon in the upper right to find out whether any services are using it. In JDeveloper, right-click the business service and select **Explore Dependencies**.

To delete a business service:

1. In the Project or Application Navigator, right-click the business service to delete, and select **Delete**.

A confirmation dialog appears.

2. In JDeveloper, if other resources reference this business service the confirmation dialog displays the number of references. Click **Show Usages** to view information about the reference.

3. On the confirmation dialog, click **Yes** to confirm you want to delete the service.

The business service is deleted.

4. If you are working in the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Improving Performance by Caching Business Service Results

If you use business services that return results that do not change often, you can configure those business services to cache results. When you enable result caching, the service returns results from the cache rather than invoking the external service. This improves performance by reducing network overhead to access the external service. Result caching also helps improve scalability by reducing the load on the back-end servers that host the external service.

In this section, the term *result cache* refers to the cache itself, which all business services share to store their respective results, and the term *cached result* refers to a single result in the result cache. For business services that use result caching, you can control the time to live for the cached result. After the cached result expires, the next business service call results in invoking the back-end service to get the result. This result is then stored in the cache for future requests to access.

Service Bus uses the result caching mechanism of Oracle Coherence, which is included with WebLogic Server. The Service Bus implementation of result caching includes global enable/disable, cache message variables, configuration fields on each business service, and cache options for service statistics, debugging, and alert rules.

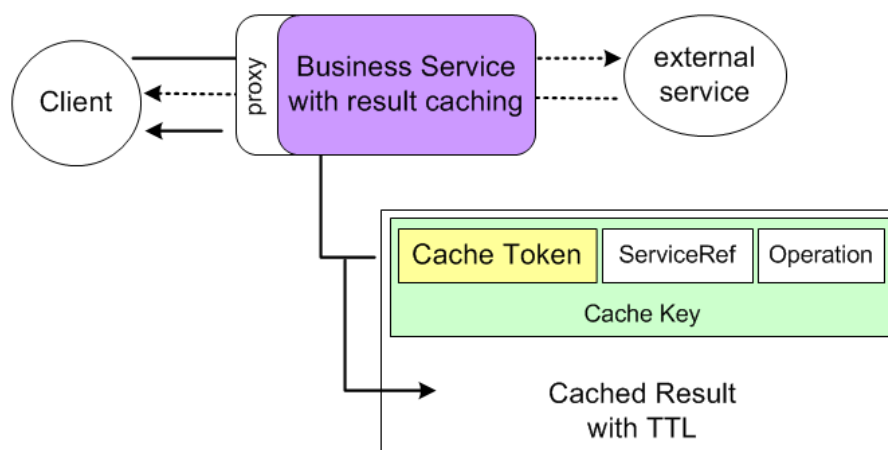
How Result Caching Works

Figure 9-4 illustrates a client invoking a business service and receiving a response that contains cached results.

Note:

Result caching works only with request/response operations.

Figure 9-4 Business Service Result Caching



Each cached result is uniquely identified by a cache key that is made up of the ServiceRef (the unique identifier for the service which is the fully qualified path name of the service), the operation being invoked, and a cache token string. The cache token helps to uniquely identify a single cache result among other cache results for one business service. You control the value of the cache token. You can set the cache token either by configuring the cache token expression in the result caching configuration for

the business service or by using the cache-token metadata element in `$transportMetaData` using the pipeline.

If the business service locates cached results through a cache key, it returns those cached results to the client instead of invoking the external service directly.

In [Figure 9-4](#), the solid arrows represent the message path between the client and a cached result. The dotted arrows show the message path if no cached result exists. If no cached result exists, the business service invokes the external service directly, returns the result to the client, and stores the result in cache. A result cache could be empty for a number of reasons, such as for a first-time invocation where no cache exists yet, a caching error, or the cache was flushed.

For cache expiration, cached results have a time-to-live (TTL) attribute. You can configure cache expiration either with the Expiration Time property in the result caching configuration on the business service or the cache-ttl element in `$transportMetaData` using the pipeline. If Coherence finds that the TTL has expired, it flushes the cache, and the business service invokes the external service for a result. That result is then stored in the cache (if there is no error in the result), and the result is available in the cache so that it can be returned to the next request.

Flushing Cached Results

Service Bus with Coherence can flush an individual cached result or all cached results for a business service. The following events illustrate how the cache is flushed:

- Cache TTL has expired. Each cached result has its own TTL. When a TTL is reached, Coherence flushes that individual cached result.
- Disable result caching on a single business service. When you disable result caching on a business service, Service Bus triggers flushing of all cached results for that business service in Coherence.
- Update, rename, or delete a business service. These actions trigger the flushing of all cached results for that business service from Coherence.
- Update a dependent resource. Updating a dependent resource, such as a WSDL document, triggers the flushing of all cached results for that business service from Coherence. However, changes to the following dependent resources do not cause cache flushing: service provider, UDDI registry, and alert destination.
- Globally disable result caching. Globally disabling result caching, triggers the flushing of the entire result cache (all cached results for all business services) from Coherence.

Result Caching Best Practices

Because cached results bypass the security of invoking an external service directly, do not use result caching with business services that provide security with a non-static service account or a WS-Security policy. Before deploying a Service Bus environment that will use result caching in production, you should plan and implement Coherence setup and configuration to allow for best performance, as described in *Understanding Configuration in Developing Applications with Oracle Coherence*.

Result Cache Metadata

The result cache uses a cache key to identify cached results, and an expiration time to determine when to flush cached results.

- [Cache Token](#)
- [Expiration Time](#)
- [Request Metadata](#)
- [Response Metadata](#)

Cache Token

Service Bus uses cache keys to identify cached results for retrieval or population, and the cache token portion of the cache key provides the unique identifier. You can use an expression, the *cache token expression*, to generate the cache token part of the cache key to uniquely identify a cached result for the business service. To generate the cache token from a value in the request (in the pipeline or split-join that invokes the business service), use an expression that gets the value from the pipeline `$body`, `$header`, `$operation`, or `$transportMeta-data` (`$outbound/ctx:transport/ctx:request` or `$outbound/ctx:transport/ctx:response`). For example, you can populate the cache-token from a customer ID in the message `$body`.

The cache token expression must resolve to a String or the value of simple content, such as an attribute or an element with no child elements. If the expression evaluates to null or causes an error, results are not cached. You can also generate the cache token from the request, without setting a cache token expression in the business service configuration. To do this, include a value in `$outbound/ctx:transport/ctx:request/ctx:cache-token` in the pipeline. Any value in that cache-token overrides the cache token expression in the business service configuration.

Expiration Time

The expiration time, or *time to live* (TTL), determines when an entry in the business service's result cache is flushed. You can use the default expiration time, define a duration of time before the result cache is flushed, or define an expression that generates the expiration time from a value in the request or response. The default expiration time is defined in the `expiry-delay` value in the `osb-coherence-cache-config.xml` file in `resultcache.gar`. You can define the duration directly in the business service configuration.

To generate the expiration time from a value in the request or response, use an expression that gets the value from the pipeline or split-join `$body`, `$header`, `$operation`, or `$transportMeta-data` (`$outbound/ctx:transport/ctx:request` or `$outbound/ctx:transport/ctx:response`). For example, use a value you have set in the Cache-Control HTTP header.

The expiration time must resolve to an integer (representing seconds), an XQuery `dayTimeDuration` (XSD type), or the integer value of simple content representing seconds, such as an attribute or an element with no child elements. If the expression evaluates to null or causes an error, results are not cached.

You can also generate the expiration from the request, without setting an expiration time in the business service configuration. To do this, include a value in `$outbound/ctx:transport/ctx:request/ctx:cache-ttl` in the pipeline or split-join. Any value in the `cache-ttl` element overrides the expiration time in the business service configuration.

Request Metadata

The request metadata used with result caching include **cache-token** and **cache-ttl**, both String values. You can configure both in the business service configuration. You

can also leave the cache token or TTL undefined in the business service and provide the cache token or TTL in the request with these metadata. When you set the cache token or TTL in the request, those values override any cache token or TTL you have defined in the business service configuration.

When using expressions to configure result caching, whether with the cache token expression, the TTL, or both, you can enter the namespaces and corresponding prefixes to use in the expressions. This field also lets you view a list of existing namespaces.

Response Metadata

The response metadata used with result caching include the following:

- **cache-token:** Contains the cache token that was used to retrieve content from the result cache or add content to the result cache after invoking the external service.
- **cache-originated:** Contains a boolean value, true or false. A value of true means the returned content came from the result cache. A value of false means the returned value came from invoking the external service.

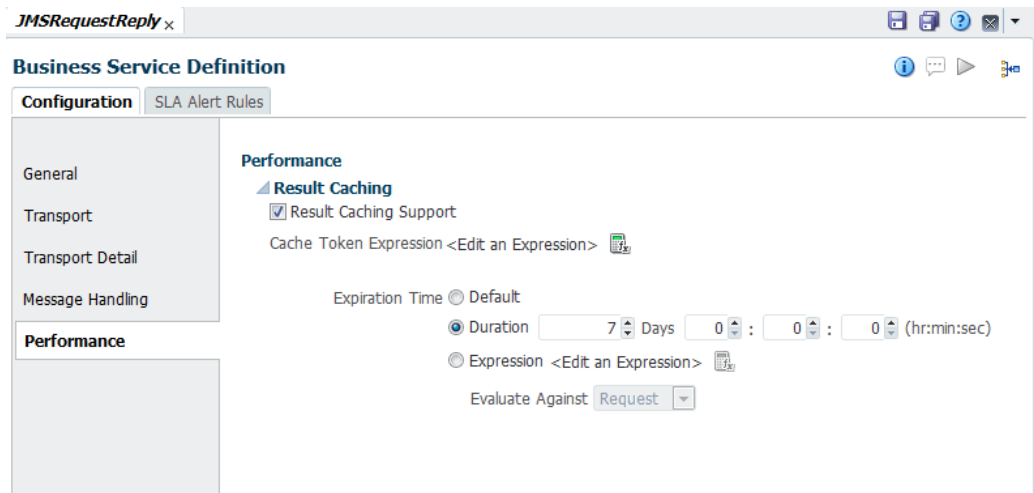
Testing Result Caching

Result caching takes effect only when the business service configured with result caching is invoked (for example, with a route or service callout activity) from a pipeline or split-join. Therefore, in order to test result caching, do not invoke the business service directly from the Test Console. Instead, use the Test Console to test the pipeline or the split-join that invokes the business service.

How to Configure a Business Service for Result Caching

If you invoke business services whose results seldom change, result caching improves business service performance by returning cached results to the client instead of invoking an external service directly. You can only configure result caching for a business service using the Oracle Service Bus Console.

For both cache token expressions and expiration time expressions, you use the Expression Editor to define the expression. For more information about working with the Expression Editor, see [Transforming Data with XQuery](#). The following image shows the Performance tab in the Oracle Service Bus Console, where you configure result caching.

Figure 9-5 Business Service Performance Page in the Console

To configure result caching:

1. In the Oracle Service Bus Console Project Navigator, navigate to and open the business service you want to configure.
2. On the Business Service Definition Editor, select the **Performance** subtab.
3. Select **Result Caching Support**.

Note:

Even though result caching is enabled here, you must also enable it in Fusion Middleware Control as described in "Configuring Operational Settings at the Global Level" in *Administering Oracle Service Bus*.

4. To define an expression to generate the cache token part of the cache key, click the **Expression Editor** icon next to the **Cache Token Expression** field.

For more information about the cache token expression, see [Cache Token](#).

5. To define an expiration time for the business service's result cache, select one of the following:
 - **Default:** This option uses the `expiry-delay` value in the `osb-coherence-cache-config.xml` file in `resultcache.jar`. The default is 5 minutes.
 - **Duration:** This option lets you specify a length of time to live. Use the **Days** and **hr:min:sec** fields to define the duration.
 - **XQuery Expression:** This option uses an XQuery expression that gets an expiration time from the request or response. To define the expression, click the **Expression Editor** icon next to the **Expression** field. After you define the expression, select whether to evaluate the expression against the request or response in the **Evaluate Against** field.

Note:

A duration of zero (0) means no expiration. A negative duration means do not cache.

For more information about the expiration time, see [Expiration Time](#).

Result Caching Advanced Configuration

In each Service Bus domain, you can modify how the domain uses Coherence for business service result caching. Service Bus provides its own default Coherence configuration for the servers in a domain by providing two files, `resultcache.gar` and `resultcache.ear`, both located in `MW_HOME/osb/lib/apps`. The GAR file defines the Coherence cache to use for result caching. In order to set up the results cache, you deploy `resultcache.gar` to WebLogic Server and specify the servers or clusters to which the cache is targeted. You can then configure the cache using the console (or WLST commands, if you prefer).

The default cache configuration is embedded in the GAR file. You can define your own configuration for the cache by extracting `osb-coherence-cache-config.xml` from the GAR file and modifying the properties as needed. By default, a distributed cache scheme is used for the result cache. For more information, see "Cache Configuration Elements" in *Developing Applications with Oracle Coherence*.

To use a different cache configuration, you need to create a new Cache Configuration for the Coherence cluster in the WebLogic Server Administration Console. The Cache Configuration must be named `/osb/service/ResultCache`, the JNDI name must be `servicebus/result-cache`, and it must be in the same Coherence Cluster used by Service Bus. You can use a different configuration on different servers in your domain.

Working with Unicast and Multicast

You can configure unicast settings to restrict Coherence cache access to only the local server. With this configuration, nodes started on different servers do not join the same Coherence cluster to share cached information. Alternatively, you can configure multicast values to create a Coherence cluster that is shared by any WebLogic Server node on the same subnet created from the same template. You configure these properties on the General Configuration tab of the Coherence cluster in the WebLogic Server Administration Console.

A best practice is to configure the Coherence cluster to use a unicast listener with an explicit list of nodes for the Coherence cluster. For information about multicast and unicast properties, see the online help provided with WebLogic Server. Also see "Using Well Known Addresses" in *Developing Applications with Oracle Coherence*.

You can specify overrides using system properties. Use the following guidelines when configuring the Coherence cluster to ensure the correct sharing of a Coherence cluster among multiple servers:

- If you want to switch from a multicast listener to a unicast listener in a cluster, configure well-known addresses.
- If you have multiple WebLogic Server clusters in the same subnet, modify the relevant Coherence address and port properties to ensure correct sharing of a Coherence cluster. Do not use the same address and port as those used for the WebLogic Server cluster.

- If you have multiple Admin Servers with Managed Servers in the same subnet, modify the relevant Coherence address and port properties to ensure correct sharing of a Coherence cluster.
- If you have any combination of WebLogic Server clusters and Admin Servers with Managed Servers in the same subnet, modify the relevant Coherence address and port properties to ensure correct sharing of a Coherence cluster.
- If multiple Coherence clusters are running in the same subnet, modify the Multicast Address and Multicast Port to specify which Coherence cluster a node should connect to.

How to Disable Coherence for Service Bus

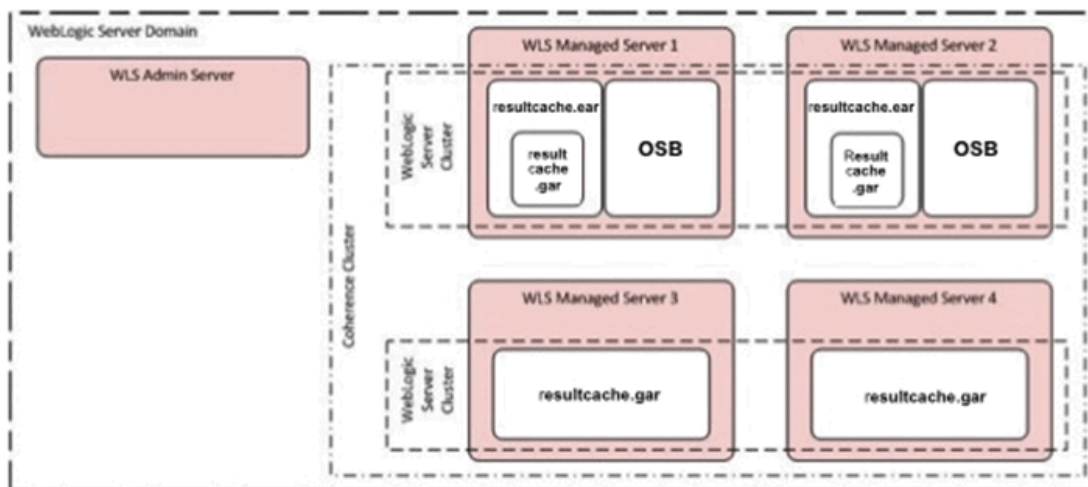
To prevent Service Bus from using Coherence completely, perform the following steps:

1. Delete all Coherence cluster resources targeted to the servers that are running Service Bus.
2. Undeploy the Service Bus result cache enterprise application (`resultcache.ear`).
3. Disable global result caching.

About Out-of-Process Coherence Servers

The following figure illustrates an out-of-process Coherence server.

Figure 9-6 Out-of-Process Coherence Cluster



In the above example, there are two WebLogic Server clusters. The first WebLogic Server cluster contains Managed Servers 1 and 2 and also has the following:

- Service Bus running
- Service Bus results cache enterprise application deployed (`resultcache.ear`)
- Storage disabled

The second WebLogic Server cluster contains Managed Servers 3 and 4 and also has the following:

- Service Bus result cache grid archive deployed (`resultcache.gar`)

- Storage enabled

All the cached entries will be stored in Managed Servers 3 and 4.

How to Use an Out-of-Process Coherence Cache Server

If you plan to use result caching heavily with Service Bus and want to avoid using too much heap space for result caching, you can set up a Coherence cache server to run on its own JVM rather than sharing a Service Bus domain JVM. Running a Coherence cache server outside of a Service Bus JVM—out of process—lets the Coherence cache server use its own heap space without affecting the heap space Service Bus uses to process messages.

Note:

Any out-of-process Coherence cache server used with Service Bus must use the same version of Coherence as the version included with Service Bus.

Creating an Out-of-Process Coherence Cache Server

To create an out-of-process Coherence cache server:

1. Create new WebLogic Server nodes and clusters that use the same Coherence cluster.
2. Deploy the `MW_HOME/osb/lib/apps/resultcache.gar` file to the new clusters.

Configuring the Servers for an Out-of-Process Coherence Cache Server

In order to use an out-of-process Coherence cache server, you need to disable local caching on each Service Bus node.

To configure the servers for an out-of-process Coherence cache server:

1. Disable local caching on each Service Bus node by adding the following argument to the Service Bus node startup:

```
-Dtangosol.coherence.distributed.localstorage=false
```

2. Set the Coherence cluster name with the following argument.

```
-DOSB.coherence.cluster=cluster_name
```

More Information on Configuring and Using Oracle Coherence

You can perform many other types of cache configuration flexibly, without changing application code, using the Oracle Coherence configuration framework. For example, you can use attributes to modify the cache type and behavior, and you can query the cache. For more information, see *Using Caches in Developing Applications with Oracle Coherence*.

Improving Service Performance with Split-Join

This chapter provides an overview of split-joins and how to create them, and also demonstrates static and dynamic split-join scenarios. Split-join is an advanced mediation feature that helps you improve service performance by concurrently processing individual messages in a request.

This chapter includes the following sections:

- [Introduction to Split-Joins](#)
- [Service Level Agreement Alert Rules](#)
- [Working with Split-Joins in JDeveloper](#)
- [Adding Communication Operations in JDeveloper](#)
- [Adding Flow Control Operations in JDeveloper](#)
- [Adding Assign Operations in JDeveloper](#)
- [Working with Split-Joins in the Oracle Service Bus Console](#)
- [Static and Dynamic Split-Join Samples](#)

Introduction to Split-Joins

A split-join is a mediation pattern that can be used in a Service Bus to direct the flow and processing of messages. Split-joins let you split a service payload, such as an order, into individual messages that are sent to multiple services concurrently, as opposed to standard sequential processing. This greatly improves service performance. Split-join achieves this by splitting an input message payload into sub-messages (split), routing them concurrently to their destinations, and aggregating the responses into one overall return message (join). This process of payload splitting and response aggregation is called a split-join pattern.

Split-joins are particularly useful for optimizing overall response times in scenarios where payloads delivered by faster systems are being directed to responding services on slower systems. Without split-join, individual messages in a payload are normally resolved in sequential order by the recipient, which can take a long time if the responding system is slow. With split-joins, multiple messages are processed simultaneously, which reduces burden on the responding system and greatly improves response times. Without a split-join, the overall response time is the sum of the individual response times for each message. With a split-join, the overall response time is roughly that of the longest individual message's response time plus some minor system overhead.

You create and design split-joins in the JDeveloper Split-Join Definition Editor. You can then export the split-join and its associated resources, and import those resources to the Oracle Service Bus Console for testing and production. A split-join is saved to a `.flow` file in JDeveloper, and is always based on a WSDL operation. A split-join can be invoked from a proxy service, a pipeline, or another split-join. A split-join can invoke a proxy or business service, a pipeline, or another split-join.

There are two types of split-join pattern: static split-join to handle a known number of requests, and dynamic split-join to handle a variable number of requests. These patterns are described in the following sections.

Note:

A split-join can invoke another split-join in the same Service Bus configuration, providing more flexibility in service design by letting you split up complex split-join functionality into multiple split-joins. This allows for componentization and reuse of split-join functionality. Performance is maintained, because there is no marshalling and unmarshalling of data between the split-joins.

Ensure that you do not create circular split-join references; Service Bus does not check for circular references.

Static Split-Joins

A static split-join branches from the main execution thread of a Service Bus message flow by splitting a payload into a fixed number of new branches according to the configuration of the split-join. At design time you determine the number and variety of services to be invoked. For instance, a customer places an order for a cable package that includes three separate services: internet service, TV service, and telephone service. In the static use case, you could execute all three requests in separate parallel branches to improve performance time over the standard sequential execution.

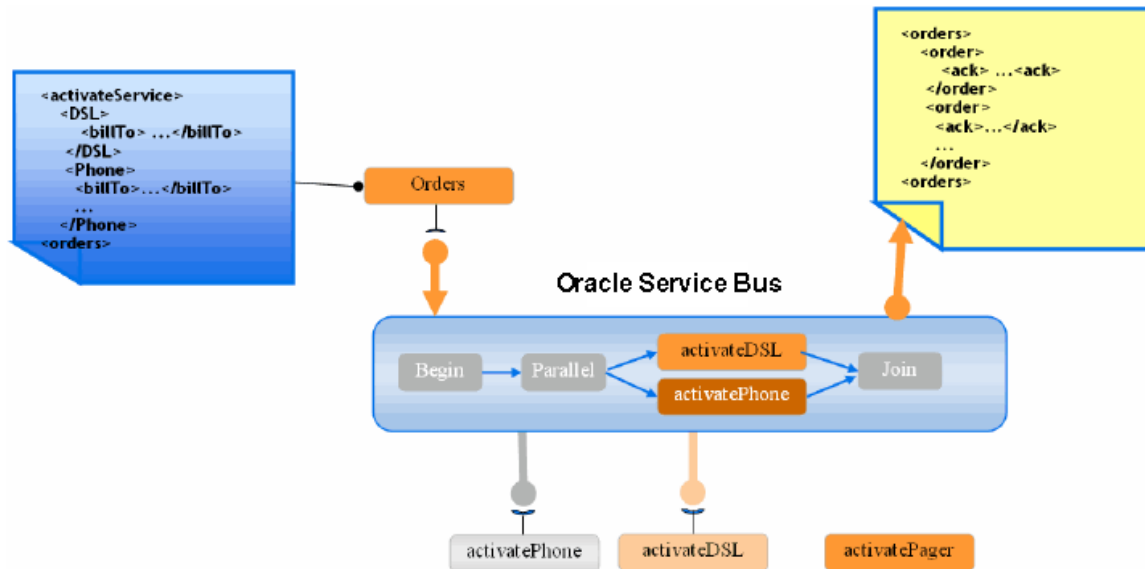
Static Split-Join – Sample Scenario

This scenario illustrates a telco company that employs static split-join to process a customer's order for a communications services package. In this case, the customer might sign up for DSL and voice services all at once. Rather than executing each request in the payload separately in order, the telco can execute the messages in parallel using a static split-join.

Static split-join is the ideal pattern in this case because you know there will always be exactly two incoming service requests for this particular service package: DSL and voice. Splitting the requests into parallel branches allows them to be processed concurrently, which improves the overall response time for processing the payload. After all messages are processed, the generated responses are aggregated back into one reply in the execution thread.

[Figure 10-1](#) illustrates a static split-join that splits two known service requests, DSL activation and phone activation, processes each request in parallel, and joins the responses into a single reply.

Figure 10-1 Static Split-Join – Known Number of Service Requests



Dynamic Split-Join

A dynamic split-join branches from the main execution thread of a Service Bus pipeline by dynamically creating new branches according to the contents of the incoming payload. The number of message requests created is variable. A dynamic split-join uses conditional logic to determine the number of branches to create. All requests are handled simultaneously, and the responses are aggregated into a single reply. For instance, a retailer places a batch order containing a variable number of individual purchase orders. In the dynamic use case, you could parse the batch order and create a separate message request for each purchase. Like the static use case, these messages are then executed in parallel for improved performance.

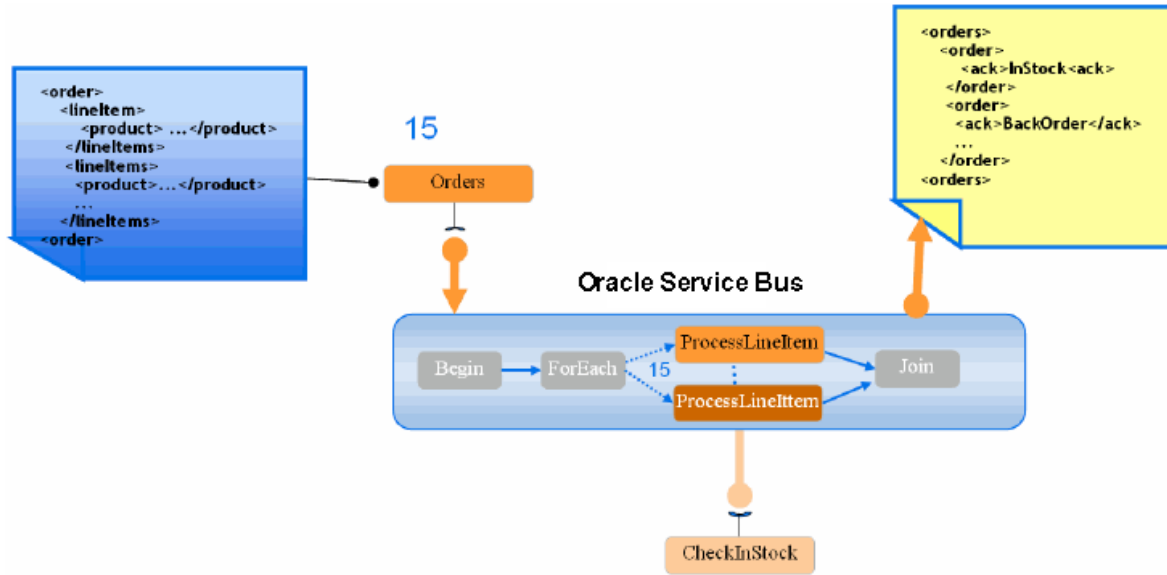
Dynamic Split-Join – Sample Scenario

This scenario illustrates a company that uses dynamic split-join when it places automated stationery orders for its employees. If the orders are automatically placed every week based on employee submissions, there is no way to know how many individual orders are included in any one weekly order. Rather than placing each order separately, the company could use a dynamic split-join to place the orders concurrently using a dynamic split-join.

Dynamic split-join is the ideal pattern in this case, because there is no way of knowing how many orders will be submitted each week. The dynamic split-join loops through all the orders and places them in parallel. You can also limit the number of orders that are processed. After all of the orders have been processed, the generated order responses are aggregated back into one reply in the execution thread.

Figure 10-2 illustrates a dynamic split-join that splits 15 orders, processes them concurrently, and joins the responses into a single reply.

Figure 10-2 Dynamic Split-Join – Unknown Number of Service Requests



Split-Join Operations

The Split-Join Components window lists all the operations you can use to construct a split-join. The operations are divided into the following categories: Communication, Flow Control, and Assign.

Split-Join Communication Operations

Communication operations define how the split-join interacts with external services. The available operations are described in [Table 10-1](#).

Table 10-1 Split-Join Communication Operations

Operation	Description
Invoke Service	This operation invokes a WSDL-based, non-transport-typed business service, a WSDL-based proxy service, a WSDL-based proxy service, or a split-join. See How to Invoke a Service .
Reply	This operation sends a response or fault back to the Oracle Service Bus message flow. See How to Configure a Reply .

Split-Join Flow Control Operations

Flow control operations define how incoming messages flow through the split-join.

Table 10-2 Split-Join Flow Control Operations

Operation	Description
For Each	This operation executes the logic configured within its Scope a specified number of times. See How to Iterate Through a Variable Number of Requests .

Table 10-2 (Cont.) Split-Join Flow Control Operations

Operation	Description
Condition Operations	<p>Condition operations let you define conditions that evaluate to true or false, and then carry out the behavior defined for each condition. You can define any of the following conditions:</p> <ul style="list-style-type: none"> • If: The associated If branch of an if-else operation is executed when the condition evaluates to true. Else-if operations also appear in the conditional node. The associated Else If branch of an if-else operation is executed when the initial If condition evaluates to false but the secondary condition evaluates to true. See How to Define If-Else Conditional Logic. • While: The associated operation is repeated until the condition evaluates to false. The condition is evaluated before each loop commences. See How to Repeat an Operation Until it Evaluates to False. • Repeat Until: The associated operation is repeated until the condition evaluates to true. The condition is evaluated after each loop finishes. See How to Repeat an Operation Until it Evaluates to True.
Parallel	<p>This operation creates a fixed number of configured parallel branches, so you can define a static split-join that handles a fixed number of message requests. Parallels contain one or more Scope branches. See How to Process a Fixed Number of Requests in Parallel.</p>
Raise Error	<p>This operation generates an error that causes the split-join to stop normal processing. If the error is not handled using an error handler, the split-join will terminate and a Fault will be sent to the Oracle Service Bus message flow. See How to Raise an Error.</p>
Re-Raise Error	<p>This operation lets you re-raise an error caught by an error handler catch or catch all. You can configure a name and description for a Re-Raise Error operation. See How to Re-Raise an Error.</p>
Scope	<p>This operation creates a context that influences the behavior of its enclosed operations. Local variables and the error handler defined within the scope are restricted to this context. There are no configuration properties for a Scope operation. See How to Create a Container Node.</p>
Wait	<p>This operation inserts a pause in the split-join flow for a short duration to wait for other dependent jobs to complete. After the specified duration is reached, the split-join execution resumes. See How to Insert a Pause in Processing.</p>

Split-Join Assign Operations

The assign operations let you manipulate the data in the message you process, including initializing and updating a variable. You can perform the following operations in an assign node: assign, copy, delete, insert, Java callout, log, and replace.

Table 10-3 Split-Join Assign Operations

Operation	Description
Assign	This operation lets you assign the result of an XQuery expression to a variable. See How to Assign a Value to a Variable .
Copy	This operation lets you copy the information specified by an XPath expression from a source document to a destination document. See How to Copy a Value from a Source to a Destination Document .
Delete	This operation lets you delete a set of nodes specified by an XPath Expression. See How to Delete a Set of Nodes .
Insert	This operation lets you insert the result of an XQuery expression at an identified place relative to nodes selected by an XPath Expression. See How to Insert the Result of an XQuery Expression .
Java Callout	This operation lets you invoke a static Java method from a split-join for custom actions to be handled in Java such as validation, transformation, and logging. See How to Invoke a Java Method in a Split-Join .
Log	This operation lets you log data at a specified severity so that administrators can take appropriate action. See How to Log Split-Join Data .
Replace	This operation lets you replace a node or the contents of a node specified by an XPath expression. See How to Replace a Node or Its Contents .

Using Split-Join with Content in SOAP Headers

You can use split-join to enhance the performance of services that place message content in SOAP headers. By default, split-joins do not propagate SOAP headers; however, you can modify the WSDL file to accommodate this, allowing proxy services to pass SOAP headers into split-joins and allowing split-joins to pass SOAP headers to proxy and business services as an invocation or response.

To enable this capability, you must declare the header parts along with the body parts in a single request/response message in the split-join WSDL file and in the WSDL file of the proxy or business services invoked by the split-join. With the message parts declared in the WSDL files, SOAP header content is available to split-joins in the request/response message variables.

Following is an example of the message and binding definitions in the WSDL file.

Message

```
<wsdl:message name="retrieveCustomerOverviewByIdRequestMessage">
  <wsdl:part name="retrieveCustomerOverviewByIdRequest"
    element="co:retrieveCustomerOverviewByIdRequest"/>
  <wsdl:part name="serviceContext" element="sc:serviceContext"/>
</wsdl:message>
```

Binding

```
<wsdl:input>
<soap:body use="literal" parts="retrieveCustomerOverviewByIdRequest" />
  <soap:header message="tns:retrieveCustomerOverviewByIdRequestMessage"
    part="serviceContext" use="literal" />
</wsdl:input>
```

Transaction Support

Split-joins provide support for propagating transactions. Many split-join operations provide an option for setting specific quality of service (QoS) values, which control transaction support. The QoS value of **Exactly Once** on a split-join operation ensures the operation executes in the context of a transaction if one exists.

Setting QoS values on individual operations gives you the flexibility to execute multiple operations in the context of a transaction and execute other operations outside of a transaction in a single split-join. Operations set with a QoS of **Exactly Once** are executed in the transaction. Operations set with a QoS of **Best Effort** do not execute in the context of a transaction.

Split-joins do not handle transaction rollback in the case of exceptions. It is the responsibility of the service component that called the split-join to handle transaction exceptions and rollback.

The following split-join operations support transaction propagation:

- Invoke Service
- Assign
- Delete
- Insert
- Java Callout
- Replace

Security with Split-Joins

Split-joins do not enforce security policies, which means you cannot create a split-join with a WSDL file that includes policies, and you cannot call a WSDL-based business service that contains WSDL policies from a split-join.

To ensure security enforcement when using split-joins, use proxy services to handle security enforcement in the following ways:

- Use the inbound proxy that invokes the split-join to enforce policies.
- If the split-join needs to invoke a WSDL business service that contains policies, have the split-join call a local proxy (configured without the security policies), which in turn invokes the business service with the required policies.

Split-Join Resource Type and Environment Variable

If you reference split-joins in any scripts or custom code, use the following values:

- **typeId:** FLOW
- **Work manager environment value type:** Work Manager

Note that work manager is another name for dispatch policy.

Service Level Agreement Alert Rules

Service Level Agreement (SLA) alert rules define conditions under which an alert is generated. These conditions are typically indicators of the overall health of the Service Bus application or of a specific service component. For information about defining SLA alert rules for a split-join, see "Creating Service Level Agreement Alert Rules" in *Administering Oracle Service Bus*.

Working with Split-Joins in JDeveloper

You can only create and configure split-joins in JDeveloper. JDeveloper provides a graphical modeling editor so you can easily model your split-joins by dragging operations to the canvas, and configuring their properties in the Properties window.

How to Create a Split-Join in JDeveloper

You create and configure split-joins in JDeveloper. If you develop Service Bus projects in the Oracle Service Bus Console, you can import the split-joins created in JDeveloper into the console. Split-Joins are defined in a `.flow` file, which is generated when you create a split-join in JDeveloper. The file might have references to proxy services, business services, or to external WSDL resources. Resources on which the split-join depends must be present on the server before you can activate the split-join.

To create a split-join in JDeveloper:

1. In Oracle JDeveloper, open or create the application and project to which you want to add the split-join.
2. In the Application Navigator, right-click the project, point to **New**, and then select **Split-Join**.

The Create Split-Join Service wizard appears.

3. On the Create Service page, enter a name for the split-join and, optionally, enter a location and description.

For naming requirements, see [Naming Guidelines for Service Bus Components](#). Do not use the following characters in split-join names: leading space, trailing space, / \ * : " < > ? |

4. Optionally, enter a brief description of the split-join or modify the location of the files.

By default, files are saved in the project folder.

5. Click **Next**.
6. On the Type page, do one of the following:
 - To use an existing WSDL file, click **Find existing WSDLs** to the right of the **WSDL** field and search for and select the WSDL file to use.
 - To create a new WSDL file, click **Generate WSDL from schema(s)** to the right of the **WSDL** field. For more information, see "Generating a WSDL File" in *Developing SOA Applications with Oracle SOA Suite*.
7. Select the WSDL binding and operation to use from the options lists that appear for those fields.

8. To generate a proxy service from this split-join, select **Expose as Proxy Service**, and enter the following information:
 - **Proxy Name:** A unique name for the proxy service. See the naming requirements above.
 - **Proxy Location:** The path where you want to store the proxy service file. The default is the project folder.
 - **Proxy Transport:** Select the type of transport to use for the proxy service. For more information, see [Working with JCA Adapters, Transports, and Bindings](#).
9. Click **Finish**.

A basic split-join is created and displayed as a diagram in the Design view of the Split-Join Definition Editor. By default, it consists of a start node, a receive node, and a reply node (if the split-join is based on a request/response WSDL file). The start node contains the variables introspected from the WSDL operation. The receive node is used to receive incoming request messages. The reply node is used to send response messages.

How to Generate a Split-Join from a WSDL Document in JDeveloper

You can use a WSDL document that already exists in the Service Bus application to generate a split-join. If the WSDL document does not exist in the application, import it and then perform the following steps.

To generate a split-join from a WSDL document in JDeveloper:

1. In the Application Navigator, right-click the existing WSDL document, point to **Service Bus**, and then select **Generate Split Join**.
2. Name and configure the service, as described in [How to Create a Split-Join in JDeveloper](#).

How to Display the Components Window and Properties Windows

You select the operations you add to a split-join from the Components window, so the window must be visible in JDeveloper in order to configure a split-join. You configure each operation using the JDeveloper Properties window, which also must be visible.

Displaying the Components Window

To display the Components Window

- If the Components window is not visible on the JDeveloper window, select **Window > Components**.

The Components window appears on the right side of the JDeveloper window.

Displaying the Properties Window

To display the Properties Window

- If the Properties window is not visible on the JDeveloper window, select **Window > Properties**.

The Properties window appears on the right side of the JDeveloper window. You can move this window to a different location, such as below the Split-Join Definition Editor, for better readability.

How to Configure the Start Node

The Start Node is generated automatically when you create a new split-join. It is the starting point from which all the other nodes proceed. The only element to configure in the Start Node is its name.

To change the Start Node name:

To change the name of the label for the Start Node, click the name of the node. In the field that appears, enter a unique, identifying string for the node. The name you enter appears underneath the node in the split-join editor

How to View External Services

The external services listed in the Start Node are those invoked outside the context of the split-join. They are specified in an Invoke Service node but are listed here for convenience.

To view external services:

To view external services, click the left-arrow button on the Start Node. The External Services box appears to the left of the Start Node. Hover your mouse over an external service to view the complete path of the service resource.

How to Configure Global and Local Variables

Variables in the Start Node store data that can be referenced globally, that is by any node in the split-join. By default, every Start Node is assigned both a request and a response variable when the split-join is initially created. From the Start Node, you can either create a new global variable or edit an existing global variable.

Variables in a Scope store data that can be referenced locally, that is only by the node to which it is attached. From a Scope, you can create new local variables or edit an existing local variable.

For more information about global and local variables, see [How to Create a Container Node](#).

Defining Global and Local Variables

To define global and local variables:

1. In the Split-Join Definition Editor, do one of the following:
 - To create a global variable, right-click the Start Node, and then select **Create Variable**.
 - To create a local variable, right-click the Scope to which you want to add the variable, and then select **Create Variable**.

The Create Variable Alias dialog appears.

2. Enter a name for the variable.
3. Select the variable type (Built in, XML Schema, or WSDL Message).
4. Click **Browse** next to the variable type you selected to select the type from the type chooser dialog.
5. Click **OK**.

The new variable appears in the list of variables in the Variables box.

Editing Global or Local Variables

To edit a global or local variable:

1. If the variable you want to edit is not visible on the editor, click the arrow button on the lower left side of the Start Node or Scope containing the variable.

The Variables box appears.

2. Right-click the variable to edit and select **Edit Variable**.

The Edit Variable Alias dialog appears.

3. Do any of the following:
 - Modify the variable's name.
 - Select the variable type (Built in, XML Schema, or WSDL Message).
 - Click **Browse** next to the selected variable type to select the type from the type chooser dialog.
4. Click **OK**.

How to Configure the Receive Operation

A Receive operation is generated automatically whenever you create a new split-join. The Receive operation places incoming request data in a variable and makes the contents available for later nodes to use. To configure the receive operation, you can specify the incoming message variable the Receive operation initializes and, optionally, a new name and description for the operation. A default variable, `request`, is automatically defined for the Receive operation.

To configure the receive operation:

1. If the Properties window is not visible on the JDeveloper window, select **Window > Properties**.
2. Select the Receive operation in the Split-Join Definition Editor.
3. To specify the incoming message variable, click the Receive tab on the Properties window and then do one of the following:
 - To select an existing variable, select the variable name from the list of available options in the **Request Variable** field.
 - To create a new variable, click **Create Variable** next to the **Request Variable** field. On the Create Variable dialog, enter a name for the variable and click **OK**.
4. To change the name of the Receive operation, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.

The new name appears beneath the node in the Split-Join Definition Editor.

5. In the **Description** field, enter any notes that you think are important.
6. In the JDeveloper toolbar, click **Save**.

Adding Communication Operations in JDeveloper

Communication operations define how the split-join interacts with external services. They include the Invoke Service and Reply operations.

- [How to Invoke a Service](#)
- [How to Configure a Reply](#)

How to Invoke a Service

Use the Invoke Service operation to invoke external, WSDL-based business services, WSDL-based proxy services, WSDL-based pipelines, and split-joins.

To invoke a service from a split-join:

1. Under Communication in the Components window, click **Invoke Service** and drag it onto the editor in the location in the flow where you want to invoke the service.
2. Select the new **Invoke Service** node.
3. In the Properties window, click the Invoke tab.
4. In the **Service** field, click **Browse** to select the service to invoke.
5. In the **Operation** field, select the operation upon which the Invoke Service is based.
6. In the **Request Variable** field, select a message type variable with the type matching the operation's input message type, or click **Create Variable** to create a new variable.
7. In the **Response Variable** field, select a message type variable with the type matching the operation's output message type, or click **Create Variable** to create a new variable.

Note:

An Invoke Service requires both a request variable and a response variable unless it is a one-way invocation. Either type of variable can be global (available within the entire split-join) or local (available within a particular context Scope.)

8. In the QoS field, select either **Best Effort** (not executed in the context of a transaction) or **Exactly Once** (executed in a transaction).
9. To change the name of the Invoke Service node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.

The new name appears beneath the node in the Split-Join Definition Editor.
10. In the **Description** field, enter any notes that you think are important.
11. On the JDeveloper toolbar, click **Save**.

How to Configure a Reply

A global Reply node is generated automatically when you create a new split-join. The purpose of the global Reply is to send a response back to the calling service. However, you can also create a Reply elsewhere in the split-join, including within error handlers.

The Reply can either send a response or a fault back to the client, depending on how you configure the variable. The available fault options vary depending upon whether the Reply is global or local.

- A global Reply (that is, a Reply in a split-join outside of an Error Handler) can never have a SOAP Fault but can have a WSDL Fault. This is why the SOAP Fault option is disabled in this case.
- A local Reply (that is, a Reply attached to an Error Handler) can have either a WSDL Fault or a SOAP Fault. WSDL Faults are available only if they were defined in the WSDL file upon which the split-join is based. The SOAP Fault option is always available provided one has been previously defined in the Error Handler.

Note:

Switching back and forth between the Response and Fault buttons will clear either configuration. For instance, if you previously selected **Propagate SOAP Fault** and you then switch to the **Response** configuration, **Propagate SOAP Fault** is no longer selected.

The default Reply operation automatically includes an implicit exit operation to end that instance of the flow without triggering a fault. The exit operation is not visible in the development environment.

To configure a reply:

In some circumstances, no Faults or only a SOAP Fault will be available for a Reply operation.

1. To add a new Reply operation, under Communication in the Components window, click **Reply** and drag it onto the editor in the appropriate location.
2. Select a **Reply** node to configure it.
3. In the Properties window, click the Reply tab.
4. To send a response back to the calling service, do the following:
 - a. Select the **Response** option.
 - b. In the **Response** field, select a message variable whose type matches the operation's output message type. Select from the list of available variables or click **Create Variable** to define a new message variable.
5. To send a WSDL fault as a response, do the following:
 - a. Select the **Fault Name** option.
 - b. In the **Fault Name** field, select the name of the fault to send back to the message flow from the list of existing faults.

- c. In the **Fault Variable** field, select the variable to which the fault will be assigned. Select from the list of available variables or click **Create Variable** to define a new fault variable.
6. To propagate the SOAP fault in the SOAP fault variable defined in the Error Handler, select **Propagate SOAP Fault**.
7. To change the name of the Reply node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.
The new name appears beneath the node in the Split-Join Definition Editor.
8. In the **Description** field, enter any notes that you think are important.
9. On the JDeveloper toolbar, click **Save**.

Adding Flow Control Operations in JDeveloper

Flow control operations define how incoming messages move through the split-join. They let you define conditional logic, define error handling, and define parallel processing.

- [How to Create a Container Node](#)
- [How to Iterate Through a Variable Number of Requests](#)
- [How to Process a Fixed Number of Requests in Parallel](#)
- [How to Define If-Else Conditional Logic](#)
- [How to Create Error Handlers](#)
- [How to Raise an Error](#)
- [How to Re-Raise an Error](#)
- [How to Repeat an Operation Until it Evaluates to True](#)
- [How to Repeat an Operation Until it Evaluates to False](#)

How to Create a Container Node

A Scope is a container that groups various elements together. The container creates a context that influences the behavior of its enclosed elements. Local variables and any error handlers defined within the scope are restricted to this context. However, some nodes within the scope may operate both locally (that is, within the scope) and globally (that is, outside of the scope.) For instance, an Invoke Service within a certain scope might call upon an service external to the scope's context.

Although variables are visible in the scope in which they are defined and in all scopes nested within that scope, a variable declared in an outer scope is hidden when you declare a variable with an identical name in an inner scope. For example, if you define variable `myVar` in an outer scope (`So`) and then define variable `myVar` again in an inner scope (`Si`) which is contained by the outer scope `So`, then you can only access the `myVar` you defined in the inner scope (`Si`). This `myVar` overrides the `myVar` you defined in the outer scope `So`.

To create a container node:

1. Under Flow Control in the Components window, click **Scope** and drag it onto the editor in the appropriate location in the flow.
2. Select the new **Scope** node.
3. To change the name of the Scope node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.

The new name appears beneath the node in the Split-Join Definition Editor.

4. Add operations to the scope to define its processing logic, and configure them using the Properties windows.

For information about the available operations, see [Adding Flow Control Operations in JDeveloper](#) and [Adding Assign Operations in JDeveloper](#).

5. To define local variables for the Scope node, see [How to Configure Global and Local Variables](#).
6. On the JDeveloper toolbar, click **Save**.

How to Iterate Through a Variable Number of Requests

A For Each node executes logic configured within its scope a specified number of times. Use the For Each operation to create conditional logic for iterating through a variable number of requests. It is primarily used to create dynamic Split-Joins.

To iterate through a variable number of requests:

1. Under Flow Control in the Components window, click **For Each** and drag it onto the editor in the appropriate location in the flow.
2. Select the new **For Each** node.
3. In the Properties window, click the For Each tab.
4. In the **Execution Mode** field, select whether to process each iteration of the For Each loop sequentially or in parallel. .
5. In the **Counter Variable Name** field, enter the name of the implicit variable that counts the iterations.
6. Next to the **Start Counter Value** field, click the Expression Builder icon to launch the XPath Expression Builder and define an expression that specifies the initial value of the counter variable.

Note:

The lowest possible starting and finishing counter value is "1."

7. Next to the **Final Counter Value** field, click the Expression Builder icon to launch the XPath Expression Builder and define an expression that specifies the final value of the counter variable after the last iteration.
8. Next to the **Number of Finished Branches** field, click the Expression Builder icon to launch the XPath Expression Builder and define an expression that determines when to stop creating branches.

9. To count only successfully completed branches to determine whether the completion condition has been met, select **Successful Branches Only**.
10. To change the name of the For Each node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.

The new name appears beneath the node in the Split-Join Definition Editor.

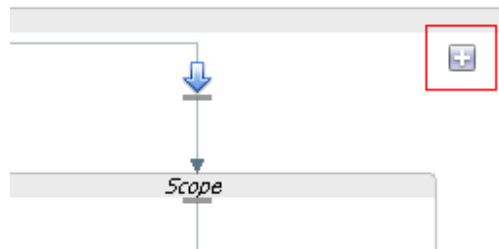
11. In the **Description** field, enter any notes that you think are important.
12. On the JDeveloper toolbar, click **Save**.

How to Process a Fixed Number of Requests in Parallel

A Parallel operation creates a fixed number of configured parallel branches, letting you create a static split-join that handles a fixed number of message requests. Each branch has its own Scope, which in turn can contain any number of operations.

A Parallel node is essentially a placeholder for a fixed number of processing branches, each with its own scope. Two branches are automatically generated when you add a Parallel operation to the flow. You can define the unique processing logic for each scope by dragging the appropriate operations into the scope. You can also add more branches with the Add Scope button.

Figure 10-3 Add Scope Button



To process a fixed number of requests in parallel:

1. Under Flow Control in the Components window, click **Parallel** and drag it onto the editor in the appropriate location in the flow.
2. To add more branches, click the Add Scope button in the upper right corner of the Parallel node.
3. For each branch, define the processing logic by dragging additional operations into the scope and then configuring them in the Properties window.

For information about the operations you can add, see [Adding Flow Control Operations in JDeveloper](#) and [Adding Assign Operations in JDeveloper](#).

4. To change the name of the Parallel node, click **Parallel** at the top of the node and enter a new name in the field that appears.
5. On the JDeveloper toolbar, click **Save**.

How to Define If-Else Conditional Logic

An If Activity provides conditional logic within a split-join. It is composed of a number of nodes that determine the behavior for the overall If activity. Each node

must be individually configured. When you create an If activity, an If branch and an Else branch are automatically generated within it.

Each If and Else If branch provides a unit of conditional logic (defined by an XPath Expression) within the overall If activity. Those branches also define the processing logic to carry out when the conditions are met. The Else branch defines the processing logic to carry out when the conditions are not met. You can add an unlimited number of Else If nodes to the If operation.

To define if-else conditional logic:

1. Under Flow Control in the Components window, click If and drag it onto the editor in the appropriate location in the flow.

An If node appears with an If branch and an Else branch.

2. To configure the If branch, do the following:

- a. In the new If node, select the If branch.
- b. In the Properties window, click the Condition tab.
- c. Next to the **Condition** field, click the Expression Builder icon to launch the XPath Expression Builder and define an expression that defines the If condition.
- d. To define the processing that occurs if the If condition is met, drag operations from the Components window to the If branch and then configure them in the Properties window.

For information about the operations you can add, see [Adding Flow Control Operations in JDeveloper](#) and [Adding Assign Operations in JDeveloper](#).

3. To add an Else If branch, do the following:

- a. Click the yellow diamond-shaped icon in the upper right corner of the If node, or right-click in the If node and select Add **Elseif**.

Figure 10-4 Add Else If Button



A new Else If branch appears to the left of the Else branch.

- b. Configure the Else If branch in the same way as the If branch, described in step 2.
4. To define the processing that occurs if none of the conditions are met, drag operations from the Components window to the Else branch and then configure them in the Properties window.
5. To change the name of the If node or any of the branches, select the node or branch icon, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.

6. In the **Description** field, enter any notes you think are important.
7. On the JDeveloper toolbar, click **Save**.

How to Create Error Handlers

An error handler receives and handles errors. An error handler can be attached to a Start Node or a Scope. When attached to a Start Node, it is a *global* error handler and serves as a catch-all for the output of all local Raise Error nodes. When attached to a Scope, it only handles errors raised locally.

To create an error handler:

1. Select the Start Node or Scope node to which you want to add the Error Handler.
2. Right-click the selected node and select **Add Catch** or **Add CatchAll**.
A new Error Handlers node appears to the right of the selected node.
3. To invoke a SOAP Fault for a Catch All handler, select the Catch All branch, click the Catch All tab on the Properties window, and enter the fault name in the **SOAP Fault Variable Name**.
4. To configure a Catch handler, select the Catch branch, click the Catch tab on the Properties window, and do one of the following:
 - To invoke a fault that you define, select **User-defined Fault** and click the **Edit** icon. On the Edit QName dialog, enter the fault's name and namespace.
 - To invoke a predefined fault, select **Pre-defined Fault** and select a fault from the list of options.
5. To configure additional processing for a Catch or Catch All branch before a response is sent, drag Assign, If, and/or Reply nodes from the Components window and configure them in the Properties window.
6. To change the name of the Error Handler node or any of the branches, select the node or branch icon, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.
7. In the **Description** field, enter any notes you think are important.
8. On the JDeveloper toolbar, click **Save**.

How to Raise an Error

The Raise Error generates an error that causes the split-join to stop normal processing. If the error is not handled using an Error Handler, the split-join will terminate and a Fault will be sent to the Service Bus message flow. Configuring a Raise Error can optionally include documenting the nature of the error in the General Information tab.

To raise an error:

1. Under Flow Control in the Components window, click **Raise Error** and drag it onto the editor in the appropriate location in the flow.
2. Select the new **Raise Error** node.
3. In the Properties window, click the Raise Error tab.

4. To invoke a fault that you define, select **User-defined Fault** and click the **Edit** icon. On the Edit QName dialog, enter the fault's name and namespace.
5. To invoke a predefined fault, select **Pre-defined Fault** and select a fault from the list of options.
6. To change the name of the Raise Error node, select the node or branch icon, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.
7. In the **Description** field, enter any notes you think are important.
8. On the JDeveloper toolbar, click **Save**.

How to Re-Raise an Error

You can add a Re-Raise Error operation to an error handler. This operation lets you re-raise an error caught by an error handler Catch or Catch All operation. This operation does not require any configuration, but you can modify the name and add a description.

To re-raise an error:

1. Under Flow Control in the Components window, click **Re-Raise Error** and drag it onto the editor in the appropriate location in the flow.
2. Select the new **Re-Raise Error** node.
3. To change the name of the Re-Raise Error node, enter a unique, identifying string in the **Name** field on the Properties window.
4. In the **Description** field, enter any notes you think are important.
5. On the JDeveloper toolbar, click **Save**.

How to Repeat an Operation Until it Evaluates to True

A Repeat Until node provides conditional logic within a split-join. When you define a Repeat Until node, the associated operations repeat until the condition you define evaluates to true. The associated operations are defined in the loop of the Repeat Until node. The condition is evaluated after each loop finishes.

To repeat an operation until it evaluates to true:

1. Under Flow Control in the Components window, click **Repeat Until** and drag it onto the editor in the appropriate location in the flow.

A Repeat Until node appears with a conditional branch.

2. In the new Repeat Until node, select the **Condition** icon.
3. In the Properties window, click the Condition tab.
4. Next to the **Condition** field, click the Expression Builder icon to launch the XPath Expression Builder and define an expression that must evaluate to true in order for the process to stop repeating.
5. To define the processing logic that is repeated until the condition evaluates to true, drag operations from the Components window to the Repeat Until node and then configure them in the Properties window.

For information about the operations you can add, see [Adding Flow Control Operations in JDeveloper](#) and [Adding Assign Operations in JDeveloper](#).

6. To change the name of the Repeat Until node, select the node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.
7. In the **Description** field, enter any notes you think are important.
8. On the JDeveloper toolbar, click **Save**.

How to Repeat an Operation Until it Evaluates to False

A While node provides conditional logic within a split-join. When you define a While node, the associated operations repeat until the condition you define evaluates to false. The associated operations are defined in the loop of the Repeat Until node. The condition is evaluated before each loop finishes.

To repeat an operation until it evaluates to false:

1. Under Flow Control in the Components window, click **While** and drag it onto the editor in the appropriate location in the flow.

A While node appears with a conditional branch.

2. In the new While node, select the **Condition** icon.
3. In the Properties window, click the Condition tab.
4. Next to the **Condition** field, click the Expression Builder icon to launch the XPath Expression Builder and define an expression that must evaluate to false in order for the process to stop repeating.
5. To define the processing logic that is repeated until the condition evaluates to false, drag operations from the Components window to the While node and then configure them in the Properties window.

For information about the operations you can add, see [Adding Flow Control Operations in JDeveloper](#) and [Adding Assign Operations in JDeveloper](#).

6. To change the name of the While node, select the node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.
7. In the **Description** field, enter any notes you think are important.
8. On the JDeveloper toolbar, click **Save**.

How to Insert a Pause in Processing

A Wait operation inserts a pause in the split-join flow for a short duration to wait for other dependent jobs to complete. After the specified duration is reached, the split-join execution resumes.

To insert a pause in processing:

1. Under Flow Control in the Components window, click **Wait** and drag it onto the editor in the appropriate location in the flow.

A While node appears with a conditional branch.

2. Select the new **Wait** node.

3. In the Properties window, click the Wait tab.
4. Next to the **Duration** field, click the Expression Builder icon to launch the XPath Expression Builder and define an expression that evaluates to a duration type of `xsd:duration` in the following format:

PnYnMnDTnHnMnS (number of years, months, days, hours, minutes, and seconds, with a date/time separator, represented by "T".)
5. To change the name of the Wait node, select the node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.
6. In the **Description** field, enter any notes you think are important.
7. On the JDeveloper toolbar, click **Save**.

Adding Assign Operations in JDeveloper

Assign operations include Assign, Copy, Delete, Insert, Java Callout, Log, and Replace. Every Assign is composed of one or more of these operations, which you can add to the Assign using the Design view. Below are the assign operations you can use in a split-join:

- **Assign:** Assigns the result of an XQuery or XSLT expression to a variable.
- **Copy:** Copies the information specified by an XPath expression from a source document to a destination document.
- **Delete:** Deletes a set of nodes specified by an XPath expression.

Note:

Unlike the Service Bus delete, only an XPath expression may be deleted in a split-join, not the entire variable.

- **Insert:** Inserts the result of an XQuery Expression at an identified place relative to nodes selected by an XPath expression.
- **Java Callout:** Invokes a Java method for processing such as validation, transformation, and logging.
- **Log:** Logs split-join data at a specified severity to the server log file.
- **Replace:** Replaces a node or the contents of a node specified by an XPath expression.

About Transformations and Expressions in Assign Operations

You can use a variety of XQuery and XSLT resources to define the transformations and expressions that derive values for the assign operations. Most assign operations support the following methods. You can find more information in the links given below.

- [Transforming Data with XQuery](#)
- [Transforming Data with XSLT](#)

- [Working With Expression Editors in Oracle Service Bus Console](#)
- "Creating Transformations with the XQuery Mapper" in *Developing SOA Applications with Oracle SOA Suite*
- "Creating Transformations with the XSLT Map Editor" in *Developing SOA Applications with Oracle SOA Suite*

Note:

The Assign operations in the split-join editor are similar to the corresponding pipeline actions. However, one important difference is that when you are using the XQuery, XSLT, or XPath Editors to edit expressions in the split-join context, only variables and namespaces internal to the split-join are available.

Assign Operation Expression Resolution

The assign functionality in split-joins conforms to the WS-BPEL specification for resolution of XPath, XQuery, and XSLT expressions to simple type variables. Supported simple types for binding expressions to variables in split-joins are String, Boolean, and Float. The Assign operation converts the value you provide to the type with which the variable is defined.

For example:

- If you assign `<order><number>4</number></order>` to a response variable defined as a String (`$response.result`), Service Bus returns `<number>4</number>` as a String in the result through a simple copy of the child element and value.
- If you map `<order><number>4</number></order>` to a String variable (such as `myStr`), then assign `$myStr` to `$response.result`, Service Bus returns `<result>4</result>`, because it first converts the value in `$myStr` to a String before it makes the assignment to the `$response.result` String variable.

How to Assign a Value to a Variable

Use an Assign to manipulate data by initializing and updating a variable using XSLT or XQuery expressions or resources. You can also use dynamic XSLT or XQuery.

When Service Bus binds variables in an inline XQuery, it assumes the type is `xs:string`. This can cause parser errors in operations with constants that are incompatible with `xs:string`. To ensure compatible types, use an explicit XQuery cast. For example, the following inline XQuery will fail. Although the `$itemsTotal` is of type `xs:double`, it is bound as an `xs:string`, which is incompatible in the test against 10000.

```
if ($itemsTotal < 10000) then . . .
```

To make this inline XQuery work, explicitly cast the `$itemsTotal` to an `xs:double`:

```
if (($itemsTotal cast as xs:double) < 10000) then . . .
```

When creating an assign operation to a String result or variable, make sure your expression returns a String value. Assigning a non-String value to a String result or String global variable does not cause a `MismatchedAssignmentFailure` exception, as specified by the WS-BPEL specification.

To assign a value to a variable:

1. Under Assign Operations in the Components window, click **Assign** and drag it onto the editor in the location in the flow where you want to update the variable.
2. Select the new **Assign** node.
3. In the Properties window, click the Assign tab.
4. Next to the **Value** field, select the Expression Builder icon or click the down arrow next to the icon to select the type of transformation you want to use to derive the variable value.

The expression builder for the transformation you selected appears.

5. Define the transformation or select a resource to use. For more information, see [About Transformations and Expressions in Assign Operations](#).

Note:

When you select XQuery or XSLT resources that are not already included in the current project, you need to import them. When you select the resource from the expression builder, an import dialog automatically appears. For more information, see [How to Import Resources in JDeveloper](#).

6. To specify the variable to which you are assigning the value, do one of the following:
 - To select an existing variable, select the variable name from the list of available options in the **Variable** field.
 - To create a new variable, click **Create Variable** next to the **Variable** field. Complete the Create Variable Alias dialog, as described in ["To define global and local variables:"](#).
7. In the QoS field, select either **Best Effort** (not executed in the context of a transaction) or **Exactly Once** (executed in a transaction).
8. To change the name of the Assign node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.

The new name appears beneath the node in the Split-Join Definition Editor.

9. In the **Description** field, enter any notes that you think are important.
10. On the JDeveloper toolbar, click **Save**.

How to Copy a Value from a Source to a Destination Document

The Copy operation lets you copy the information specified by an XPath expression or literal value from a source document to a destination document. It is an operation unique to the split-join editor.

To copy a value:

1. Under Assign Operations in the Components window, click **Copy** and drag it onto the editor in the location in the flow where you want to copy the data.

2. Select the new **Copy** node.
3. In the Properties window, click the Copy tab.
4. To use the existing element name in the destination to hold the copied value, select **Keep Source Element**.

If this option is not selected, the name of the source element in the destination is used to hold the copied value.
5. In the **From: Type** field, select the type of element to copy from.
6. Do one of the following:
 - If you selected Expression, Variable, or XML Fragment, click the Expression Builder icon by the **Value** field. Define the expression in the dialog that appears.
 - If you selected Literal, enter the literal string to copy in the **Value** field.
7. In the **To: Type** field, select the type of element to copy to.
8. Click the Expression Builder icon below the **Value** or **Query** field. Define the expression in the dialog that appears.
9. To change the name of the Copy node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.

The new name appears beneath the node in the Split-Join Definition Editor.
10. In the **Description** field, enter any notes that you think are important.
11. On the JDeveloper toolbar, click **Save**.

How to Delete a Set of Nodes

The Delete operation lets you delete a set of nodes specified by an XPath expression.

Note:

Unlike the Service Bus delete, only an XPath expression may be deleted in a split-join, not the entire variable.

To delete a set of nodes:

1. Under Assign Operations in the Components window, click **Delete** and drag it onto the editor in the location in the flow where you want to delete the nodes.
2. Select the new **Delete** node.
3. In the Properties window, click the Delete tab.
4. In the **Location** field, select the variable on which the XPath expression is executed to select the nodes to be deleted.
5. Click the Expression Builder icon to launch the expression editor and define an XPath expression that specifies the nodes to be deleted.
6. In the QoS field, select either **Best Effort** (not executed in the context of a transaction) or **Exactly Once** (executed in a transaction).

7. To change the name of the Delete node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.

The new name appears beneath the node in the Split-Join Definition Editor.

8. In the **Description** field, enter any notes that you think are important.
9. On the JDeveloper toolbar, click **Save**.

How to Insert the Result of an XQuery Expression

The Insert operation inserts the result of an XQuery expression at an identified place relative to nodes selected by an XPath expression.

To insert the result of an XQuery expression:

1. Under Assign Operations in the Components window, click **Insert** and drag it onto the editor in the location in the flow where you want to insert the result.
2. Select the new **Insert** node.
3. In the Properties window, click the Insert tab.
4. Next to the **Value** field, select the Expression Builder icon or click the down arrow next to the icon to select the type of transformation you want to use to create the data that to insert at a specified location in a variable.

The expression builder for the transformation you selected appears.

5. In the expression builder, define the transformation or select a resource to use. For more information, see [About Transformations and Expressions in Assign Operations](#).

Note:

When you select XQuery or XSLT resources that are not already included in the current project, you need to import them. When you select the resource from the expression builder, an import dialog automatically appears. For more information, see [How to Import Resources in JDeveloper](#).

6. In the **Position** field, select the position where you want to insert the data. Select from before, after, as first child of, or as last child of.
7. In the **Location** field, select the variable into which you want to insert the data, or click **Create Variable** to add a new variable.
8. Click the Expression Builder icon below the **Location** field to launch the XPath Expression Builder, and define an expression that specifies the nodes to be selected.
9. In the QoS field, select either **Best Effort** (not executed in the context of a transaction) or **Exactly Once** (executed in a transaction).
10. To change the name of the Insert node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.

The new name appears beneath the node in the Split-Join Definition Editor.

11. In the **Description** field, enter any notes that you think are important.

12. On the JDeveloper toolbar, click **Save**.

How to Invoke a Java Method in a Split-Join

A Java callout operation lets you invoke a static Java method from a split-join for custom operations to be handled in Java such as validation, transformation, and logging.

To invoke a Java method in a split-join:

1. Under Assign Operations in the Components window, click **Java Callout** and drag it onto the editor in the location in the flow where you want to use the callout.
2. Select the new **Java Callout** node.
3. In the Properties window, click the Java Callout tab.
4. Next to the **Method** field, click **Browse**, navigate to and select the JAR file that contains the method to invoke, and then select the method.

Any arguments for the method appears in the Arguments table.

5. In the **Value** column, click the Expression Builder icon, or click the down arrow next to the icon to select a transformation type.

The expression builder for the transformation you selected appears.

6. Define an expression to map data to the input parameters of the static Java method. For more information, see [About Transformations and Expressions in Assign Operations](#).

Note:

When you select XQuery or XSLT resources that are not already included in the current project, you need to import them. When you select the resource from the expression builder, an import dialog automatically appears. For more information, see [How to Import Resources in JDeveloper](#).

7. In the **Return** field, select the variable to contain the result value for the Java method from the list of options. If the variable does not exist, click **Create Variable** to add a new one.
8. In the **Service Account** field, click **Browse** to select a service account to use to put the appropriate subject on the thread when executing the Java callout.
9. In the QoS field, select either **Best Effort** (not executed in the context of a transaction) or **Exactly Once** (executed in a transaction).
10. To change the name of the Java Callout node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.

The new name appears beneath the node in the Split-Join Definition Editor.

11. In the **Description** field, enter any notes that you think are important.
12. On the JDeveloper toolbar, click **Save**.

How to Log Split-Join Data

A log operation lets you log split-join data of a specified severity to the server log file. Administrators can use log information to take appropriate actions based on the severity of the data logged.

To log split-join data:

1. Under Assign Operations in the Components window, click **Log** and drag it onto the editor in the location in the flow where you want to create log entries.
2. Select the new **Log** node.
3. In the Properties window, click the Log tab.
4. Next to the **Content** field, click the Expression Builder icon, or click the down arrow next to the icon to select a transformation type.

The expression builder for the transformation you selected appears.

5. Define an expression to select the data to be logged. For more information, see [About Transformations and Expressions in Assign Operations](#).

Note:

When you select XQuery or XSLT resources that are not already included in the current project, you need to import them. When you select the resource from the expression builder, an import dialog automatically appears. For more information, see [How to Import Resources in JDeveloper](#).

6. In the **Summary** field, specify a note for the log. The annotation is logged along with the data selected by the expression
7. In the **Severity** field, select one of the following the severity levels for the log:
 - Debug
 - Info
 - Warning
 - Error
8. To change the name of the Log node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.

The new name appears beneath the node in the Split-Join Definition Editor.

9. In the **Description** field, enter any notes that you think are important.
10. On the JDeveloper toolbar, click **Save**.

How to Replace a Node or Its Contents

A Replace operation replaces a node or the contents of a node specified by an XPath expression.

To replace a node or its contents:

1. Under Assign Operations in the Components window, click **Replace** and drag it onto the editor in the location in the flow where you want to replace data.
2. Select the new **Replace** node.
3. In the Properties window, click the Replace tab.
4. In the **Location** field, select the variable containing the data to be replaced, or click **Create Variable** to add a new variable.
5. Click the Expression Builder icon below the **Location** field to launch the XPath Expression Builder, and define an expression that specifies the nodes to be replaced.
6. Next to the **Value** field, click the Expression Builder icon, or click the down arrow next to the icon to select a transformation type.

The expression builder for the transformation you selected appears.

7. Define an expression to select the data to be replaced. For more information, see [About Transformations and Expressions in Assign Operations](#).

Note:

When you select XQuery or XSLT resources that are not already included in the current project, you need to import them. When you select the resource from the expression builder, an import dialog automatically appears. For more information, see [How to Import Resources in JDeveloper](#).

8. In the **Replace** field, select **Entire Node** to replace the entire node or select **Node Contents** to replace just the content of the node.
9. In the QoS field, select either **Best Effort** (not executed in the context of a transaction) or **Exactly Once** (executed in a transaction).
10. To change the name of the Log node, click the General tab of the Properties window, and enter a unique, identifying string in the **Name** field.

The new name appears beneath the node in the Split-Join Definition Editor.

11. In the **Description** field, enter any notes that you think are important.
12. On the JDeveloper toolbar, click **Save**.

Working with Split-Joins in the Oracle Service Bus Console

You create and configure split-joins in JDeveloper, but you can import split-joins into the Oracle Service Bus Console to add them to projects in the console. You can also import them by importing the entire project. From the console, you can specify dispatch policies, update the XQuery version, and define SLA alerts.

How to Import a Split-Join into the Console

Before you can work with a split-join in the console, you need to export it from JDeveloper and then import it into the console. Once the resource is in the console, you can open it in the Split-Join Definition Editor to configure certain properties and define

SLA alert rules. For information about exporting and importing resources, see [Importing and Exporting Resources in the Oracle Service Bus Console](#).

How to Configure Split-Joins in the Console

You can only configure certain properties of split-joins in the Oracle Service Bus Console; you cannot configure the processing logic of the split-join.

To configure split-joins in the Console:

1. In the Project Navigator, navigate to the split-join you want to configure.
2. Right-click the split-join and select **Open**.
The split-join appears in the Split-Join Definition Editor.
3. In the **Description** field, enter any important notes about the split-join.
4. In the **Dispatch Policy** field, select the WebLogic Server Work Manager to use to execute the split-join.
5. In the **Version for snippets** field, select the version of XQuery for processing XQuery in the split-join.
6. Click **Save**.

How to Define Service Level Agreement Rules for a Split-Join

Service level agreement (SLA) alerts let system administrators know when certain conditions are met that indicate the health of a split-join. You can define alerts based on statistics such as error counts, message counts, elapsed time, and failure or success ratios for message processing. For more information about defining SLA alerts, see "Creating Service Level Agreement Alert Rules" in *Administering Oracle Service Bus*.

Static and Dynamic Split-Join Samples

This section describes how to create static and dynamic split-joins using sample scenarios.

- [Designing a Static Split-Join](#)
- [Designing a Dynamic Split-Join](#)

Designing a Static Split-Join

This scenario creates a new split-join called *Service Availability* that handles orders for a telco's cable service package including TV, phone, and internet service. The idea is for the split-join to receive an incoming package order and to reply with an order acknowledgment for each type of service. In this case, *Service Availability* is designed as a *Static* split-join because there are three message requests per order, one for each type of service. In this particular example the customer requests only TV and DSL service.

Creating the *Service Availability* split-join may include the following tasks:

[Creating a New Split-Join](#)

[Adding an Assign](#)

[Adding a Parallel Node](#)

[Adding an Assign for Each Branch](#)

[Adding an Invoke Service](#)

[Adding an Assign for Each Branch](#)

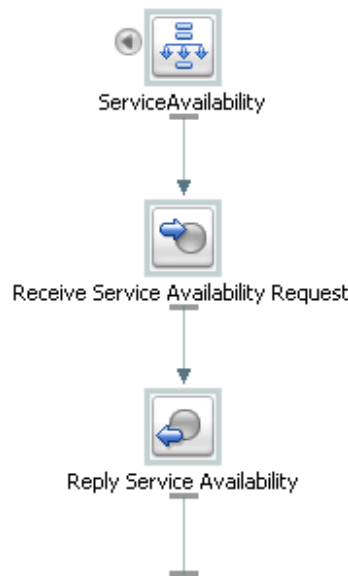
[Exporting and Testing the Split-Join](#)

Creating a New Split-Join

Create a new split-join based on the WSDL operation you want to use for placing the order. In this case the WSDL operation we want is called "telecom."

After you select the WSDL operation, a skeleton of the newly created split-join appears in the split-join editor, as shown in the following figure. It consists of a Start Node, a Receive, a Reply. Edit the labels in the Properties window to better reflect the specific function of each node in this particular split-join.

Figure 10-5 New Split-Join



The Start Node contains both a Request variable and a Response variable that were determined by the WSDL operation initially selected. The Receive receives the incoming request message (in this case for the three or fewer different kinds of cable service), and the Reply generates a response message and sends it back to the client.

Note:

The Receive node requires no further configuration. Similarly the Reply requires no further configuration, unless to generate an error fault, which is not the case in this scenario.

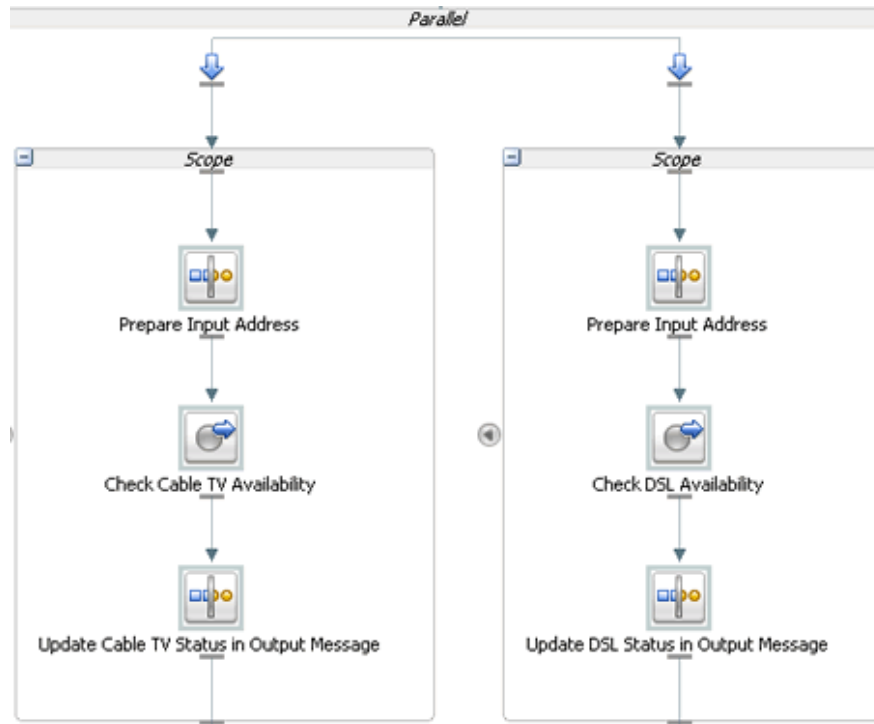
Adding an Assign

The first Assign, *Prepare Output Message*, contains an Assign operation that prepares the Response variable in a form such that the later nodes can work on the data within it. This output message is relayed to the final Reply node in the split-join and, in turn, returned to the original client.

Adding a Parallel Node

The Parallel node contains two main branches, one to check cable TV availability and one to check DSL availability. Each branch is composed of a number of actions, the sequence and general configuration being the same for both branches.

Figure 10-6 *Parallel Node*



Adding an Assign for Each Branch

The first Assign in each Parallel branch, *Prepare Input Address*, copies the incoming customer address data into a Variable that is referenced to check the availability of the service at that location. The Assigns are the same for each branch and would be for additional branches as well.

Adding an Invoke Service

An External Service is then invoked to check whether the requested service type is available at the customer's location. Each branch contains one Invoke Service, *Check Cable TV Availability* and *Check DSL Availability*. Each invocation calls an External Service, which compares the customer's address (stored in the Variable initialized in the previous step) to the availability of the service at that location. The result is then stored in an output Variable that is passed on to the final Assign in the Branch below.

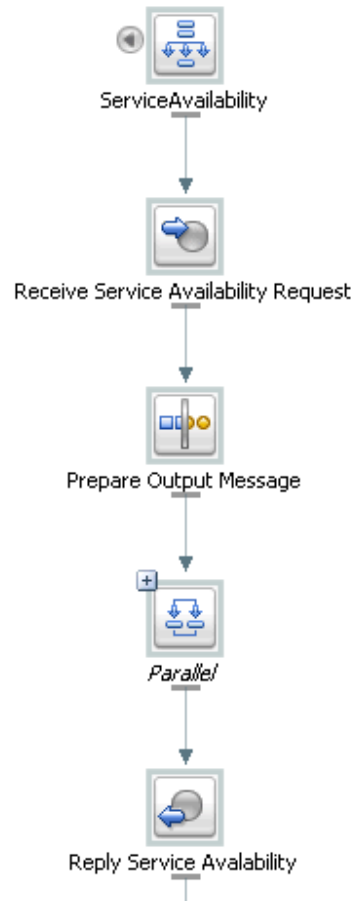
Adding an Assign for Each Branch

The final two Assigns, *Update Cable TV Status in Output Message* and *Update DSL Status in Output Message*, take the results of the external service invocations and put them into the output message using a Replace operation. The aggregated response are then sent to the original client in the final Reply node, which requires no further configuration.

Exporting and Testing the Split-Join

After you design the split-join, you can export it to the Oracle Service Bus Console for testing and production.

Figure 10-7 Completed Split-Join Ready for Testing



Related Topics

- [How to Create a Split-Join in JDeveloper](#)
- [How to Configure the Start Node](#)
- [How to Assign a Value to a Variable](#)
- [How to Invoke a Service](#)
- [How to Process a Fixed Number of Requests in Parallel](#)

Designing a Dynamic Split-Join

This scenario illustrates a split-join that handles a batch order from a retailer containing a variable number of individual purchase orders (as opposed to a fixed number of orders). The idea is for the split-join to receive the batch order and to reply with an order acknowledgment for each order within. This would be a *Dynamic* split-join because the number of individual purchase order requests is variable and unknown at design time.

Creating this split-join may include the following tasks:

[Creating a New Split-Join](#)

[Adding an Assign](#)

[Adding a For Each](#)

[Adding an Assign](#)

[Adding an Invoke Service](#)

[Adding an Assign](#)

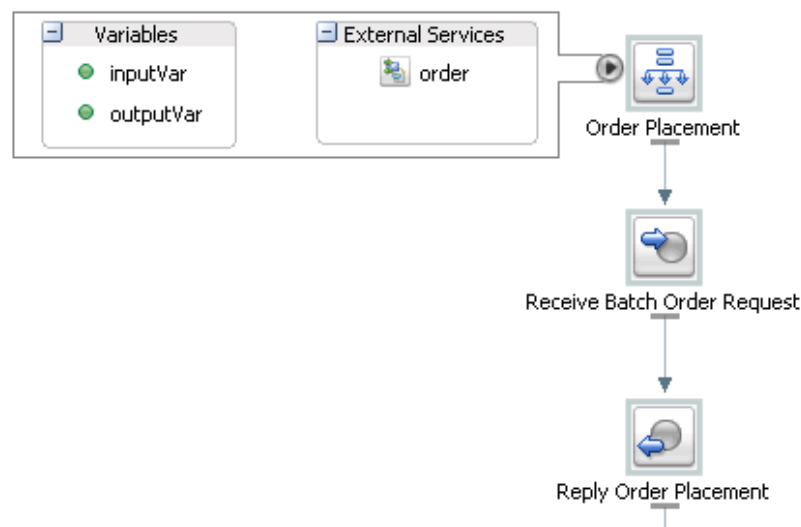
[Adding an Error Handler](#)

[Exporting and Testing the Split-Join](#)

Creating a New Split-Join

Create a new split-join based off of the WSDL operation you want to use for placing the order. In this case the WSDL operation we want is called "batchOrders." After the operation is selected, a skeleton of the newly created split-join appears in the split-join editor consisting of a Start Node, a Receive, a Reply. The labels are then edited in the general properties tab to better reflect the specific function of each node in this particular split-join.

Figure 10-8 *New Split-Join With Edited Labels*



The Start Node, *Order Placement*, contains both a request variable, *inputVar*, and an response variable, *outputVar*. The Receive, *Receive Batch Order Request*, will initialize the contents of the Request Variable (in this case purchase orders), and the Reply, *Reply Order Placement*, will send a response, based on the order acknowledgments aggregated into the Response Variable, back to the client. In this example *Order Placement* also contains a callout to an External Service, "Order" that will be invoked to approve individual orders.

Note:

The Receive node requires no further configuration. Similarly, the Reply requires no further configuration, unless you would like to generate an error fault—which is not the case in this scenario (see [How to Configure a Reply](#) for more information on generating faults).

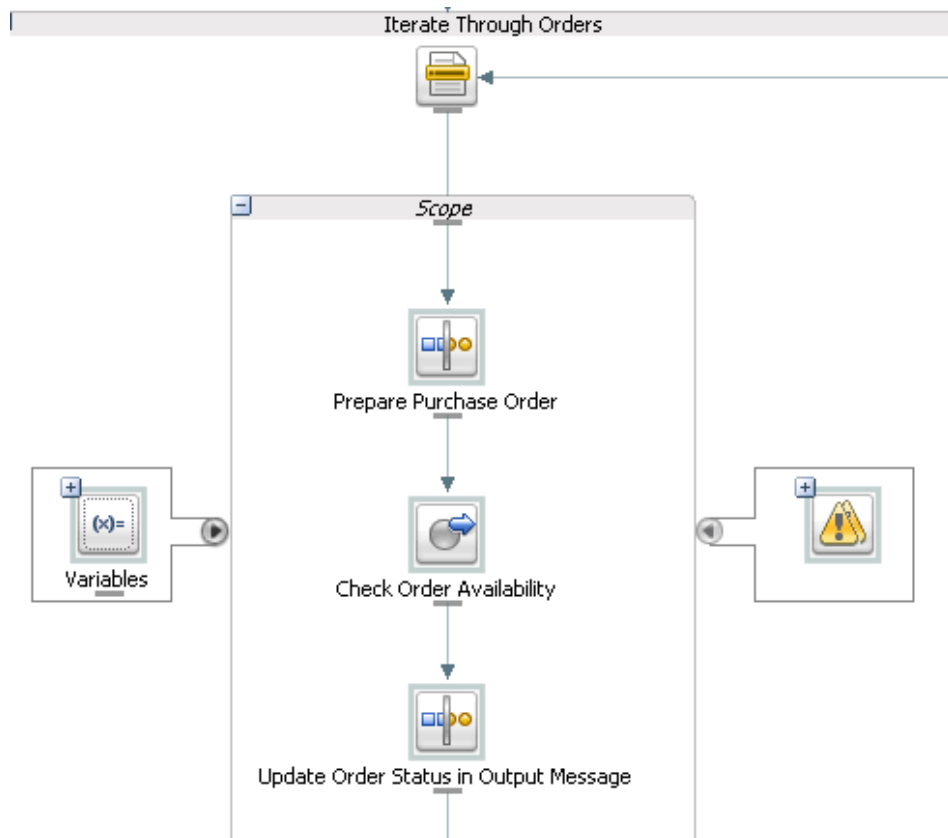
Adding an Assign

The first Assign, *Prepare Output Message*, contains an Assign operation that prepares the response variable (here labeled an "Output Message" for readability) in a form such that the later nodes can work on the data captured within it (that is, Copy/Insert/Assign/Replace/Delete into the Variable). In this case, that data would consist of order acknowledgments and/or errors. This Output Message is relayed to the final Reply node in the split-join and, in turn, returned to the original client.

Adding a For Each

The For Each, *Iterate Through Orders*, contains logic that will parse through each order in the batch, send it to an external proxy for approval, and capture an order acknowledgment in response. If there is a problem with an order, an error is sent from the invoked proxy and captured in the Error Handler. The following figure depicts the entire scope of the For Each logic.

Figure 10-9 For Each Node Labeled "Iterate Through Orders"



Adding an Assign

The Assign, *Prepare Purchase Order*, copies the incoming purchase order requests into a variable that is referenced to check approval of the order in the next step.

Adding an Invoke Service

An external service, *Check Order Availability*, is then invoked to approve each individual purchase order. If the order is accepted, the service responds with an order acknowledgment. If the order is not accepted, the service responds with an error. The result is then stored in an output variable that is passed on to the final Assign in the next step.

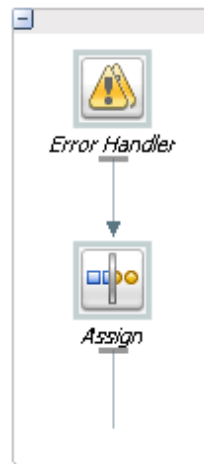
Adding an Assign

The final Assign, *Update Order Status in Output Message*, takes the results of the external service invocation and copies them into the output message using an Insert operation. The aggregated response is then sent to the original client in the final Reply node, which requires no further configuration.

Adding an Error Handler

The Error Handler captures any Errors returned by the invoked service. It takes these errors and inserts them into the output message using an Assign operation.

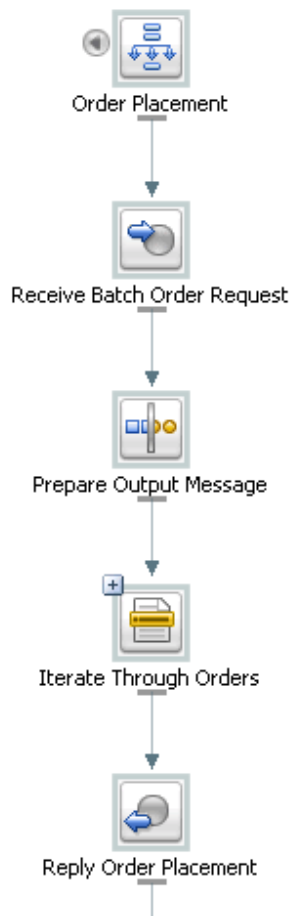
Figure 10-10 Error Handler



Exporting and Testing the Split-Join

After you design the split-join, you can export it to the Oracle Service Bus Console for testing and production.

Figure 10-11 *Completed Split-Join Ready for Testing*



Working with WSDL Documents

This chapter describes Web Service Definition Language (WSDL) documents and how you can use them in Service Bus projects to generate business and proxy services, pipelines, and split-joins. A WSDL document is the formal description of a web service, defining what the service can do, where it resides, and how to invoke it.

This chapter contains the following sections:

- [WSDL Overview](#)
- [WSDL Documents in Service Bus](#)
- [Services Based on WSDL Ports and on WSDL Bindings](#)
- [Importing and Exporting WSDL Resources](#)
- [Working with WSDL Documents in JDeveloper](#)
- [Working with WSDL Documents in the Oracle Service Bus Console](#)
- [Viewing Effective WSDL Documents](#)

WSDL Overview

A WSDL document describes a service, its location, its operations, and the way in which clients can communicate with it. This section provides a very brief introduction to WSDL, to provide context for discussing Service Bus features.

[Table 11-1](#) summarizes the main elements used to define WSDL services.

Table 11-1 High-level WSDL Elements

Element	Description
types	Type definitions for message content.
message	Abstract definition of the data being exchanged. A message consists of parts, which describe the logical, abstract content of the message.
portType	Abstract collection of operations supported by the service.
operation	Abstract description of an action supported by the service.
binding	Concrete protocol and data format specification for a port type.
port	A single endpoint, consisting of a network address and a binding.

Table 11-1 (Cont.) High-level WSDL Elements

Element	Description
service	Collection of related ports, or endpoints.

WSDL specifies SOAP, HTTP, MIME, and Service Bus-specific binding extensions, which extend the WSDL binding mechanism to support features specific to the protocol or message format.

WSDL Types

The `types` element is a container for data type definitions. It uses a type system, such as XML Schema (XSD), to define the vocabulary of messages handled by this service. For example, a service that provides stock quotes might define an XML vocabulary, with the terms `TradePriceRequest` and `TradePrice`, as shown in the following example.

Example -WSDL Types Example

```
<types>
  <schema targetNamespace="http://example.com/stockquote.xsd"
    xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="TradePriceRequest">
      <complexType>
        <all>
          <element name="tickerSymbol" type="string"/>
        </all>
      </complexType>
    </element>
    <element name="TradePrice">
      <complexType>
        <all>
          <element name="price" type="float"/>
        </all>
      </complexType>
    </element>
  </schema>
</types>
```

WSDL Messages

The `message` element provides an abstract, typed definition of the data being communicated. A message consists of parts, each of which describes one logical, abstract unit of the message. A WSDL document can define one or more messages, each of which may have one or more parts. For example, the WSDL fragment in the following example defines four message types, `sellerInfoMessage`, `buyerInfoMessage`, `response`, and `negotiationMessage`, each of which has one or more parts.

Example - WSDL Message Example

```
<message name="sellerInfoMessage">
  <part name="inventoryItem" type="xsd:string"/>
  <part name="askingPrice" type="xsd:integer"/>
</message>

<message name="buyerInfoMessage">
```

```

    <part name="item" type="xsd:string"/>
    <part name="offer" type="xsd:integer"/>
</message>

<message name="response">
  <part name="result" type="xsd:string"/>
</message>

<message name="negotiationMessage">
  <part name="item" type="xsd:string"/>
  <part name="price" type="xsd:integer"/>
  <part name="outcome" type="xsd:string"/>
</message>

```

WSDL Port Types

The `portType` element defines a set of operations supported by one or more endpoints, which are defined in the `port` element (see [WSDL Services and Ports](#)). The port type provides the public interface for the operations provided by the service. Each operation is defined in an `<operation>` element, each of which is an abstract description of an action supported by the service.

For example, the following example defines a port type with one operation, `GetLastTradePrice`, which can handle an input message, `GetLastTradePriceInput`, and an output message, `GetLastTradePriceOutput`. The concrete descriptions of these messages are defined in the WSDL binding, as shown in the `soap:operation` subelement in the following example.

Example - WSDL Port Type and Operation Example

```

<portType name="StockQuotePortType">
  <operation name="GetLastTradePrice">
    <input message="tns:GetLastTradePriceInput"/>
    <output message="tns:GetLastTradePriceOutput"/>
  </operation>
</portType>

```

WSDL Bindings

The `binding` element specifies a concrete data format specification and a concrete transport protocol for a port type. The following example specifies the binding for the `StockQuotePortType` port type, which is provided as the value for the `type` attribute. The `soap:binding` subelement signifies that the binding is bound to the SOAP protocol format. In that subelement, the `style` attribute specifies that the data format is SOAP document style, and the `transport` attribute specifies that the transport protocol is HTTP.

Example - WSDL Binding Example

```

<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
    <soap:operation
      soapAction="http://example.com/GetLastTradePrice"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>

```

```
    </operation>  
</binding>
```

WSDL Services and Ports

The `service` element defines a collection of related endpoints, each of which is defined in a child `port` element. A port is defined as a binding associated with a network address. The following example defines two ports, `StockQuotePort`, and `StockQuotePortUK`. They both use the same binding, `tns:StockQuoteSoapBinding`, which is concretely defined in `binding`, but they have different network addresses: `http://example.com/stockquote` and `http://example.uk/stockquote`. These are alternative ports available for this service.

Example - WSDL service and port Example

```
<service name="StockQuoteService">  
  <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">  
    <soap:address location="http://example.com:9999/stockquote"/>  
  </port>  
  <port name="StockQuotePortUK" binding="tns:StockQuoteSoapBinding">  
    <soap:address location="http://example.uk:9999/stockquote"/>  
  </port>  
</service>
```

WSDL Documents in Service Bus

In Service Bus, a WSDL document describes a proxy or business service, pipeline, or split-join. You can base SOAP and XML services on an existing WSDL resource. Service Bus defines some types of business services, proxy services, and pipelines using a WSDL document, an XML-based specification for describing web services. All split-joins are based on a WSDL document. A WSDL document describes service operations, input and output parameters, and how a client application connects to the service. For the WSDL 1.1 specification, see the W3C Note, "W3C Web Services Description Language (WSDL) 1.1," at <http://www.w3.org/TR/wSDL>.

Oracle Service Bus defines proxy services and business services in terms of two WSDL entities:

- The abstract WSDL interface, which defines the operations in that interface and the types of message parts in the operation signature.
- The binding WSDL interface, which defines the binding of the message parts to the message (packaging), and the binding of the message to the transport.

Web Service Types

If a service has a well-defined WSDL interface, it is recommended, although not required, that you use the WSDL document to define the service. There are three types of WSDL documents you can define:

- [SOAP Document Wrapped Web Services](#)
- [SOAP Document Style Web Services](#)
- [SOAP RPC Web Services](#)

SOAP Document Wrapped Web Services

A document wrapped web service is described in a WSDL file as a Document Style Service. However, it follows some additional conventions. Standard document-oriented web service operations take only one parameter or message part, typically an XML document. This means that the methods that implement the operations must also have only one parameter. Document-wrapped web service operations, however, can take any number of parameters, although the parameter values will be wrapped into one complex data type in a SOAP message. This wrapped complex data type is described in the WSDL document as the single document for the operation.

SOAP Document Style Web Services

You can configure Service Bus services as SOAP-style services. The following example provides an example of a WSDL for a sample document style web service using SOAP 1.1.

Example - WSDL for a Sample Document Style Web Service

```
<definitions name="Lookup"
targetNamespace="http://example.com/lookup/service/defs"
xmlns:tns="http://example.com/lookup/service/defs"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:docs="http://example.com/lookup/docs"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xs:schema targetNamespace="http://example.com/lookup/docs"
      elementFormDefault="qualified">
      <xs:element name="PurchaseOrg" type="xs:string"/>
      <xs:element name="LegacyBoolean" type="xs:boolean"/>
    </xs:schema>
  </types>
  <message name="lookupReq">
    <part name="request" element="docs:purchaseorg"/>
  </message>
  <message name="lookupResp">
    <part name="result" element="docs:legacyboolean"/>
  </message>
  <portType name="LookupPortType">
    <operation name="lookup">
      <input message="tns:lookupReq"/>
      <output message="tns:lookupResp"/>
    </operation>
  </portType>
  <binding name="LookupBinding" type="tns:lookupPortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="lookup">
      <soap:operation/>
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
</definitions>
```

The service has an operation (equivalent to a method in a Java class) called `lookup`. The binding indicates that this is a SOAP 1.1 document style web service.

When the WSDL document shown above is used for a request, the value of the body variable (`$body`) that the document style proxy service obtains is displayed in the following example.

Note:

Namespace declarations have been removed from the XML in the listings that follow for the sake of clarity.

Example - Body Variable Value

```
<soap-env:body>
  <req:purchaseorg>Oracle</req:purchaseorg>
</soap-env:body>
```

In the previous example, `soap-env` is the predefined SOAP 1.1 namespace and `req` is the namespace of the `PurchaseOrg` element (`http://example.com/lookup/docs`).

If the business service to which the proxy service is routing uses the above WSDL file, the value for the body variable (`$body`) given above is the value of the body variable (`$body`) from the proxy service.

The value of the body variable (`$body`) for the response from the invoked business service that the proxy service receives is displayed in the following example.

Note:

Namespace declarations have been removed from the XML in the listings that follow for the sake of clarity.

Example - Body Variable Value

```
<soap-env:body>
  <req:legacyboolean>true</req:legacyboolean>
</soap-env:body>
```

This is also the value of the body variable (`$body`) for the response returned by the proxy service using this WSDL file.

There are many tools available that take the WSDL file of a proxy service (obtained by adding the `?WSDL` suffix to the URL of the proxy service in the browser) and generate a Java class with the appropriate request and response parameters to invoke the operations of the service. This Java class can be used to invoke the proxy service that uses this WSDL file.

SOAP RPC Web Services

You can configure proxy services and business services as RPC-style services. The following example provides an example of a WSDL file for a sample RPC style web service using SOAP 1.1.

Example - WSDL File for a Sample RPC Style Web Service

```

<definitions name="Lookup"
targetNamespace="http://example.com/lookup/service/defs"
xmlns:tns="http://example.com/lookup/service/defs"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:docs="http://example.com/lookup/docs"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xs:schema targetNamespace="http://example.com/lookup/docs"
      elementFormDefault="qualified">
      <xs:complexType name="RequestDoc">
        <xs:sequence>
          <xs:element name="PurchaseOrg" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="ResponseDoc">
        <xs:sequence>
          <xs:element name="LegacyBoolean" type="xs:boolean"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </types>
  <message name="lookupReq">
    <part name="request" type="docs: RequestDoc"/>
  </message>
  <message name="lookupResp">
    <part name="result" type="docs: ResponseDoc"/>
  </message>
  <portType name="LookupPortType">
    <operation name="lookup">
      <input message="tns:lookupReq"/>
      <output message="tns:lookupResp"/>
    </operation>
  </portType>
  <binding name="LookupBinding" type="tns:lookupPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="lookup">
      <soap:operation/>
      <input>
        <soap:body use="literal" namespace="http://example.com/lookup/service"/>
      </input>
      <output>
        <soap:body use="literal" namespace="http://example.com/lookup/service"/>
      </output>
    </operation>
  </binding>
</definitions>

```

The service described in the preceding listing includes an operation (equivalent to a method in a Java class) called `lookup`. The binding indicates that this is a SOAP RPC web service. In other words, the web service's operation receives a set of request parameters and returns a set of response parameters. The `lookup` operation has a parameter called `request` and a return parameter called `result`. The namespace of the operation in the binding is:

```
http://example.com/lookup/service/defs
```

When the WSDL document shown in the previous WSDL File for a sample RPC Style Web Service Example is used for a request, the value of the body variable (`$body`) that the SOAP RPC proxy service obtains is displayed in the following example.

Note:

Namespace declarations have been removed from the XML in the listings that follow for the sake of clarity.

Example - Body Variable Value

```
<soap-env:body>
  <ns:lookup>
    <request>
      <req:purchaseorg>Oracle</req:purchaseorg>
    </request>
  </ns:lookup>
</soap-env:body>
```

In the above, `soap-env` is the predefined SOAP 1.1 name space, `ns` is the operation namespace (`http://example.com/lookup/service`) and, `req` is the namespace of the `PurchaseOrg` element (`http://example.com/lookup/docs`).

If the business service to which the proxy service routes the messages uses the WSDL file shown in the previous example, the value for the body variable (`$body`), shown in the following example, is the value of the body variable (`$body`) from the proxy service.

When this WSDL document is used for a request, the value of the body variable (`$body`) for the response from the invoked business service that the proxy service receives is displayed in the following example.

Example - Body Variable Value

```
<soap-env:body>
  <ns:lookupResponse>
    <result>
      <req:legacyboolean>true</req:legacyboolean>
    </result>
  </ns:lookupResponse>
</soap-env:body>
```

This is also the value of the body variable (`$body`) for the response returned by the proxy service using this WSDL file.

There are many tools available that take the WSDL file of a proxy service (obtained by adding the `?WSDL` suffix to the URL of the proxy in the browser) and generate a Java class with the appropriate request and response parameters to invoke the operations of that service. You can use such Java classes to invoke the proxy services that use this WSDL file.

The benefits of using a WSDL document include the following:

- The system can provide metrics for each operation in a WSDL document.
- Operational branching is possible in the pipeline. For more information, see [Branching in Pipelines](#).
- For SOAP 1.1 services, the `SOAPAction` header is automatically populated for services invoked by a proxy service. For SOAP 1.2 services, the `action` parameter of the `Content-Type` header is automatically populated for services invoked by a proxy service.

- A WSDL file is required for services using WS-Security. WS-Policies are attached to WSDL files.
- The system supports the `<url>?WSDL` syntax, which allows you to dynamically obtain the WSDL file of an HTTP proxy service. This is useful for a number of SOAP client generation tools. For more information, see [Viewing Service Bus Resources in a Web Browser](#).
- In the XQuery and XPath editors and condition builders, it is easy to manipulate the body content variable (`$body`) because the editor provides a default mapping of `$body` to the request message in the WSDL file of a proxy service. See [Message Context](#).
- The runtime contents of `$body` for a specific action can be different from the default mapping displayed in the editor. This is because Service Bus is not a programming language in which typed variables are declared and used. Instead, variables are untyped and are created dynamically at runtime when a value is assigned. In addition, the type of the variable is the type that is implied by its contents at any point in the message flow. To enable you to easily create XQuery and XPath expressions, the editor allows you to map the type for a given variable by mapping the variable to the type in the editor. To learn about using the XQuery and XPath editor to create expressions, see [Working with Variable Structures](#).

About Effective WSDL Documents and Generated WSDL Documents

In Service Bus, you can base a new service on an existing WSDL file (called a WSDL resource) and then override or add configuration properties. In the runtime, Service Bus generates an effective WSDL document for the service that includes the configuration of the WSDL resource along with additional transport and runtime configuration.

Effective WSDL Documents

For WSDL-based services, Service Bus uses *effective* WSDL documents in the runtime instead of the actual `.wsdl` files you create when you develop Service Bus services. The *effective* WSDL document represents a service's WSDL properties as configured in Service Bus and also includes additional properties configured outside of the source WSDL document. The source WSDL document serves as a template for the effective WSDL document.

When you create a service based on a WSDL document, Service Bus generates an effective WSDL document at runtime by combining properties from the original WSDL document, any transport properties you configured, runtime settings (like the target server), and any WS-Policy configurations. Any properties you add or change from the original WSDL document during runtime are included in the effective WSDL document. Properties from the source WSDL document that are not used in the new configuration are omitted from the effective WSDL document.

Service Bus can generate effective WSDL documents for SOAP and XML services that are created from a WSDL document and that use any transport that supports WSDL-based services, such as HTTP, JMS, SB, and so on. Service Bus cannot generate effective WSDL documents for services of the following types: Any SOAP, Any XML, and messaging.

Effective WSDL documents have different characteristics for proxy services and business services and for services based on WSDL ports or on WSDL bindings. For more information, see [Services Based on WSDL Ports and on WSDL Bindings](#).

Generated WSDL Documents

A *generated* WSDL document is an effective WSDL document that Service Bus generates for transport-type services that were not created from a WSDL resource but that can be described using a WSDL document. For example, you can generate a WSDL document from an EJB-based service.

Services Based on WSDL Ports and on WSDL Bindings

When you create a service based on a WSDL resource, you must base the service on a WSDL port or on a WSDL binding.

- When you create a new service based on a *binding* in a WSDL resource, you are choosing the protocol and data format defined in the selected `binding` element in the WSDL resource.
- When you create a new service based on a *port* in a WSDL resource, you are choosing the binding and the network address defined in the `port` element.

When you create or modify the service, you can change the transport, but you cannot override the data format. The port and binding definitions from the original WSDL resource are modified in the effective WSDL depending on a number of factors, as described in the following sections.

Effective WSDL Documents for Proxy Services

The following characteristics apply to effective WSDL documents generated for proxy services:

- The effective WSDL document has one and only one `wsdl:service` section.
- The `wsdl:service` section has one and only one `wsdl:port` section.
- For SOAP services, any existing `<wsdl:service>` definition is removed, and a new service definition containing a single `<wsdl:port>` is created.
- For SOAP binding over any of the supported transports the `wsdl:binding` section contains the standard WSDL SOAP binding elements along with a unique transport URI that identifies the transport.
- For XML binding over HTTP, the `wsdl:binding` section uses the standard binding elements specified in the WSDL 1.1 specification.
- For XML binding over any of the other supported transports the `wsdl:binding` section uses Oracle (Service Bus) proprietary WSDL XML binding elements.

If the service is based on a binding, the following characteristics apply:

- The effective WSDL document defines a new service and port (`<bindingname>QSService` and `<bindingname>QSPort`). None of the ports defined in the WSDL resource are included in the effective WSDL document.
- There may be multiple ports in that WSDL document associated with that binding. Each port can use a different URL. Therefore, the effective WSDL document uses the binding but generates an artificial port from the configuration on the service for that binding. All other ports are removed.

If the service is based on a port, the following characteristics apply:

- The port on which the service is based is also defined in the effective WSDL document. Any other ports defined in the WSDL resource are not included. Furthermore, if you base the proxy service on a WSDL port, the effective WSDL document uses that port name. The binding is determined from the port, and in turn, the port type is determined from the binding.
- The effective WSDL document preserves any WS-Policies associated with the port defined in the WSDL resource.
- The transport address specified in the port definition in the source WSDL document is never used as the address for a proxy service in the effective WSDL document:
 - For HTTP services, you must specify a transport address when configuring the transport. That address is used in the port definition in the effective WSDL document.
 - The URL specified as the transport address for a proxy service is always relative to a path in a Service Bus domain, because Service Bus always hosts proxy services.
- For SOAP-protocol WSDL services, the transport URI in the SOAP binding depends on the transport implementation. For standard transports (like HTTP and JMS), this value is as per the SOAP specification or another universally accepted value. For transports for which SOAP does not define a standard value, Service Bus sets one consisting of a predefined namespace with the transport ID appended at the end: `http://www.oracle.com/transport/2007/05/`.
- There is one service element in the effective WSDL document, and the port address contains a URL whose syntax and semantic are defined by the transport selected in the binding.

Effective WSDL Files for Non-Transport-Type Business Services

The following characteristics apply to effective WSDL documents generated for business services that are not transport typed:

- The effective WSDL document has one and only one `wsdl:service` section.
- The `wsdl:service` section may have more than one `wsdl:port` sections. This is generally true when load balancing is used and there are multiple endpoint addresses that can be used using one of the load-balance algorithms supported.
- For SOAP binding over any of the supported transports, the `wsdl:binding` section contains the standard WSDL SOAP binding elements along with a unique transport URI that identifies the transport.
- For XML binding over any of the supported transports, the `wsdl:binding` section contains the Oracle WSDL XML binding elements.
- The URL specified as the transport address is a remote location where the remote service is hosted.

If the service is based on a port, the following characteristics apply:

- The effective WSDL document defines a port with the same name as the port in the source WSDL document. If multiple endpoint addresses are configured for the business service, the effective WSDL document generated from it defines ports for

all the endpoints, with the names `<portname_in_template>_1`, `<portname_in_template>_2`, ...

- The binding for the new service is determined from the port, and the port type is in turn determined from the binding.
- The transport address URL is a remote location where the remote service is hosted.

If the service is based on a binding, the following characteristics apply:

- The effective WSDL document defines a new service and port, based on the port in the WSDL resource. None of the ports defined in the WSDL resource are included in the effective WSDL file.
- A binding in the WSDL resource may be associated with multiple ports. Each port can use a different URL and have a different WS-Policy attached to it. Therefore, the generated WSDL document uses the binding but generates an artificial port for that binding with no WS-Policy.
- For XML-based WSDL services, standard HTTP binding is used only if the service uses HTTP transport. For all other services created from an XML-based WSDL file, the effective WSDL document uses Oracle binding.

Effective WSDL Files for Transport-Type Business Services

Service Bus does not guarantee one and only one `wsdl:service` section in effective WSDL documents generated for transport-type business services. Because the WSDL document is generated by the transport, Service Bus does not generate nor clean up extra service-port sections. Service Bus does, however, evaluate dependencies and sets their references during export.

Examples of Proxy Services Based on a Port and on a Binding

The following example shows fragments of port and binding definitions in a WSDL resource.

Example - WSDL Resource

```
<portType name="StockQuotePortType">
...
</portType>
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
...
</binding>
<service name="StockQuoteService">
  <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
    <soap:address location="http://example.com:9999/stockquote"/>
  </port>
  <port name="StockQuotePortUK" binding="tns:StockQuoteSoapBinding">
    <soap:address location="http://example.uk:9999/stockquote"/>
  </port>
</service>
```

A Service Based on a Port

If you create a proxy service based on the example in [Examples of Proxy Services Based on a Port and on a Binding](#), the effective WSDL document will look similar to the fragment in the following example.

Example - Effective WSDL File for a Proxy Service Based on a Port

```

<binding name="StockQuoteSoapBinding" type="ns:StockQuotePortType">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  ...
</binding>
<service name="StockQuoteService">
  <port name="StockQuotePort" binding="ns:StockQuoteSoapBinding">
    <soap:address location="http://host:port/project/proxyname"/>
  </port>
</service>

```

Notice the following about the above example:

- The service name, `StockQuoteService`, is the same in both the template and the effective WSDL document.
- The binding is the same in both the template and the effective WSDL document. In this example, it specifies that the service will use the HTTP transport protocol for SOAP document style messages. The binding also specifies the same binding operation in both the WSDL resource and the effective WSDL document, but that is not shown in this example.
- The second port defined in the WSDL resource, `StockQuotePortUK`, is not defined in the effective WSDL document.
- The transport address (URI) defined in the WSDL resource's port, `http://example.com:9999/stockquote`, is different from the address generated in the effective WSDL document's port, `http://host:port/project_path/proxy_service_name`. The effective WSDL document uses the address that was specified for the transport when it was being configured.

A Service Based on a Binding

If you create a proxy service based on the `StockQuoteBinding` binding in [Examples of Proxy Services Based on a Port and on a Binding](#), the effective WSDL document will look something like the fragment in the following section.

Example - Effective WSDL Document for a Proxy Service Based on a Binding

```

<binding name="StockQuoteSoapBinding" type="ns:StockQuotePortType">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  ...
</binding>
<wsdl:service name="StockQuoteSoapBindingQSService"
  <wsdl:port name="StockQuoteSoapBindingQSPort"
    binding="ns:StockQuoteSoapBinding">
    <soap:address location="http://host:port/project/proxyname"/>
  </wsdl:port>
</wsdl:service>

```

Notice the following about the above example:

- The service and the port in the WSDL resource are different from the service and the port generated in the effective WSDL document.

- The service name in the WSDL resource and the effective WSDL document are different: `StockQuoteService` in the template and `StockQuoteSoapBindingQSService` in the effective WSDL document.
- The binding is the same in both the WSDL resource and effective WSDL document. In this example, it specifies that the service will use the HTTP transport protocol for SOAP document style messages.
- The binding also specifies the same binding operation in both the template and the effective WSDL document, but that is not shown in this example.
- The transport address (URI) defined in the WSDL resource's port, `http://example.com:9999/stockquote`, is different from the address generated in the effective WSDL's port, `http://host:port/project/stockquote`.

Importing and Exporting WSDL Resources

Exporting a WSDL document generates a JAR file that contains the effective WSDL document along with any associated dependencies, such as XML schemas. Service Bus evaluates the dependencies, and the appropriate location is added to the `location` attribute of the WSDL `import` element. You cannot export a generated WSDL document.

For more information about importing and exporting resources, see [Importing and Exporting Resources and Configurations](#).

Working with WSDL Documents in JDeveloper

In JDeveloper, a WSDL file is generated and added to the Service Bus project when you create a new proxy or business service based on a JCA adapter. You can also create a WSDL file from scratch using standard JDeveloper tools. WSDL files are not generated in the Oracle Service Bus Console. You need to import the required WSDL file in order to use it with a service.

WSDL files are a standard feature in JDeveloper. For information about the editors and tools you use to create WSDL files, see "Developing Applications Using XML" in *Developing Applications with Oracle JDeveloper*.

How to Create a WSDL Resource in JDeveloper

If the WSDL resource you want to create contains URL references to external schemas that do not currently exist in JDeveloper, such as `http://www.w3.org/2001/XMLSchema.xsd`, you must add those URL-referenced schemas, along with any dependent schemas, to Service Bus by creating XML Schema resources. WSDL resources in Service Bus can only reference locally available schemas. You can generate the WSDL file using an XML schema definition (XSD) file or using a sample file.

Defining input and output types for a new WSDL document requires specifying an XML type for the message parts. If you want to use a schema XSD file that resides on your local file system, make sure the XSD file and any XSD files that it imports all reside in the JDeveloper project directory. This ensures that the schema is deployed with the project and is made available at runtime.

To create a WSDL document in JDeveloper:

1. Do one of the following:

- To create a new WSDL resource: In the Application Navigator, right-click the project or folder in which you want to place the new WSDL file. Point to **New** and select **SOA WSDL Document**.
- To create a new WSDL resource from within a service's creation wizard: Click the **Generate WSDL** icon to the right of the **WSDL** field.

The Create WSDL dialog appears.

2. On the Create WSDL dialog, enter a unique name for the WSDL file, and enter the directory where you want to store the file or accept the default.

By default, files are stored in the project folder. This must be the current project directory or one of its subdirectories. If the specified directory does not exist, Oracle JDeveloper creates it.

3. In the **Namespace** field, enter a namespace address for the WSDL file; for example, `http://example.com/OrderProcess/wsdl`.

The namespace that you specify is defined as the `tns` namespace in the WSDL file.

4. In the **Binding** field, enter the name of the binding in the WSDL file to create for the service. Select the binding type from the list of available options (SOAP 1.1, SOAP 1.2, or XML).
5. Select **Create Port Type** to create a new port type and operation for the WSDL document, or select **Select Port Type** to select a port type from an existing WSDL document.
6. Do one of the following:
 - If you chose to *create* a port type, continue to step 7.
 - If you chose to *select* a port type, click **Select WSDL** next to the **WSDL URL** field, and browse to and select the WSDL file to use. Then select the port type from the list of available options. Skip to step 14.
7. In the **Port Type** field, enter a name for the port type that will contain the operation to use.
8. In the **Operation** field, enter the name for the operation to use.

Note:

Spaces and special characters are not allowed in an operation name or port type. Only alphabetic and numeric characters are supported, and the first character cannot be a number.

9. In the **Interface Type** field, select **One-Way Interface** for request-only messaging, or select **Synchronous Interface** for request-response messaging.

The available fields change based on your selection.

10. To the upper right of the **Input** field, click **Add a new message part**.
11. On the Add Message Part dialog, do the following:
 - a. In the **Part Name** field, enter a name for the message part.

- b. To the right of the **URL** field, click the **browse for schema file** icon to browse for and select an XML type.

The Type Chooser dialog appears with a list of the schema and WSDL files to choose from.

- c. Expand the Type Explorer tree to locate and select the schema root element to use, and click **OK**.

The Add Message Part dialog reappears with the **URL** and **Schema Element** fields populated from the Type Chooser dialog. If you selected an XSD simple type, the **Schema Element** field is replaced by a **Simple Type** field.

Tip:

If the schema you want to use is not located in the project in which you are working, you can import a schema XSD file or WSDL file into the project using the **Import Schema File** or **Import WSDL** icon in the upper right corner of the dialog.

- d. Click **OK** again.

The new message part information appears in the **Input** field of the Create WSDL dialog.

12. If needed, repeat the above steps to define additional message parts.
13. If you selected **Synchronous Interface** as the interface type, repeat the above steps to define the output and fault message parts.
14. Click **OK**.

Note:

Partner link types are generally used in BPEL, so you do not need to select **Generate partnerlinkType extension** for Service Bus.

How to Generate a WSDL File from a Service in JDeveloper

A generated WSDL file is a WSDL file resource that Service Bus generates for a service that did not start with a WSDL resource but that can be described using a WSDL file. You can generate a WSDL file associated with an EJB or JEJB transport-typed business service or a JEJB proxy service.

To generate a WSDL file from service in JDeveloper:

1. In the Application Navigator, right-click the project or folder that contains the proxy or business service from which you want to generate the WSDL file.
2. Point to **Service Bus** and then click **Generate WSDL**.

The Generate WSDL dialog appears.

3. Specify a **WSDL Name** and location for the WSDL file.

By default, the current location is the path to the project and the name of the folder in which the service resides.

4. Click **OK**.

Caution:

If you are generating a WSDL file to use with a pipeline configured for *resequencing*, it's important to note that the resequencer supports only one-way (request) operations. The WSDL file produced here includes output tags, which need to be removed from the port type and binding definitions before you can use it with the pipeline.

How to Edit a WSDL Document in JDeveloper

Once you create a WSDL document, you can further configure them using the standard XML Editor in JDeveloper. For information about the editor and tool you use to view and configure WSDL documents, see "Developing Applications Using XML" in *Developing Applications with Oracle JDeveloper*.

How to Delete a WSDL Document in JDeveloper

Once you create a WSDL document, you can delete it from the project and the file system. Using the refactoring feature of JDeveloper, you can view any resources that reference the WSDL document. To view dependencies before deleting, right-click the WSDL document and select **Explore Dependencies**.

To delete a WSDL document in JDeveloper:

1. In the Application Navigator, right-click the WSDL file you want to delete.
2. Point the **Refactor** and select **Delete**.
The Confirm Delete dialog appears.
3. If the WSDL document is referenced by another Service Bus resource, click **Show Usages** to view the references to verify you want to delete the WSDL document.
4. When you are certain you want to delete the WSDL document, click **Yes**.

Working with WSDL Documents in the Oracle Service Bus Console

If you are using the Oracle Service Bus Console, you can create WSDL documents by importing them or by creating a WSDL resource. For more information on importing, see [Importing and Exporting Resources and Configurations](#). Use the following procedure to manually create WSDL resources. You should have an existing WSDL file to upload into the WSDL resource.

How to Create a WSDL Resource in the Console

If the WSDL resource you want to create contains URL references to external schemas that do not currently exist in Service Bus, such as `http://www.w3.org/2001/XMLSchema.xsd`, you must import those URL-referenced schemas—and any dependent schemas—into the Oracle Service Bus Console by creating XML Schema resources. WSDL resources in Service Bus can only reference locally available schemas.

To create a WSDL resource in the Console:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.

2. In the Project Navigator, right-click the project or folder to contain the new WSDL document, point to **Create**, and select **WSDL**.

The Create WSDL dialog appears.

3. Do one of the following:
 - To create the resource from an existing WSDL file, click **Browse** next to the **File Upload** field and then navigate to and select the WSDL file to use.

The **Resource Name** field is automatically populated with the file name minus the file extension. You can change this name.

- To create a new WSDL file, enter a unique name for the WSDL resource.

4. Optionally, enter a brief description of the resource.

5. Click **Create**.

The WSDL elements, if defined, appear in the WSDL Definition Editor.

6. To modify the WSDL file content, do the following:

- a. Click **Edit Source** in the toolbar.

The Edit Source dialog appears.

- b. To browse to and select a new WSDL file to upload, click **Browse**.

- c. To modify the contents of the file, update the code directly in the Contents section of the dialog.

- d. Click **Save**.

7. In the WSDL Definition Editor toolbar, click **Save**.

If there are any unresolved references for the new WSDL document, a **Conflict** icon appears next to the editor's title bar. Use the previous and next arrow buttons to scroll through any errors.

8. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Export a WSDL File in the Console

Use the Project or Folder Definition Editor to export a WSDL file associated with a proxy service or a business service. Service Bus exports the WSDL file as a JAR file. You can export a WSDL file only when you are outside a session.

By exporting a WSDL document, you make it available for consumption by external tools such as IDEs. Note that the WSDL export feature is different from the Export Resources feature, which you use to move and stage resources between two domains. This feature is not available for all service types.

Exporting a WSDL File from a Project or Folder in the Console

To export a WSDL file from a project or folder in the console:

1. In the Project Navigator, click the project or folder that contains the WSDL file to export.

The Project or Folder Definition Editor appears.

2. In the resources table, click the **Export WSDL** icon in the row of the proxy or business service whose WSDL file you want to export.

A file dialog appears. The type of dialog and its options differ depending on the web browser you are using.

3. Click **Open** to open the JAR file or click **Save** to save the JAR file to your desktop.

Exporting a WSDL File From a Service Definition Editor

To export a WSDL file from a service definition editor:

1. Make sure you are not in an open sessions in the Oracle Service Bus Console.
2. In the Project Navigator, click the proxy or business service from which you want to export the WSDL file.

The Proxy or Business Service Definition Editor appears.

3. In editor's toolbar in the upper right, click the **Export WSDL**.

A file dialog appears. The type of dialog and its options differ depending on the web browser you are using.

4. Click **Open** to open the JAR file or click **Save** to save the JAR file to your desktop.

How to Generate a WSDL File from a Service in the Console

A generated WSDL file is a WSDL file resource that Service Bus generates for a service that did not start with a WSDL resource but that can be described using a WSDL file. Use the Project or Folder Definition Editor to generate a WSDL file associated with an EJB or JEJB transport-typed business service or a JEJB proxy service. You must be working within a session to generate a WSDL file.

To generate a WSDL file from a business service in the console:

1. In the Project Navigator, click the project or folder that contains the proxy or business service from which you want to generate the WSDL file.

The Project or Folder Definition Editor appears.

2. In the resources table, click the **Generate WSDL** icon in the row of the service whose WSDL file you want to generate.

The Generate WSDL Resource dialog appears with the current location (project name and the name of the folder in which the business service resides) highlighted.

3. Specify a name and location for the WSDL resource, and then click **Generate WSDL**.

Caution:

If you are generating this WSDL resource to use with a pipeline configured for *resequencing*, it's important to note that the resequencer supports only one-way (request) operations. The WSDL file produced here includes output tags, which need to be removed from the port type and binding definitions before you can use it with the pipeline.

How to Edit a WSDL Document in the Console

Once you create a WSDL resource in the Oracle Service Bus Console, you can modify the file as needed, but only by modifying the source code directly.

To edit a WSDL Document:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.

2. In the Project Navigator, click the WSDL resource to edit.

The WSDL Definition Editor appears.

3. Click **Edit Source** in the toolbar.

The Edit Source dialog appears.

4. To browse to and select a new WSDL file to upload, click **Browse**.

5. To modify the contents of the file, update the code directly in the Contents section of the dialog.

6. Click **Save**.

7. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Delete a WSDL Document in the Console

If any resources reference the WSDL document you want to delete, remove those references before deleting the WSDL resource. In the Oracle Service Bus Console, open the WSDL document in the WSDL Definition Editor and click the **References** icon in the upper right to find out if it has any references.

To delete a WSDL document in the console:

1. In the Application Navigator, expand the project and folders containing the WSDL resource to delete.

2. Right-click the name of the WSDL document, and select **Delete**.

The WSDL resource is deleted. A Deletion Warning icon appears when other resources reference this resource. You can delete the resource with a warning confirmation. This might result in conflicts due to unresolved references to the deleted resource.

3. Click **Activate** to end the session and deploy the configuration to the runtime.

Viewing Effective WSDL Documents

You can view both the design-time and the effective WSDL documents through a web browser. Accessing the files through a browser window displays the contents of the file in XML format.

There are three ways to access an effective WSDL document:

- In a web browser, enter the fixed HTTP URL, using the following syntax:

```
http://host:port/sbresource?PROXY/project_path/proxy_service_name
```


or

`http://host:port/sbresource?BIZ/project_path/business_service_name`

This works for all services for which Service Bus can generate effective WSDL documents.

- In a web browser, enter the URL for an HTTP-based proxy service, appended with `?WSDL`.

This works only for HTTP-transport-based services for which Service Bus can generate effective WSDL documents.

- Export the WSDL resource. For more information, see [How to Export a WSDL File in the Console](#) and [How to Generate a WSDL File from a Service in the Console](#). For general export information, see [Importing and Exporting Resources and Configurations](#).

Service Bus provides a resource servlet that is used to expose the resources registered in Service Bus through a URL. For more information on accessing Service Bus resources with a URL, see [Viewing Service Bus Resources in a Web Browser](#).

To view the design-time WSDL document, enter the URL using the following syntax:

`http://host:port/sbresource?WSDL/project_path/wsd_name`

Part III

Working with Oracle Service Bus Pipelines

This part describes how to create and use pipelines in your Service Bus projects.

This part contains the following chapters:

- [Modeling Message Flow in Oracle Service Bus](#)
- [Working with Pipelines in Oracle Service Bus Console](#)
- [Working with Pipeline Actions in Oracle Service Bus Console](#)
- [Working With Expression Editors in Oracle Service Bus Console](#)
- [Working with Pipelines in Oracle JDeveloper](#)
- [Working with Pipeline Actions in Oracle JDeveloper](#)
- [Working with Pipeline Templates](#)

Modeling Message Flow in Oracle Service Bus

This chapter describes the high-level aspects and concepts of creating and configuring pipelines, or message flows, using the Oracle Service Bus Console. Topics include pipeline components, message transformation, routing and service callout, error handling, message context, and quality of service (QoS).

In Service Bus, a message flow defines the implementation of a pipeline. You can create and configure pipelines in Oracle Service Bus Console or Oracle JDeveloper. You can also define message flow in split-joins. For more information, see [Improving Service Performance with Split-Join](#).

The following sections describe pipelines in Service Bus:

- [Pipeline Components](#)
- [Branching in Pipelines](#)
- [Configuring Actions in Stages and Route Nodes](#)
- [Performing Transformations in Pipelines](#)
- [Constructing Service Callout Messages](#)
- [Using Attachments with Service Callout Messages](#)
- [Handling Errors as the Result of a Service Callout](#)
- [Handling Errors in Pipelines](#)
- [Using Dynamic Routing](#)
- [Accessing Databases Using XQuery](#)
- [Understanding Message Context](#)
- [Working with Variable Structures](#)
- [Quality of Service](#)
- [Using Work Managers with Service Bus](#)
- [Content Types, JMS Type, and Encoding](#)
- [Throttling Pattern](#)
- [WS-I Compliance](#)
- [Converting Between SOAP 1.1 and SOAP 1.2](#)

For instructions on creating and configuring pipelines using the Oracle Service Bus Console, see:

- [Working with Pipelines in Oracle Service Bus Console](#)
- [Working with Pipeline Actions in Oracle Service Bus Console](#)

For instructions on creating and configuring pipelines using Oracle JDeveloper, see:

- [Working with Pipelines in Oracle JDeveloper](#)
- [Working with Pipeline Actions in Oracle JDeveloper](#)

Pipeline Components

A pipeline is composed of components that define the logic for routing and manipulating messages as they flow through a pipeline. Nodes are configured to route messages through the message flow. Stages and actions contain rules for processing and transforming messages.

[Table 12-1](#) describes the components available for defining pipelines.

Table 12-1 Pipeline Components

Component Type	Summary
Start node	Every pipeline begins with a start node. All messages enter the pipeline through the start node, and all response messages are returned to the client through the start node. There is nothing to configure in a start node.
Pipeline pair node	A pipeline pair node combines a single request pipeline and a single response pipeline in one top-level element. A pipeline pair node can have only one direct descendant in the pipeline. During request processing, only the request pipeline is executed when Service Bus processes a pipeline pair node. The execution path is reversed when Service Bus processes the response pipeline.
Stage	Request pipelines, response pipelines, and error handlers can contain stages, where you configure actions to manipulate messages passing through the pipeline. See also Configuring Actions in Stages and Route Nodes .
Error handler	An error handler can be attached to any node or stage, to handle potential errors at that location. See also Handling Errors in Pipelines .
Branch node	A branch node allows processing to proceed along exactly one of several possible paths. Operational branching is supported for WSDL-based services, where the branching is based on operations defined in the WSDL file. Conditional branching is supported for conditions defined in an XPath-based switch table. See also Branching in Pipelines .

Table 12-1 (Cont.) Pipeline Components

Component Type	Summary
Route node	<p>A route node performs request/response communication with another service or component. It represents the boundary between request and response processing for the pipeline. When the route node dispatches a request message, the request processing is considered complete. When the route node receives a response message, the response processing begins. The route node supports conditional routing as well as request and response transformations.</p> <p>Because a route node represents the boundary between request and response processing, it cannot have any descendants in the pipeline. See also Configuring Actions in Stages and Route Nodes.</p>

Building a Message Flow

In general, a message flow is likely to be designed in one of the following ways:

- For non-operational services (services that are not based on WSDL files with operations), the flow consists of a single pipeline pair at the root followed by a route node.
- For operational services, the flow consists of a single pipeline pair at the root, followed by a branch node based on an operation, with each branch consisting of a pipeline pair followed by a route node.

Message Execution

[Table 12-2](#) and [Table 12-3](#) briefly describe how request and response messages are processed in a typical message flow.

Table 12-2 Path of a Request Message During a Message Flow

Pipeline Node	What Happens During Message Processing?
Request Processing	Request Processing container.
Start node	Request processing begins at the start node.
Pipeline pair node	Executes the request pipeline only.
Branch node	Evaluates the branch table and proceeds down the relevant branch.
Route node	<p>Performs the route along with any request transformations.</p> <p>In the pipeline, regardless of whether routing takes place or not, the route node represents the transition from processing a request to processing a response. At the route node, the direction of the message flow is reversed. If a request path does not have a route node, the response processing is initiated in the reverse direction without waiting for any response.</p>

Table 12-3 Path of a Response Message During a Message Flow

Pipeline Node	What Happens During Message Processing?
Response Processing	Skips any branch nodes and continues with the node that preceded the branch.
Route node	Executes any response transformations.
Branch node	Skips any branch nodes and continues with the node that preceded the branch.
Pipeline pair node	Executes the response pipeline.
Start node	Sends the response back to the client.

Branching in Pipelines

Two kinds of branching are supported in pipelines: *operational* branching, configured in an operational branch node, and *conditional* branching, configured in a conditional branch node.

Operational Branching

When message flows define WSDL-based pipeline services, operation-specific processing is required. When you create an operational branch node in a pipeline, you can build branching logic based on the operations defined in the WSDL file.

You must use operational branching when a pipeline is based on a WSDL file with multiple operations. You can consider using an operational branch node to handle messages separately for each operation.

Conditional Branching

Use conditional branching to branch based on a specified condition, for example the document type of a message.

Conditional branching is driven by a lookup table with each branch tagged with simple, unique string values, for example, `QuantityEqualToOrLessThan150` and `QuantityMoreThan150`.

You can configure a conditional branch to branch based on the value of a variable in the message context (declared, for example, in a stage earlier in the pipeline), or you can configure the condition to branch based on the results of an XPath expression defined in the branch itself.

At runtime, the variable or the expression is evaluated, and the resulting value is used to determine which branch to follow. If no branch matches the value, the default branch is followed. A branch node may have several descendants in the pipeline: one for each branch, including the default branch. You should always define a default branch. You should design the pipeline in such a way that the value of a lookup variable is set before reaching the branch node.

For example, consider the following case using a lookup variable. A pipeline is of type any SOAP or any XML, and you need to determine the type of the message so you can perform conditional branching. You can design a stage action to identify the message type and then design a conditional branching node later in the flow to separate processing based on the message type.

Now consider the following case using an XPath expression in the conditional branch node. You want to branch based on the quantity in an order. That quantity is passed using a variable that can be retrieved from a known location in \$body.

You could define the following XPath expression to retrieve the quantity:

```
declare namespace openuri="http://www.openuri.org/";
declare namespace com="oracle.com/demo/orders/cmnCust";
./openuri:processCust/com:cmnCust/com:Order_Items/com:Item/com:Quantity
```

The condition (for example, <500) is then evaluated in order down the message flow against the expression. Whichever condition is satisfied first determines which branch is followed. If no branch condition is satisfied, then the default branch is followed.

You can use conditional branching to expose the routing alternatives for the message flow at the top level flow view. For example, consider a situation where you want to invoke service A or service B based on a condition known early in the message flow (for example, the message type). You could configure the conditional branching in a routing table in the route node. However, that makes the branching somewhat more difficult to follow if you are just looking at the top level of the flow. Instead, you could use a conditional branch node to expose this branching in the pipeline itself and use simple route nodes as the subflows for each of the branches.

Consider your business scenario before deciding whether you configure branching in the pipeline or in a stage or route node. When making your decision, remember that configuring branches in the pipeline can be awkward in the design interface if a large number of branches extend from the branch node.

Configuring Actions in Stages and Route Nodes

Actions provide instructions for handling messages in pipeline stages, error handler stages, and route nodes. The context determines which actions are available, as described in the following sections:

- [Communication Actions](#)
- [Flow Control Actions](#)
- [Message Processing Actions](#)
- [Reporting Actions](#)

See Also

- [Working with Pipeline Actions in Oracle Service Bus Console](#)
- [Working with Pipeline Actions in Oracle JDeveloper](#)

Communication Actions

Communication actions control message flow. [Table 12-4](#) describes the communication actions.

Table 12-4 Communication Actions

Action	Use to...	Available in
Dynamic Publish	Publish a message to a service specified by an XQuery expression.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Publish	Identify a statically specified target service for a message and to configure how the message is packaged and sent to that service.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Publish Table	Publish a message to zero or more statically specified services. Switch-style condition logic is used to determine at runtime which services will be used for the publish.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Routing Options	Modify any or all of the following properties in the outbound request: URI, Quality of Service, Mode, Retry parameters, Message Priority.	<ul style="list-style-type: none"> • Route node
Service Callout	Configure a synchronous (blocking) callout to a Service Bus-registered proxy service, business service, pipeline or split-join. See Constructing Service Callout Messages .	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Transport Headers	Set the header values in messages. See Configuring Transport Headers in Pipelines .	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Dynamic Routing	Assign a route for a message based on routing information available in an XQuery resource.	<ul style="list-style-type: none"> • Route Node
Routing	Identify a target service for the message and configure how the message is routed to that service.	<ul style="list-style-type: none"> • Route Node
Routing Table	Select different routes based upon the results of a single XQuery expression.	<ul style="list-style-type: none"> • Route Node

Flow Control Actions

Flow controls actions implement conditional routing, conditional looping, and error handling. You can also use them to notify the invoker of success or to skip the rest of the actions in the stage. [Table 12-5](#) describes the flow control actions.

Table 12-5 Flow Control Actions

Action	Use to...	Available in
For each	Iterate over a sequence of values and execute a block of actions	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node

Table 12-5 (Cont.) Flow Control Actions

Action	Use to...	Available in
If... then...	Perform an action or set of actions conditionally, based on the Boolean result of an XQuery expression.	<ul style="list-style-type: none"> • Pipeline stage • Route node • Error handler stage
Raise error	Raise an exception with a specified error code (a string) and description.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Reply	Specify that an immediate reply be sent to the invoker. The reply action can be used in the request, response or error pipeline. You can configure it to result in a reply with success or failure. In the case of reply with failure where the inbound transport is HTTP, the reply action specifies that an immediate reply is sent to the invoker.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Resume	Resume message flow after an error is handled by an error handler. This action has no parameters and can only be used in error handlers.	<ul style="list-style-type: none"> • Error handler stage
Skip	Specify that at runtime, the execution of this stage is skipped and the processing proceeds to the next stage in the pipeline. This action has no parameters and can be used in the request, response or error pipelines.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node

Message Processing Actions

The actions in this category process the message flow. [Table 12-6](#) describes the message processing actions.

Table 12-6 Message Processing Actions

Action	Use to...	Available in
Assign	Assign the result of an XQuery or XSLT expression to a context variable.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Delete	Delete a context variable or a set of nodes specified by an XPath expression.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Insert	Insert the result of an XQuery or XSLT expression at an identified place relative to nodes selected by an XPath expression.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node

Table 12-6 (Cont.) Message Processing Actions

Action	Use to...	Available in
Java Callout	Invoke a Java method from within the pipeline.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
MFL Translate	Convert message content from XML to non-XML, or vice versa, in the message pipeline. An MFL is a specialized XML document used to describe the layout of binary data. It is an Oracle proprietary language used to define rules to transform formatted binary data into XML data, or vice versa.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
nXSD Translate	Convert message content from XML to native format data, or vice versa, in the message pipeline.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Rename	Rename elements selected by an XPath expression without modifying the contents of the element.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Replace	<p>Replace a node or the contents of a node specified by an XPath expression. The node or its contents are replaced with the value returned by an XQuery expression.</p> <p>A replace action can be used to replace simple values, elements and even attributes. An XQuery expression that returns nothing is equivalent to deleting the identified nodes or making them empty, depending upon whether the action is replacing entire nodes or just node contents.</p> <p>The replace action is one of a set of Update actions.</p>	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Validate	Validate elements selected by an XPath expression against an XML schema element or a WSDL resource. You can validate global elements only; Service Bus does not support validation against local elements.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node

Reporting Actions

You use the actions in this category to log or report errors and generate alerts if required in a message flow within a stage. [Table 12-7](#) describes the reporting actions.

Table 12-7 Reporting Actions

Action	Use to...	Available in
Alert	Generate alerts based on message context in a pipeline, to send to an alert destination. Unlike SLA alerts, notifications generated by the alert action are primarily intended for business purposes, or to report errors, and not for monitoring system health. Alert destination should be configured and chosen with this in mind. If pipeline alerting is not enabled for the service or enabled at the domain level, the configured alert action is bypassed during message processing.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Log	Construct a message to be logged and to define a set of attributes with which the message is logged.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node
Report	Enable message reporting for a pipeline. An XQuery/XSLT expression is used to create the data that is reported to the Service Bus dashboard. You use key value pairs to extract key identifiers from any message context variable or message payload, and ignore the rest of the message.	<ul style="list-style-type: none"> • Pipeline stage • Error handler stage • Route node

Configuring Transport Headers in Pipelines

The transport header action is a communication type action, and it is available in pipeline stages and error handler stages.

Global Pass Through and Header-Specific Copy Options

The following options are available when you configure a transport headers action:

- The **Pass all Headers through Pipeline** option specifies that at runtime, the transport headers action passes all headers through from the inbound message to the outbound message or vice versa. Every header in the source set of headers is copied to the target header set, overwriting any existing values in the target header set.
- The **Copy Header from Inbound Request** option and the **Copy Header from Outbound Response** options specifies that at runtime, the transport headers action copies the specific header with which this option is associated from the inbound message to the outbound message or vice versa.

Use the options in a way that best suits your scenario. Both options result in the headers in the source header set being copied to the target header set, overwriting any existing value in the target set. Note that the **Pass all Headers through Pipeline** option is executed before the header-specific **Copy Header** options. In other words, for a given transport headers action configuration, if you select **Pass all Headers through Pipeline**, there is no need to select the **Copy Header** option for given headers.

However, you can select **Pass all Headers through Pipeline** to copy all headers, and subsequently configure the action such that individual headers are deleted by selecting **Delete Header** for specific headers.

Caution:

Because transport headers are specific to the transport types, it is recommended that the pass-through (or copy) options only be used to copy headers between services of the same transport type. Passing (or copying) headers between services of different transport types can result in an error if the header being passed is not accepted by the target transport. For the same reasons, be careful when you specify a header name using the Set Header option.

How the Runtime Uses Transport Headers Settings

You can use transport header actions to configure the values of the transport headers for outbound requests (the messages sent out by a proxy service in route, publish, or service callout actions) and inbound responses (the response messages a proxy service sends back to clients). In general, the header values can be:

- Specified using an XQuery expression
- Passed through from the source to the target service
- Deleted while going from the source to the target service

The transport headers action allows you to set, delete, or pass-through the headers in `$inbound` or `$outbound`. If you set or delete these headers and then log `$inbound` or `$outbound`, you can see the effects of your changes. However, when the message is sent out, the Service Bus binding layer may modify or remove some headers in `$inbound` or `$outbound` and the underlying transport may in turn ignore some of these headers and use its own values. An important distinction is that any modifications done by the binding layer on a header are done directly to `$inbound` and `$outbound`, whereas modifications done by the transport affects only the message's *wire format*. For example, although you can specify a value for the outbound `Content-Length` header, the binding layer deletes it from `$outbound` when sending the message. Consequently, the modification is visible in the response path (for example, you can see the modified value if you log `$outbound`). If you set the `User-Agent` header in `$outbound`, the HTTP transport ignores it and use its own value—however, the value in `$outbound` is not changed.

[Limitations to Transport Header Values you Specify in Transport Header Actions](#) describes the transport headers that are ignored or overwritten at runtime and other limitations that exist for specific transport headers.

Limitations to Transport Header Values you Specify in Transport Header Actions

[Table 12-8](#) describes the transport headers that are ignored or overwritten at runtime and other limitations that exist for specific transport headers.

Table 12-8 Limitations to Transport Header Values You Specify in Transport Header Actions

Transport	Description of Limitation...	Outbound Request Header	Inbound Response Header
HTTP	Service Bus runtime may overwrite these headers in the binding layer when preparing the message for dispatch. If these headers are modified, \$inbound and \$outbound are updated accordingly.	Content-Type	Content-Type
HTTP	The underlying transport may ignore these headers and use different values when sending the message. Any changes done by the transport will not be reflected in \$inbound or \$outbound.	<ul style="list-style-type: none"> • Accept • Content-Length • Connection • Host • User-Agent 	<ul style="list-style-type: none"> • Content-Length • Date • Transfer-Encoding
JMS	Can only be set when the request is with respect to a one-way service or a request/response service based on JMSMessageID correlation. If sending to a request/response service based on JMSCorrelationID correlation, these headers are overwritten at runtime.	JMSCorrelationID	JMSCorrelationID
JMS	The Service Bus runtime sets these headers. In other words, any specifications you make for these headers at design time are overwritten at runtime. Note: Header names with the JMS_IBM prefix are to be used with respect to destinations hosted by an IBM MQ server.	<ul style="list-style-type: none"> • JMSMessageID • JMSRedelivered • JMSTimestamp • JMSXDeliveryCount • JMSXUserID • JMS_IBM_PutDate • JMS_IBM_PutTime • JMS_IBM_PutApplType • JMS_IBM_Encoding • JMS_IBM_Character_Set 	<ul style="list-style-type: none"> • JMSMessageID • JMSRedelivered • JMSTimestamp • JMSXDeliveryCount • JMSXUserID • JMS_IBM_PutDate • JMS_IBM_PutTime • JMS_IBM_PutApplType • JMS_IBM_Encoding • JMS_IBM_Character_Set
JMS	Because IBM MQ does not allow certain properties to be set by a client application, if you set these headers with respect to an IBM MQ destination, a runtime exception is raised.	<ul style="list-style-type: none"> • JMSXDeliveryCount • JMSXUserID • JMSXAppID 	<ul style="list-style-type: none"> • JMSXDeliveryCount • JMSXUserID • JMSXAppID

Table 12-8 (Cont.) Limitations to Transport Header Values You Specify in Transport Header Actions

Transport	Description of Limitation...	Outbound Request Header	Inbound Response Header
JMS	These headers cannot be deleted when the Pass all Headers through Pipeline option is also specified.	<ul style="list-style-type: none"> JMSDeliveryMode JMSExpiration JMSMessageID JMSRedelivered JMSTimestamp JMSXDeliveryCount 	<ul style="list-style-type: none"> JMSDeliveryMode JMSExpiration JMSMessageID JMSRedelivered JMSTimestamp JMSXDeliveryCount JMSCorelationID—if the inbound message has the correlation ID set. For example, if the inbound response comes from a registered JMS business service
FTP and File	No limitations. In other words you can set or delete the header(s) for File and FTP transports and your specifications are honored by the Service Bus runtime. Note: For FTP and file proxies, there is an transport request header <code>fileName</code> . The value of this request header is the name of the file being polled.	N/A	N/A
Email	The Service Bus runtime sets these headers. In other words, any specifications you make for these headers at design time are overwritten at runtime.	Content-Type	Content-Type
Email	Service Bus does not use these headers in outbound requests. If you set them dynamically (that is, if you set them in the <code>\$outbound</code> headers section), Service Bus ignores them. These headers are received in <code>\$inbound</code> . <code>Date</code> is the time the mail was sent by the sender. <code>From</code> is retrieved from incoming mail headers.	<ul style="list-style-type: none"> From (Name) Date 	N/A

Note:

The same limitations around setting certain transport headers and metadata are true when you set the `inbound` and `outbound` context variables, and when you use the Service Bus Test Console to test your proxy or business services.

Performing Transformations in Pipelines

Transformation maps describe the mapping between two data types. Service Bus supports data mapping that uses the XQuery and the eXtensible Stylesheet Language Transformation (XSLT) standards. XSLT maps describe XML-to-XML mappings. XQuery maps can describe XML-to-XML, XML to non-XML, and non-XML to XML mappings.

The point in a pipeline at which you specify a transformation depends on whether:

- The message format relies on target services—that is, the message format must be in a format acceptable by the route destination. This applies when the transformation is performed in a route node or in one of the publish actions.

Publish actions identify a target service for a message and configure how the message is packaged and sent to that service. Service Bus also provides publish table actions. A publish table action consists of a set of routes wrapped in a switch-style condition table. It is a shorthand construct that allows different routes to be selected, based upon the results of a single XQuery expression.

- You perform the transformation on the response or request message regardless of the route destination. In this case, you can configure the transformations in the request or response pipeline stages.

Transformations and Publish Actions

When transformations are designed in publish actions, the transformations have a local copy of the `$outbound` variable and message-related variables (`$header`, `$body`, and `$attachments`). Any changes you make to an outbound message in a publish action affect only the published message. In other words, the changes you make in the publish action are rolled back before the message flow proceeds to any actions that follow the publish action in your pipeline.

For example, consider a message flow that deals with a large purchase order, and you have to send the summary of the purchase order, through email, to the manager. The summary of the of the purchase order is created in the SOAP body of the incoming message when you include a publish action in the request pipeline. In the publish action, the purchase order data is transformed into a summary of the purchase order—for example, all the attachments in `$attachments` can be deleted because they are not required in the summary of the purchase order. After the publish action, the message in its state prior to the publish action continues through the message flow, as described in the following section.

Publish Action Behavior with Quality of Service

This section describes how the publish action behaves with different quality of service (QoS) settings.

Exactly-Once – When QoS is exactly-once, the publish action waits (blocking call) until the response from the target service is available, although the response itself is discarded. When the target is a business service, the publish action waits until the business service response is available. When the target is a proxy service, the publish action waits until the proxy service's response pipeline completes.

Best-Effort – When QoS is best-effort and the target service is a one-way proxy service or a one-way business service with retry count > 0, the publish action waits until the

target service returns. With a one-way target service there is no response, but the publish action waits until the request is delivered.

If the target proxy or business service is request-response or a one-way business service with retry count = 0, the publish action does not wait for the response (non-blocking call).

Transformations and Route Nodes

You may need to route messages to one of two destinations, based on a WS-addressing header. In that case, content-based routing and the second destination require the newer version of the document in the SOAP body. In this situation, you can configure the route node to conditionally route to one of the two destinations. You can configure a transformation in the route node to transform the document for the second destination.

You can also set the control elements in the outbound context variable (`$outbound`) to influence the behavior of the system for the outbound message (for example, you can set the Quality of Service).

See [Inbound and Outbound Variables](#) and [Constructing Messages to Dispatch](#) for information about the sub-elements of the inbound and outbound variables and how the content of messages is constructed using the values of the variables in the message context.

See Also

- [Message Context](#)
- [Transforming Data with XQuery](#) and [Transforming Data with XSLT](#)

Constructing Service Callout Messages

When Service Bus makes a call to a service using a service callout action, the content of the message is constructed using the values of variables in the message context. The message content for outbound messages is handled differently depending upon the type of the target service.

How the message content is created depends on the type of the target service and whether you choose to configure the SOAP body or the payload (parameters or document), as described in the following topics:

- [SOAP Document Style Services](#)
- [SOAP RPC Style Services](#)
- [XML Services](#)
- [Messaging Services](#)

SOAP Document Style Services

Messages for SOAP Document Style services (including EJB document and document-wrapped services), can be constructed as follows:

- The variable assigned for the request document contains the SOAP body.
- The variable assigned for the SOAP request header contains the SOAP header.

- The response must be a single XML document—it is the content of the SOAP body plus the SOAP header (if specified)

To illustrate how messages are constructed during callouts to SOAP Document Style services, consider a service callout action configured as follows:

- **Request Document Variable:** myreq
- **Response Document Variable:** myresp
- **SOAP Request Header:** reqheader
- **SOAP Response Header:** respheader

Assume also that at runtime, the request document variable, myreq, is bound to the following XML.

Example - Content of Request Variable (myreq)

```
<sayHello xmlns="http://www.openuri.org/">
  <intVal>100</intVal>
  <string>Hello Oracle</string>
</sayHello>
```

At runtime, the SOAP request header variable, reqheader, is bound to the following SOAP header.

Example - Content of SOAP Request Header Variable (reqheader)

```
<soap:Header xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
  <wsa:Action>...</wsa:Action>
  <wsa:To>...</wsa:To>
  <wsa:From>...</wsa:From>
  <wsa:ReplyTo>...</wsa:ReplyTo>
  <wsa:FaultTo>...</wsa:FaultTo>
</soap:Header>
```

In this example scenario, the full body of the message sent to the external service is shown in the following example (the contents of the myreq and reqheader variables are shown in bold).

Example - Message Sent to the Service as a Result of Service Callout Action

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  <soap:Header xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
      <wsa:Action>...</wsa:Action>
      <wsa:To>...</wsa:To>
      <wsa:From>...</wsa:From>
      <wsa:ReplyTo>...</wsa:ReplyTo>
      <wsa:FaultTo>...</wsa:FaultTo>
    </soap:Header>
  <soapenv:Body>
    <sayHello xmlns="http://www.openuri.org/">
      <intVal>100</intVal>
      <string>Hello Oracle</string>
    </sayHello>
  </soapenv:Body>
</soapenv:Envelope>
```

Based on the configuration of the service callout action described above, the response from the service is assigned to the `myresp` variable. The full response from the external service is shown in the following example.

Example - Response Message From the Service as a Result of Service Callout Action

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <env:Header/>
  <env:Body env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <m:sayHelloResponse xmlns:m="http://www.openuri.org/">
      <result xsi:type="xsd:string">This message brought to you by Hello
Oracle and the number 100
      </result>
    </m:sayHelloResponse>
  </env:Body>
</env:Envelope>
```

In this scenario, the `myresp` variable is assigned the value shown in the following example.

Example - Content of Response Variable (`myresp`) as a Result of Service Callout Action

```
<m:sayHelloResponse xmlns:m="http://www.openuri.org/">
  <result ns0:type="xsd:string" xmlns:ns0="http://www.w3.org/2001/XMLSchema-
instance">
This message brought to you by Hello Oracle and the number 100
  </result>
</m:sayHelloResponse>
```

SOAP RPC Style Services

Messages for SOAP RPC Style services (including EJB RPC services) can be constructed as follows:

- Request messages are assembled from message context variables.
 - The SOAP body is built based on the SOAP RPC format (operation wrapper, parameter wrappers, and so on).
 - The SOAP header is the content of the variable specified for the SOAP request header, if one is specified.
 - Part as element—the parameter value is the variable content.
 - Part as simple type—the parameter value is the string representation of the variable content.
 - Part as complex type—the parameter corresponds to renaming the root of the variable content after the parameter name.
- Response messages are assembled as follows:
 - The output content is the content of SOAP header, if a SOAP header is specified.
 - Part as element—the output content is the child element of the parameter; there is at most one child element.

- Part as simple/complex type—the output content is the parameter itself.

To illustrate how messages are constructed during callouts to SOAP RPC Style services, look at this example with the following configuration:

- A message context variable `input1` is bound to a value 100.
- A message context variable `input2` is bound to a string value: Hello Oracle.
- A service callout action configured as follows:
 - **Request Parameter `intval`:** `input1`
 - **Request Parameter `string`:** `input2`
 - **Response Parameter `result`:** `output1`

In this scenario, the body of the outbound message to the service is shown in the following example.

Example - Content of Outbound Message

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <sayHello2 xmlns="http://www.openuri.org/">
      <intval>100</intval>
      <string >Hello Oracle</string>
    </sayHello2>
  </soapenv:Body>
</soapenv:Envelope>
```

The response returned by the service to which the call was made is shown in the following example.

Example - Content of Response Message From the helloWorld Service

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <env:Header/>
  <env:Body env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <m:sayHello2Response xmlns:m="http://www.openuri.org/">
      <result xsi:type="n1:HelloWorldResult" xmlns:n1="java:">
        <message xsi:type="xsd:string">
          This message brought to you by Hello Oracle and the number 100
        </message>
      </result>
    </m:sayHello2Response>
  </env:Body>
</env:Envelope>
```

The message context variable `output1` is assigned the value shown in the following example.

Example - Content of Output Variable (output1)

```
<message ns0:type="xsd:string" xmlns:ns0="http://www.w3.org/2001/XMLSchema-instance">
This message brought to you by Hello Oracle and the number 100</message>
```

XML Services

Messages for XML services can be constructed as follows:

- The request message is the content of the variable assigned for the request document.
- The content of the request variable must be a single XML document.
- The output document is the response message.

To illustrate how messages are constructed during callouts to XML services, take for example a service callout action configured as follows:

- **Request Document Variable:** `myreq`
- **Response Document Variable:** `myresp`

Assume also that at runtime, the request document variable, `myreq`, is bound to the following XML.

Example - Content of `myreq` Variable

```
<sayHello xmlns="http://www.openuri.org/">
  <intval>100</intval>
  <string>Hello Oracle</string>
</sayHello>
```

In this scenario:

- The outbound message payload is the value of the `myreq` variable, as shown in the previous example.
- The response and the value assigned to the message context variable, `myresp`, is shown in the following example.

Example - Content of `myresp` Variable

```
<m:sayHelloResponse xmlns:m="http://www.openuri.org/">
  <result xsi:type="xsd:string">This message brought to you by Hello Oracle and
the number 100
  </result>
</m:sayHelloResponse>
```

Messaging Services

In the case of Messaging services:

- The request message is the content of the request variable. The content can be simple text, XML, or binary data represented by an instance of `<binary-content ref=... />` reference XML.
- Response messages are treated as binary, so the response variable will contain an instance of `<binary-content ref=... />` reference XML, regardless of the actual content received.

For example, if the request message context variable `myreq` is bound to an XML document of the following format: `<hello>there</hello>`, the outbound message contains exactly this payload. The response message context variable (`myresp`) is bound to a reference element similar to the following:

```
<binary-content ref=" cid:1850733759955566502-2ca29e5c.1079b180f61.-7fd8"/>
```

Using Attachments with Service Callout Messages

You can specify an optional variable to hold attachments for the outbound request and another variable to hold attachments from the outbound response. The Request Attachments and Response Attachments variables are used to specify the request and response attachments respectively.

You can specify attachments for the request messages, response messages, or both. The following code shows the schema type for the Request Attachments and Response Attachments variables:

```
<!-- The schema type for -->
  <complexType name="AttachmentsType">
    <sequence>
      <!-- the 'attachments' variable is just a series of attachment elements
      -->
      <element ref="mc:attachment" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
  <!-- Each attachment in the 'attachments' variable is represented by an instance
  of this element -->
  <element name="attachment" type="mc:AttachmentType"/>
  <complexType name="AttachmentType">
    <all>
      <!-- Set of MIME headers associated with attachment -->
      <element name="Content-ID" type="string" minOccurs="0"/>
      <element name="Content-Type" type="string" minOccurs="0"/>
      <element name="Content-Transfer-Encoding" type="string" minOccurs="0"/>
      <element name="Content-Description" type="string" minOccurs="0"/>
      <element name="Content-Location" type="string" minOccurs="0"/>
      <element name="Content-Disposition" type="string" minOccurs="0"/>
      <!-- Contains the attachment content itself, either in-lined or as
      <binary-content/> -->
      <element name="body" type="anyType"/>
    </all>
  </complexType>
```

The Request Attachments and Response Attachments variables are available regardless of the target service binding type and transport type.

Example of Using Attachments with SOAP-Document Style Services

The following example illustrates a sample WLS web service with a SOAP-document style JWS. The web service has a single method that returns the input argument as its return value. The input argument and the return value are sent and received as attachments.

Example - Sample SOAP-Document Style JWS

```
import javax.activation.DataHandler;

@WebService(name="AttachmentsService", targetNamespace="http://example.org")
@Binding(Binding.Type.SOAP12)
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,
             use=SOAPBinding.Use.LITERAL,
             parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)

@WLHttpTransport(contextPath="testAttachments", serviceUri="AttachmentsService")
public class AttachmentsServiceImpl
```

```

{
    @WebMethod(action="echoAttachmentAction")
    public DataHandler echoAttachment(DataHandler dh)
    {
        return dh;
    }
}

```

The following example illustrates the WSDL document corresponding to the web service in the previous Sample SOAP-Document Style JWS example.

Example - WSDL Document Corresponding to the Web Service

```

<wsdl:definitions name="AttachmentsServiceImplServiceDefinitions"
targetNamespace="http://example.org"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:exam="http://example.org"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">
  <wsdl:types>
    <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://example.org" xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:import namespace="http://www.bea.com/servers/wls90/wsee/attachment"/>
      <xs:element name="echoAttachment">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="dh" type="att:datahandler"
xmlns:att="http://www.bea.com/servers/wls90/wsee/attachment"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="echoAttachmentResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="return" type="att:datahandler"
xmlns:att="http://www.bea.com/servers/wls90/wsee/attachment"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
    <xs:schema targetNamespace="http://www.bea.com/servers/wls90/wsee/attachment"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:complexType name="datahandler">
        <xs:annotation>
          <xs:documentation>Internal type created by WebLogic - Do not
edit!</xs:documentation>
        </xs:annotation>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="echoAttachment">
    <wsdl:part element="exam:echoAttachment" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="echoAttachmentResponse">
    <wsdl:part element="exam:echoAttachmentResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="AttachmentsService">
    <wsdl:operation name="echoAttachment" parameterOrder="parameters">
      <wsdl:input message="exam:echoAttachment"/>
      <wsdl:output message="exam:echoAttachmentResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="AttachmentsServiceImplServiceSoapBinding"

```



```

type="exam:AttachmentsService">
  <soap12:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="echoAttachment">
    <soap12:operation soapAction="echoAttachmentAction" style="document"/>
    <wsdl:input>
      <soap12:body parts="parameters" use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body parts="parameters" use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="AttachmentsServiceImplService">
  <wsdl:port binding="exam:AttachmentsServiceImplServiceSoapBinding"
name="AttachmentsServiceSoapPort">
    <soap12:address
location="http://adc4110062:7001/testAttachments/AttachmentsService"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

Next, the preceding WSDL file is registered in Service Bus and a business service is registered with this WSDL SOAP port. The service callout action is configured as follows:

- **Request Document Variable:** reqDoc
- **Response Document Variable:** respDoc
- **Request Attachments Variable:** reqatt
- **Response Attachments Variable:** respatt

See [Adding Service Callout Actions in the Console](#) for more information on adding and configuring the service callout action.

The following examples show the values of the request document and request attachment context variables when the service callout action is invoked.

Example - Value of the Request Document Variable (reqDoc) at Service Callout Invocation

```

<m:echoAttachment xmlns:m="http://example.org">
  <m:dh href="cid:dh"/>
</m:echoAttachment>

```

Example - Value of the Request Attachments Variable (reqatt) at Service Callout Invocation

```

<con:attachments xmlns:con="http://www.bea.com/wli/sb/context">
  <con:attachment>
    <con:Content-Type>image/jpeg</con:Content-Type>
    <con:Content-ID><dh></con:Content-ID>
    <con:body>
      <con:binary-content ref="cid:-6175a307:131072c66ef:-7f56"/>
    </con:body>
  </con:attachment>
</con:attachments>

```

In the preceding example, the attachment body uses a binary content ref that points to a source stored in the source repository. This can be the result of java callout, MFL transformation, etc.

Note:

Make sure that the href value in the SOAP envelope matches the Content-ID header of the attachment part. This links the body of the message with the corresponding attachment. If the values do not match, you may receive a SOAP Fault from the JWS invocation.

The following example shows a sample outbound payload for the service callout action.

Example - Outbound Payload of the Service Callout

```
POST http://localhost:7001/default/test1

Content-Type: multipart/related;boundary="----=_Part_0_39288954.1310192119320";type="text/xml";start="<soapPart>"

-----=_Part_0_39288954.1310192119320
Content-Type: application/soap+xml; charset=utf-8
Content-Transfer-Encoding: 8bit
Content-ID: <soapPart>

<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header/>
  <env:Body>
    <m:echoAttachment xmlns:m="http://example.org">
      <m:dh href="cid:dh"/>
    </m:echoAttachment>
  </env:Body>
</env:Envelope>

-----=_Part_0_39288954.1310192119320
Content-Type: image/jpeg
Content-ID: <dh>

... binary content of some JPG file ...
-----=_Part_0_39288954.1310192119320--
```

The following example illustrates the JWS response to the service invocation.

Example - Java Web Service (JWS) Response

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: multipart/related;boundary="----=_Part_13_1460020940.1310334461289";type="text/xml";start="<soapPart>"
-----=_Part_13_1460020940.1310334461289
Content-Type: application/soap+xml; charset=utf-8
Content-Transfer-Encoding: 8bit
Content-ID: <soapPart>

<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <m:echoAttachmentResponse xmlns:m="http://example.org">
      <return xmlns="http://example.org" href="cid:return"/>
    </m:echoAttachmentResponse>
```

```

    </env:Body>
  </env:Envelope>

-----=_Part_13_1460020940.1310334461289
Content-Type: image/jpeg
Content-ID: <return>

... binary content of some JPG file ...
-----=_Part_13_1460020940.1310334461289--

```

The following examples show the values of the context variables in the service callout response.

Example - Value of the Response Document Variable (respDoc)

```

<m:echoAttachmentResponse xmlns:m="http://example.org">
  <return xmlns="http://example.org" href="cid:return"/>
</m:echoAttachmentResponse>

```

Example - Value of the Response Attachments Variable (respatt)

```

<con:attachments xmlns:con="http://www.bea.com/wli/sb/context">
  <con:attachment>
    <con:Content-Type>image/jpeg</con:Content-Type>
    <con:Content-ID><return></con:Content-ID>
    <con:body>
      <con:binary-content ref="cid:-6175a307:131072c66ef:-7f57"/>
    </con:body>
  </con:attachment>
</con:attachments>

```

Example of Using Attachments with SOAP RPC Style Service

The following example shows a SOAP RPC-style WSDL file. The WSDL file is registered in Service Bus as a business service. A service callout action is used to invoke this business service.

Example - WSDL Document for the Business Service

```

<?xml version="1.0"?>
<wsdl:definitions xmlns:types="http://example.com/mimetypes"
  xmlns:ref="http://ws-i.org/profiles/basic/1.1/xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  targetNamespace="http://example.com/mimewsd1"
  xmlns:tns="http://example.com/mimewsd1">

  <wsdl:types>
    <xsd:schema targetNamespace="http://example.com/mimetypes"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <xsd:import namespace="http://ws-i.org/profiles/basic/1.1/xsd" />
      <xsd:complexType name="ClaimDetailType">
        <xsd:sequence>
          <xsd:element name="Name" type="xsd:string"/>
          <xsd:element name="ClaimForm" type="ref:swaRef"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>

```

```

</wsdl:types>

<wsdl:message name="ClaimIn">
  <wsdl:part name="ClaimDetail" type="types:ClaimDetailType"/>
  <wsdl:part name="ClaimPhoto" type="xsd:base64Binary"/>
</wsdl:message>

<wsdl:message name="ClaimOut">
  <wsdl:part name="ClaimRefNo" type="xsd:string"/>
</wsdl:message>

<wsdl:portType name="ClaimPortType">
  <wsdl:operation name="SendClaim">
    <wsdl:input message="tns:ClaimIn"/>
    <wsdl:output message="tns:ClaimOut"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="ClaimBinding" type="tns:ClaimPortType">
  <soapbind:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="SendClaim">
    <soapbind:operation soapAction="http://example.com/soapaction"/>
    <wsdl:input>
      <mime:multipartRelated>
        <mime:part>
          <soapbind:body use="literal"
            parts="ClaimDetail"
            namespace="http://example.com/mimetypes"/>
        </mime:part>
        <mime:part>
          <mime:content part="ClaimPhoto"
            type="image/jpeg"/>
        </mime:part>
      </mime:multipartRelated>
    </wsdl:input>
    <wsdl:output>
      <soapbind:body use="literal"
        namespace="http://example.com/mimetypes"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
</wsdl:definitions>

```

The service callout action is configured as follows:

- **Request Parameter:** claimDetail
- **Response Parameter:** claimRefNo
- **Request Attachments Variable:** reqatt
- **Response Attachments Variable:** respatt

See [Adding Service Callout Actions in the Console](#) and [Adding Service Callout Actions in JDeveloper](#) for more information on adding and configuring the service callout action.

The following examples show the values of the message context variables when the service callout is invoked.

Example - Value of the Request Parameter (claimDetail)

```
<ClaimDetail xmlns=" http://example.com/mimetypes">
  <Name>...</Name>
  <ClaimForm>cid:claimform@example.com</ClaimForm>
</ClaimDetail>
```

Example - Value of the Request Attachment Variable (reqatt)

```
<con:attachments xmlns:con="http://www.bea.com/wli/sb/context">
  <con:attachment>
    <con:Content-Type>text/xml</con:Content-Type>
    <con:Content-ID><claimform@example.com></con:Content-ID>
    <con:body>
      <form>...XML contents of claim form referenced by the swaRef... </form>
    </con:body>
  </con:attachment>
  <con:attachment>
    <con:Content-Type>image/jpeg</con:Content-Type>
    <con:Content-ID><ClaimPhoto=4d7a5fa2-14af-451c-961b-5c3abf786796@example.com></
con:Content-ID>
    <con:body>
      <con:binary-content ref="cid:-6175a307:131072c66ef:-7f58"/>
    </con:body>
  </con:attachment>
</con:attachments>
```

The following example shows a sample outbound request.

Example - Sample Outbound Request

```
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
start="<rootpart@example.com>"

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <rootpart@example.com>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body xmlns:types="http://example.com/mimetypes">
    <types:SendClaim>
      <ClaimDetail>
        <Name>...</Name>
        <ClaimForm>cid:claimform@example.com</ClaimForm>
      </ClaimDetail>
    </types:SendClaim>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: text/xml
Content-Transfer-Encoding: 8bit
Content-ID: <claimform@example.com>

...claim form referenced by the swaRef...

--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <ClaimPhoto=4d7a5fa2-14af-451c-961b-5c3abf786796@example.com>
```

```
...MIME attachment of binary photograph...  
--MIME_boundary--
```

The following example shows a sample response.

Example - Sample Response

```
Content-Type: text/xml; charset=UTF-8  
  
<?xml version='1.0' ?>  
<SOAP-ENV:Envelope  
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"  
  <SOAP-ENV:Body xmlns:types="http://example.com/mimetypes">  
    <types:SendClaimResponse>  
      <ClaimRefNo>.....</ClaimRefNo>  
    </types:SendClaimResponse>  
  </SOAP-ENV:Body>  
</SOAP-ENV:Envelope>
```

The message context variables (`claimRefNo` and `respatt`) corresponding to the response get set. The following example shows the value of the `respatt` variable.

Example - Value of the Response Attachments Variable (`respatt`)

```
<con:attachments xmlns:con=http://www.bea.com/wli/sb/context />
```

MTOM/XOP Support

You can use SOAP Message Transmission Optimization Mechanism (MTOM) with a service callout. MTOM uses XML-binary Optimized Packaging (XOP) to transfer the binary data.

However, you cannot mix MTOM and attachments. If the target service supports MTOM and there are attachments in the outbound request, then you get the following error:

```
"Mixing of XOP/MTOM and attachments is not allowed"
```

If the target service is MTOM-enabled, you should not configure any attachment variables.

Page Attachments to Disk

Service callouts support page attachments to disk. It is possible to use the contents of a previously paged attachment for the outbound request.

When the service callout is processing the outbound response, any business service configured to page attachments to disk is invoked accordingly.

Handling Errors as the Result of a Service Callout

You can configure error handling at the message flow, pipeline, route node, and stage level. The types of errors that are received from an external service as the result of a service callout include transport errors, SOAP faults, responses that do not conform to an expected response, and so on.

The `fault` context variable is set differently for each type of error returned. You can build your business and error handling logic based on the content of the `fault` variable. To learn more about `$fault`, see [Fault Variable](#).

Transport Errors

When a transport error is received from an external service and there is no error response payload returned to Service Bus by the transport provider (for example, if an HTTP business service accepts response types other than XML or SOAP and therefore cannot receive HTTP response codes), the service callout action throws an exception, which in turn causes the pipeline to raise an error. The fault variable in a user-configured error handler is bound to a message formatted similarly to that shown in the following example.

Example - Contents of the fault Variable—Transport Error, no Error Response Payload

```
<con:fault xmlns:con="http://www.bea.com/wli/sb/context">
  <con:errorCode>BEA-380000</con:errorCode>
  <con:reason>Not Found</con:reason>
  <con:details>
    .....
  </con:details>
  <con:location>
    <con:node>PipelinePairNode1</con:node>
    <con:Pipeline>PipelinePairNode1_request</con:Pipeline>
    <con:Stage>Stage1</con:Stage>
  </con:location>
</con:fault>
```

In the case that there is a payload associated with the transport error—for example, when an HTTP 500 error code is received from the business service and there is XML payload in the response—a message context fault is generated with the custom error code: BEA-382502.

The following conditions must be met for a BEA-382502 error response code to be triggered as the result of a response from a service—when that service uses an HTTP or JMS transport:

- (HTTP) The response code must be any code 300 or greater.
- (JMS) The response must have a property set to indicate that it is an error response—the transport metadata status code set to 1 indicates an error.
- The content type must be text/xml, application/*any_string*+xml, or multipart/related.
- If the service is AnySoap or WSDL-based SOAP, then it must have a SOAP envelope. The body inside the SOAP envelope must be XML format; it cannot be text.
- If the service type is AnyXML, or a messaging service of type text returns XML content with a non-successful response code (any code other than 200 or 202).

If the transport is HTTP, the `ErrorResponseDetail` element will also contain the HTTP error code returned with the response. The `ErrorResponseDetail` element in the fault contains error response payload received from the service. The following example shows an example of the `ErrorResponseDetail` element.

Example - Contents of the fault Variable—Transport Error, with Error Response Payload

```
<ctx:Fault xmlns:ctx="http://www.bea.com/wli/sb/context">
  <ctx:errorCode>BEA-382502</ctx:errorCode>
```

```

    <ctx:reason> Service callout has received an error response from the server</
ctx:reason>
    <ctx:details>
      <alsb:ErrorResponseDetail xmlns:alsb="http://www.oracle.com/...">
        <alsb:detail> <![CDATA[
. . .
]]>
          </alsb:detail>          <alsb:http-response-code>500</alsb:http-
response-code>
        </alsb:ErrorResponseDetail>
      </ctx:details>
      <ctx:location>. . .</ctx:location>
    </ctx:Fault>

```

Note:

The XML schema for the service callout-generated fault is shown in [Unexpected Responses](#).

SOAP Faults

In case an external service returns a SOAP fault, the Service Bus runtime sets up the context variable `$fault` with a custom error code and description with the details of the fault. To do so, the contents of the 3 elements under the `<SOAP-ENV:Fault>` element in the SOAP fault are extracted and used to construct a Service Bus fault element.

Take for example a scenario in which a service returns the following error.

Example - SOAP Fault Returned From Service Callout

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Client</faultcode>
      <faultstring>Application Error</faultstring>
      <detail>
        <message>That's an Error!</message>
        <errorcode>1006</errorcode>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The `<faultcode>`, `<faultstring>`, and `<detail>` elements are extracted and wrapped in an `<alsb:ReceivedFault>` element. Note that the `faultcode` element in the example in [Unexpected Responses](#) contains a QName—any related namespace declarations are preserved. If the transport is HTTP, the `ReceivedFault` element will also contain the HTTP error code returned with the fault response.

The generated `<alsb:ReceivedFault>` element, along with the custom error code and the error string are used to construct the contents of the `fault` context variable, which in this example takes a format similar to that shown in the previous example.

Example - Contents of the Service Bus Fault Variable—SOAP Fault

```

<ctx:Fault xmlns:ctx="http://www.bea.com/wli/sb/context">
  <ctx:errorCode>OSB-382500<ctx:errorCode>
  <ctx:reason> service callout received a soap Fault response</ctx:reason>
  <ctx:details>

```



```

<alsb:ReceivedFault xmlns:alsb="http://www.oracle.com/...">
  <alsb:faultcode
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">SOAP-ENV:Client
  </alsb:faultcode>
  <alsb:faultstring>Application Error</alsb:faultstring>
  <alsb:detail>
    <message>That's an Error!</message>
    <errorcode>1006</errorcode>
  </alsb:detail>
  <alsb:http-response-code>500</alsb:http-response-code>
</alsb:ReceivedFault>
</ctx:details>
<ctx:location> </ctx:location>
</ctx:Fault>

```

Note:

The unique error code OSB-382500 is reserved for the case when service callout actions receive SOAP fault responses.

When chaining local proxy services, SOAP fault details are not propagated from one pipeline to the next in the \$fault variable. To propagate SOAP faults from proxy service to proxy service, use an error handler with a Reply with Failure action, as described in Propagating SOAP Faults Between Proxy Services in the *Developing Services with Oracle Service Bus*.

Unexpected Responses

When a service returns a response message that is not what the proxy service runtime expects, a message context fault will be generated and initialized with the custom error code OSB-382501. The details of the fault include the contents of the SOAP-Body element of the response. If the transport is HTTP, the `ReceivedFault` element will also contain the HTTP error code returned with the fault response.

The XML schema definition of the service callout-generated fault details is shown in the following example.

Example - XML Schema for the Service Callout-Generated Fault Details

```

<xs:complexType name="ReceivedFaultDetail">
  <xs:sequence>
    <xs:element name="faultcode" type="xs:QName"/>
    <xs:element name="faultstring" type="xs:string"/>
    <xs:element name="detail" minOccurs="0" >
      <xs:complexType>
        <xs:sequence>
          <xs:any namespace="##any" minOccurs="0"
maxOccurs="unbounded" processContents="lax" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="http-response-code" type="xs:int" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="UnrecognizedResponseDetail">
  <xs:sequence>

```

```

        <xs:element name="detail" minOccurs="0" type="xs:string" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="ErrorResponseDetail">
    <xs:sequence>
        <xs:element name="detail" minOccurs="0" type="xs:string" />
    </xs:sequence>
</xs:complexType>

```

Handling Errors in Pipelines

The process described in the next paragraph constitutes an error handling pipeline for the stage of an error handler. In addition, an error pipeline can be defined for a pipeline pair request or response, or for an entire pipeline.

The error handler at the stage level is invoked for handling an error; If the stage-level error handler is not able to handle a given type of error, the pipeline error handler is invoked. If the pipeline-level error handler also fails to handle the error, the service-level error handler is invoked. If the service-level error handler also fails, the error is handled by the system. [Table 12-9](#) summarizes the scope of the error handlers at various levels in the pipeline.

Table 12-9 Scope of Error Handlers

Level	Scope
Stage	Handles all the errors within a stage.
Pipeline	Handles all the errors in a pipeline, along with any unhandled errors from any stage in a pipeline.
Service	Handles all the errors in a pipeline, along with any unhandled errors in any pipeline pair requests or responses in a service. Note: All WS-Security errors are handled at this level.
System	Handles all the errors that are not handled anywhere else in a pipeline.

Note:

There are exceptions to the scope of error handlers. For example, an exception thrown by a non-XML transformation at the stage level is only caught by the service-level error handler. Suppose a transformation occurs that transforms XML to MFL for an outgoing proxy service response message, it always occurs in the binding layer. Therefore, for example, if a non-XML output is missing a mandatory field at the stage level, only a service-level error handler can catch this error.

You can handle errors by configuring a test that checks if an assertion is true and use the reply action configured false. You can repeat this test at various levels. Also you can have an error without an error handler at a lower level and handle it through an error handler at a higher level in the pipeline.

In general, it is easier to handle specific errors at a stage level of the pipeline and use error handlers at the higher level for more general default processing of errors that are

not handled at the lower levels. It is good practice to explicitly handle anticipated errors in the pipelines and allow the service-level handler to handle unanticipated errors.

Generating the Error Message, Reporting, and Replying

A predefined context variable (the `fault` variable) is used to hold information about any error that occurs during message processing. When an error occurs, this variable is populated with information before the appropriate error handler is invoked. The `fault` variable is defined only in error handler pipelines and is not set in request and response pipelines, or in route or branch nodes. For additional information about `$fault`, see [Predefined Context Variables](#).

In the event of errors for request/response type inbound messages, it is often necessary to send a message back to the originator outlining the reason why an error occurred. You can accomplish this by using a *Reply with Failure* action after configuring the message context variables with the response you want to send. For example, when an HTTP message fails, *Reply with Failure* generates the HTTP 500 status. When a JMS message fails, *Reply with Failure* sets the `JMS_BEA_Error` property to true.

An error handling pipeline is invoked if a service invoked by a proxy service returns a SOAP fault or transport error. Any received SOAP fault is stored in `$body`, so if a *Reply with Failure* is executed without modifying `$body`, the original SOAP fault is returned to the client that invoked the service. If a reply action is not configured, the system error handler generates a new SOAP fault message. The proxy service recognizes that a SOAP fault is returned because an HTTP error status is set, or the JMS property `SERVER_Error` is set to true.

Some use cases require error reporting. You can use the report action in these situations. For example, consider a scenario in which the request pipeline reports a message for tracking purposes, but the service invoked by the route node fails after the reporting action. In this case, the reporting system logged the message, but there is no guarantee that the message was processed successfully, only that the message was successfully received.

You can use the Oracle Service Bus Console to track the message to obtain an accurate picture of the message flow. This allows you to view the original reported message indicating the message was submitted for processing, and also the subsequent reported error indicating that the message was not processed correctly. To learn how to configure a report action and use the data reported at runtime, see [Working with Pipeline Actions in Oracle Service Bus Console](#).

Different Behavior of Security Fault Handling in Service Bus 11g and 12c

Service Bus 12c handles security faults in the message flow differently than in Service Bus 11g.

In Service Bus 11g, any security errors raised as a result of failed OWSM policies or custom token message level authentication on inbound requests can be handled by a user-configured service error handler in the pipeline message flow. For example, on a service with username-token authentication policy, any authentication failures trigger a service-level error handler, if one is configured.

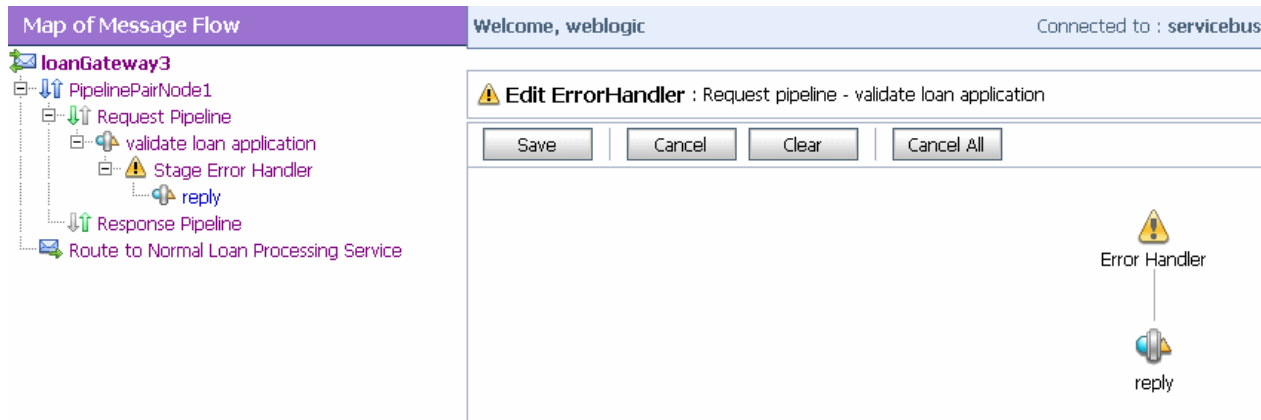
In Service Bus 12c, any security errors raised by security processing of inbound requests is not handled in the pipeline service-level error handler; rather, this results in a SOAP Fault automatically generated by the default inbound system error handler.

These faults cannot be customized. It is not possible to route the failed request to the next component (for example, a pipeline or business service).

Example of Action Configuration in Error Handlers

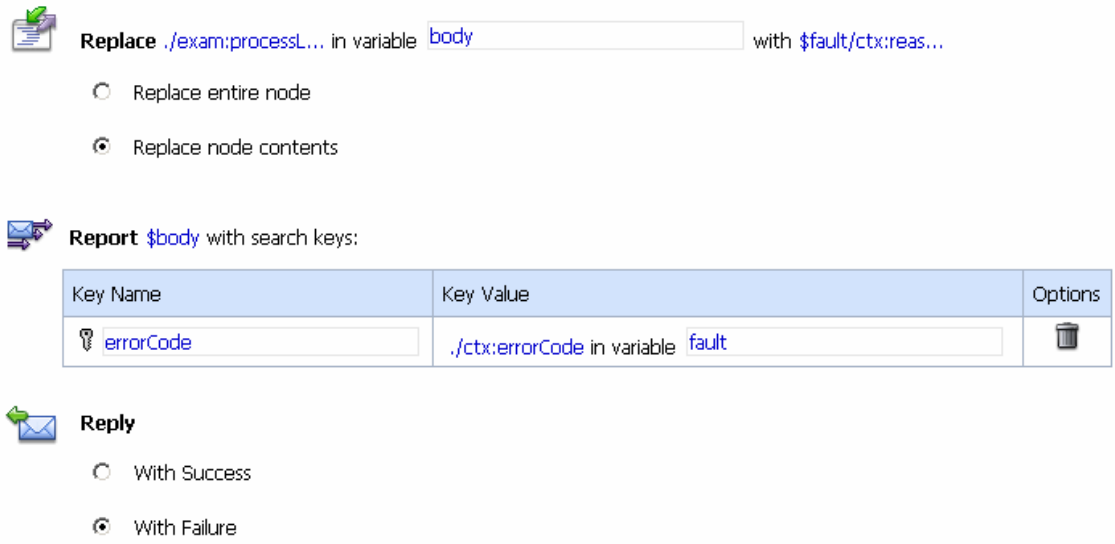
This example shows how you can configure the report and reply actions in error handlers. The pipeline shown in [Figure 12-1](#) includes an error handler on the `validate loan application` stage. The error handler in this case is a simple message flow with a single stage configured—it is represented in the Oracle Service Bus Console as shown in [Figure 12-1](#).

Figure 12-1 Error Handler Pipeline



The stage is, in turn, configured with actions (replace, report, and reply) as shown in [Figure 12-2](#).

Figure 12-2 Actions in Stage Error Handler



The actions control the behavior of the stage in the pipeline error handler as follows:

- Replace:** The contents of a specified element of the body variable are replaced with the contents of the `fault` context variable. The body variable element is specified by an XPath expression. The contents are replaced with the value returned by an XQuery expression—in this case `$fault/ctx:reason/text()`

- **Report:** Messages from the reporting action are written to the Service Bus reporting data stream if the error handler configured with this action is invoked. The JMS Reporting Provider reports the messages on the Service Bus Dashboard in Fusion Middleware Control. Service Bus provides the capability to deliver message data to one or more reporting providers. Message data is captured from the body of the message and from any other variables associated with the message, such as header or inbound variables. You can use the message delivered to the reporting provider for functions such as tracking messages or regulatory auditing.

When an error occurs, the contents of the fault context variable are reported. The key name is `errorCode`, and the key value is extracted from the fault variable using the following XPath expression: `./ctx:errorCode`. Key/value pairs are the key identifiers that identify these messages in the Dashboard at runtime.

To configure a report action and use the data reported at runtime, see [Working with Pipeline Actions in Oracle Service Bus Console](#).

- **Reply:** At runtime, an immediate reply is sent to the invoker of the `loanGateway3` proxy service (see [Figure 12-2](#)) indicating that the message had a fault. The reply is `With Failure`.

Using Dynamic Routing

When you do not know the service you need to invoke from the pipeline you are creating, you can use dynamic routing. For any given pipeline, you can use one of the following techniques to dynamically route messages:

- In a message flow pipeline, design an XQuery expression to dynamically set the fully qualified service name in Service Bus and use the dynamic route or dynamic publish actions.

Note:

Dynamic Routing can be achieved in a route node, whereas dynamic publishing can be achieved in a stage in a request pipeline or a response pipeline.

With this technique, the pipeline dynamically uses the service account of the endpoint business service to send user names and passwords in its outbound requests. For example, if a pipeline is routing a request to Business Service A, then the invoking proxy service uses the service account from Business Service A to send user names and passwords in its outbound request. See [Implementing Dynamic Routing](#).

- Configure a pipeline to route or publish messages to a business service. Then, in the request actions section for the route action or publish action, add a Routing Options action that dynamically specifies the URI of a service.

With this technique, to send user names and passwords in its outbound requests, the proxy service uses the service account of the statically defined business service, regardless of the URI to which the request is actually sent.

For information on how to use this technique, see [Implementing Dynamic Routing](#).

Note:

This technique is used when the overview of the interface is fixed. The overview of the interface includes message types, port types, and binding, and excludes the concrete interface. The concrete interface is the transport URL at which the service is located.

For a working example of dynamic service invocation, see the Service Bus samples at <http://www.oracle.com/technetwork/middleware/service-bus/learnmore/index.html>.

Implementing Dynamic Routing

You can use dynamic routing to determine the destination during the runtime of a pipeline. To achieve this you can use a routing table in an XML file to create an XQuery resource.

Note:

Instead of using the XQuery resource, you can also directly use the XML file from which the resource is created.

An XML file or the XQuery resource can be maintained easily. At runtime you provide the entry in the routing table that will determine the routing or publishing destination of the pipeline. The XML file or the XQuery resource contains a routing table, which maps a logical identifier (such as the name of a company) to the physical identifier (the fully qualified name of the service in Service Bus). The logical identifier, which is extracted from the message, maps on to the physical identifier, which is the name of the service you want to invoke.

Note:

To use the dynamic route action, you need the fully qualified name of the service in Service Bus.

In a pipeline the logical identifier is obtained with an XPath into the message. You assign the XML table in the XQuery resource to a variable. You implement a query against the variable in the routing table to extract the physical identifier based on the corresponding logical identifier. Using this variable you will be able to invoke the required service. The following sections describe how to implement dynamic routing.

- [Sample XML File](#)
- [Creating an XQuery Resource From the Sample XML](#)
- [Creating and Configuring the Pipeline to Implement Dynamic Routing](#)
- [Guidelines for Implementing Identity-Based Routing](#)

Sample XML File

You can create an XQuery resource from the following XML file. Save this as sampleXquery.xml.

Example - Sample XML File

```
<routing>
  <row>
    <logical>Oracle</logical>
    <physical>default/goldservice</physical>
  </row>
  <row>
    <logical>ABC Corp</logical>
    <physical>default/silverservice</physical>
  </row>
</routing>
```

Creating an XQuery Resource From the Sample XML

To create an XQuery resource from the sample XML in the Oracle Service Bus Console:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.
2. In the left pane of the Oracle Service Bus Console window, click the **Resources** tab. The Project Navigator appears.
3. Expand the **All Projects** node by clicking the **Expand** (arrow) icon before it.
4. Click the project name to which you wish to add the XQuery resource.
5. From the **Create** drop-down list, select **XQuery**. The Create XQuery dialog appears.
6. In the **Resource Name** field, enter the name of the resource. This is a mandatory.
7. In the **Resource Description** field, provide the a description for the resource. This is optional.
8. Click **Choose File** to select the XML file you are using as the XQuery resource.
9. Click **Create** to create the XQuery resource.
10. Activate the session.

Creating and Configuring the Pipeline to Implement Dynamic Routing

To implement dynamic routing with a pipeline:

1. In the Project Navigator, select the project to which you want to add a pipeline, and then click the down arrow next to the **Create** icon.
2. Select **Pipeline** from the list of options.
The Create Pipeline dialog appears.
3. In the **Pipeline Name** field of the **General** section, enter the name of the pipeline. This is mandatory. Optionally, specify a **Description** for the pipeline.
4. Select the **Service Type** for the pipeline. For more information on selecting the service type, see [Service Types and Protocols for Proxy Services](#).
5. Select **Expose as Proxy Service** to create a proxy service corresponding to the pipeline message flow. Specify the name and other details for the proxy service to be created.

6. Click **Create** to create the pipeline resource. The pipeline is created and opened up for editing.
7. Click the **Open Message Flow** icon near the top right corner of the window. The **Edit Message Flow** page appears. The message flow initially consists of a single icon.
8. Click the start node (pipeline icon), and select **Add Pipeline Pair** to add a pipeline pair to the message flow.
9. Click **Request Pipeline** icon select **Add Stage** from the menu.
10. Click the Stage1 icon to and select **Edit Stage** from the menu. The **Edit Stage Configuration** page appears.
11. Click **Add Action** icon. Choose **Add an Action** item from the menu.
12. Choose the **Assign** action from **Message Processing**.
13. Click **Expression**. The **XQuery Expression Editor** is displayed.
14. Click **XQuery Resources**. The browser displays the page where you can import the XQuery resource. Click the **Browse** to locate the XQuery resource.
15. Click **Validate** to validate the imported XQuery resource.
16. Save the imported XQuery resource on successful validation.
17. On the **Edit Stage Configuration** page, enter the name of the variable in the field.

This assigns the XQuery resource to this variable. The variable now contains the externalized routing table.
18. Add another **Assign** action.
19. Enter the following XQuery:

```
<ctx: route>
<ctx: service isProxy='false'> {$routingtable/row[logical/text()=
$loglogicalidentifier]/physical/text()}
</ctx: service>
</ctx: route>
```


In the above code, replace `$loglogicalidentifier` by the actual XPath to extract the logical identifier from the message (example from `$body`).
20. Click **Validate** to validate the XQuery.
21. Save the XQuery on successful validation.
22. On the **Edit Stage Configuration** page, enter the name of the variable (for example, *routeresult*) in the field.

This extracts the XML used by the dynamic route action into this variable.
23. Click the pipeline icon to add a route node to the end of the pipeline.
24. Click the **Route Node** icon and select **Edit** from the menu.
25. Click the **Add Action** icon. Choose **Add an Action** item from the menu.

26. Choose the **Dynamic Route** action.
27. Click **Expression**. The XQuery Expression Editor is displayed.
28. Enter the variable; for example, `$routeresult`.

Guidelines for Implementing Identity-Based Routing

If you want to dynamically route message based on the identity of an authenticated user, Service Bus stores information such as user name, group membership (/principals/group), and the name of the subject (/subject-properties/property/name) in the following inbound context variables:

- `$inbound/ctx:security/ctx:transportClient/*`
- `$inbound/ctx:security/ctx:messageLevelClient/*`

For more information on these context variables, see [Table A-6](#).

Using the guidance provided in [Using Dynamic Routing](#), use XQuery or simple XML to map authenticated user identity characteristics to different endpoints using the desired mapping technique.

The following predefined Service Bus XQuery functions are also available to perform security checks in identity-based routing:

- [fn-bea:lookupBasicCredentials](#)
- [fn-bea:isUserInGroup](#)
- [fn-bea:isUserInRole](#)

For a working example of dynamic service invocation, see the Service Bus samples at <http://www.oracle.com/technetwork/middleware/service-bus/learnmore/index.html>.

Accessing Databases Using XQuery

Service Bus provides read-access to databases from pipelines without requiring you to write a custom EJB or custom Java code and without the need for a separate database product like Oracle Data Service Integrator. You can use the `execute-sql()` function to make a simple JDBC call to a database to perform simple database reads. Any SQL query is legal, from a query that gets a single tax rate for the supplied location to a query that does a complex join to obtain an order's current status from several underlying database tables.

A database query can be used to get data for message enrichment, for routing decisions, or for customizing the behavior of a pipeline. Take for example a scenario in which a Service Bus pipeline receives "request for quote" messages. The pipeline can route the requests based on the customer's priority to one of a number of quotation business services (say, standard, gold, or platinum level services). The pipeline can then perform a SQL-based augmentation of the results that those services return—for example, based on the selected ship method and the weight of the order, the shipping cost can be looked up and that cost added to the request for quote message.

[fn-bea:execute-sql\(\)](#) describes the syntax for the function and provides examples of its use. The `execute-sql()` function returns typed data and automatically translates values between SQL/JDBC and XQuery data models.

You can store the returned element in a user-defined variable in a Service Bus pipeline.

The following databases and JDBC drivers are supported using the `execute-sql()` function:

- The sample database provided by WebLogic Server.
IBM DB2/NT 8
- Microsoft SQL Server 2000, 2005
- Oracle⁹ⁱ, Oracle Database 10g, Oracle Database 11g, Oracle Database 12c
- Sybase 12.5.2 and 12.5.3
- WebLogic Type 4 JDBC drivers
- Third-party drivers supported by WebLogic Server

Use non-XA drivers for datasources you use with the `fn-bea:execute-sql()` function—the function supports read-only access to the datasources.

Caution:

In addition to specifying a non-XA JDBC driver class to use to connect to the database, you must ensure that you disable global transactions and two-phase commit. (Global transactions are enabled by default in the Oracle WebLogic Server Administration Console for JDBC data sources.) These specifications can be made for your data source using the Oracle WebLogic Server Administration Console. See "Create JDBC Data Sources" in the *Oracle WebLogic Server Administration Console Online Help*.

For complete information about database and JDBC drivers support in Service Bus, see *Oracle Fusion Middleware Supported System Configurations* at:

<http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html>

Databases other than the core set described in the preceding listing are also supported. However, for the core databases listed above, the XQuery engine does a better recognition and mapping of data types to XQuery types than it does for the non-core databases—in some cases, a core database's proprietary JDBC extensions are used when fetching data. For the non-core databases, the XQuery engine relies totally on the standard type codes provided by the JDBC driver and standard JDBC resultset access methods.

When designing your pipeline, you can enter XQueries inline as part of an action definition instead of entering them as resources. You can also use inline XQueries for conditions in If Then actions in pipeline. For information about using the inline XQuery editor, see [Creating Variable Structure Mappings](#).

Understanding Message Context

The message context is a set of variables that hold message context and information about messages as they are routed through Service Bus. Together, the `header`, `body`, and `attachments` variables, (referenced as `$header`, `$body` and `$attachments` in XQuery statements) represent the message as it flows through Service Bus. The

canonical form of the message is SOAP. Even if the service type is not SOAP, the message appears as SOAP in the Service Bus message context.

[Table 12-10](#) describes the Service Bus message context variables.

Table 12-10 Predefined Context Variables in Service Bus

Context Variable	Description	See Also
header	For SOAP messages, <code>\$header</code> contains the SOAP header. If the pipeline is SOAP 1.2, <code>\$header</code> contains a SOAP 1.2 Header element. For message types other than SOAP, <code>\$header</code> contains an empty SOAP header element.	Message-Related Variables
body	This varies depending on the message type, as described below: <ul style="list-style-type: none"> • SOAP messages: The <code><SOAP:Body></code> part extracted from the SOAP envelope. If the pipeline is SOAP 1.2, the <code>\$body</code> variable contains a SOAP 1.2 Body element. • Non-SOAP, non-binary messages: The entire message content wrapped in a <code><SOAP:Body></code> element. • Binary messages: A <code><SOAP:Body></code> wrapped reference to an in-memory copy of the binary message. • Java objects: A <code><SOAP:Body></code> wrapped reference to an in-memory copy of the Java object. 	Message-Related Variables
attachments	The MIME attachments for a given message.	Message-Related Variables
inbound	The inbound transport headers along with information about the proxy service that received a message.	Inbound and Outbound Variables
outbound	The outbound transport headers along with information about the target service to which a message is to be sent.	Inbound and Outbound Variables
operation	The operation being invoked on a pipeline.	Operation Variable
fault	Information about errors that have occurred during the processing of a message.	Fault Variable
messageId	The transport provider-specific message identifier. This ID should uniquely identify the message among other messages going through the Service Bus runtime, but it is not required that this value be unique.	messageID Variable

Message Context Components

In a Message Context, `$header` contains a SOAP header element and `$body` contains a SOAP Body element. The Header and Body elements are qualified by the SOAP 1.1

or SOAP 1.2 namespace depending on the service type of the pipeline. Also in a Message Context, `$attachments` contains a wrapper element called `attachments` with one child attachment element per attachment. The attachment element has a `body` element with the actual attachment.

When a message is received by a pipeline, the message contents are used to initialize the `header`, `body`, and `attachments` variables. For SOAP services, the `Header` and `Body` elements are taken directly from the envelope of the received SOAP message and assigned to `$header` and `$body` respectively. For non-SOAP services, the entire content of the message is typically wrapped in a `Body` element (qualified by the SOAP 1.1 namespace) and assigned to `$body`, and an empty `Header` element (qualified by the SOAP 1.1 namespace) is assigned to `$header`.

Binary and MFL messages are initialized differently. For MFL messages, the equivalent XML document is inserted into the `Body` element that is assigned to `$body`. For binary messages, the message data is stored internally and a piece of reference XML is inserted into the `Body` element that is assigned to `$body`. The reference XML looks like `<binary-content ref="..." />`, where `"..."` contains a unique identifier assigned by the pipeline.

The message context is defined by an XML schema. You must use XQuery expressions to manipulate the context variables in the pipeline. The predefined context variables provided by Service Bus can be grouped into the following types:

- Message-related variables
- Inbound and outbound variables
- Operation variable
- Fault variable

For information about the predefined context variables, see [Predefined Context Variables](#).

The `$body` contains message payload variable. When a message is dispatched from Service Bus you can decide the variables, whose you want to include in the outgoing message. That determination is dependent upon whether the target endpoint is expecting a SOAP or a non-SOAP message:

- For a binary, any text or XML message content inside the `Body` element in `$body` is sent.
- For MFL messages, the `Body` element in `$body` contains the XML equivalent of the MFL document.
- For text messages, the `Body` element in `$body` contains the text. For text attachments, the `body` element in `$attachments` contains the text. If the contents are XML instead of simple text, the XML is sent as a text message.
- For XML messages, the `Body` element in `$body` contains the XML. For XML attachments, the `body` element in `$attachments` contains the XML.
- SOAP messages are constructed by wrapping the contents of the header and body variables inside a `<soap:Envelope>` element. (The SOAP 1.1 namespace is used for SOAP 1.1 services, while the SOAP 1.2 namespace is used for SOAP 1.2 services.) If the body variable contains a piece of reference XML, it is sent. That is the referenced content is not substituted in the message.

For non-SOAP services, if the Body element of `$body` contains a binary-content element, then the referenced content stored internally is sent 'as is', regardless of the target service type.

For more information, see [Message Context](#).

The types for the message context variables are defined by the message context schema (`MessageContext.xsd`). When working with the message context variables in the Oracle XQuery Mapper, you need to reference `MessageContext.xsd`, which is available in a JAR file, `OSB_ORACLE_HOME/lib/sb-schemas.jar`, and the transport-specific schemas, which are available at

`OSB_ORACLE_HOME/lib/transport/`

To learn about the message context schema and the transport specific schemas, see [Message Context Schema](#).

Guidelines for Viewing and Altering Message Context

Consider the following guidelines when you want to inspect or alter the message context:

- In an XQuery expression, the root element in a variable is not present in the path in a reference to an element in that variable. For example, the following XQuery expression obtains the `Content-Description` of the first attachment in a message:

```
$attachments/ctx:attachment[1]/ctx:content-Description
```

To obtain the second attachment

```
$attachments/ctx:attachment[2]/ctx:body/*
```

- A context variable can be empty or it can contain a single XML element or a string value. However, an XQuery expression often returns a sequence. When you use an XQuery expression to assign a value to a variable, only the first element in the sequence returned by the expression is stored as the variable value. For example, if you want to assign the value of a WS-Addressing Message ID from a SOAP header (assuming there is one in the header) to a variable named `idvar`, the assign action specification is:

```
assign data($header/wsa:messageID to variable idvar
```

Note:

In this case, if two WS-Addressing MessageID headers exist, the `idvar` variable will be assigned the value of the first one.

- The variables `$header`, `$body`, and `$attachments` are never empty. However, `$header` can contain an empty SOAP Header element, `$body` can contain an empty SOAP Body element, and `$attachments` can contain an empty attachment element.
- In cases in which you use a transformation resource (XSLT or XQuery), the transformation resource is defined to transform the document in the SOAP body of a message. To make this transformation case easy and efficient, the input parameter to the transformation can be an XQuery expression. For example, you can use the following XQuery expression to feed the business document in the Body element of a message (`$body`) as input to a transformation:

```
$body/* [1]
```

The result of the transformation can be put back in `$body` with a replace action. That is replace the content of `$body`, which is the content of the `Body` element. For more information, see [Transforming Data with XQuery](#) and [Transforming Data with XSLT](#).

- In addition to inserting or replacing a single element, you can also insert or replace a selected sequence of elements using an insert or replace action. You can configure an XQuery expression to return a sequence of elements. For example, you can use insert and replace actions to copy a set of transport headers from `$inbound` to `$outbound`. For information on adding an action, see [Adding and Editing Pipeline Actions in the Console](#). For an example, see [Copying JMS Properties From Inbound to Outbound](#).

Copying JMS Properties From Inbound to Outbound

It is assumed that the interfaces of the proxy services and of the invoked business service may be different. Therefore, Service Bus does not propagate any information (such as the transport headers and JMS properties) from the inbound variable to the outbound variable.

The transport headers for the proxy service's request and response messages are in `$inbound` and the transport headers for the invoked business service's request and response are in `$outbound`.

For example, the following XQuery expression can be used in a case where the user-defined JMS properties for a one-way message (an invocation with no response) need to be copied from inbound message to outbound message:

Use the transport headers action to set

```
$inbound/ctx:transport/ctx:request/tp:headers/tp:user-header
```

as the first child of:

```
./ctx:transport/ctx:request/tp:headers
```

in the outbound variable.

To learn how to configure the transport header action, see [Adding Transport Header Actions in the Console](#).

Working with Variable Structures

This section includes the following topics:

- [Using the Inline XQuery Expression Editor](#)
- [Using Variable Structures](#)

You can find examples at [Creating Variable Structure Mappings](#).

Using the Inline XQuery Expression Editor

Service Bus allows you to import XQueries that have been created with an external tool such as the Oracle XQuery Mapper. You can use these XQueries anywhere in the pipeline by binding the XQuery resource input to an Inline XQuery, and binding the XQuery resource output to an action that uses the result as the input; for example, the assign, replace, or insert actions.

However, you can enter the XQuery inline as part of the action definition instead of entering the XQuery as a resource. You can also use Inline XQueries for the condition in an **If...Then...** action.

Use the Inline XQuery Expression Editor to enter simple XQueries that consist of the following:

- Fragments of XML with embedded XQueries.
- Simple variable paths along the child axis.

Note:

For more complex XQueries, it is recommended that you use the XQuery Mapper, especially if you are not familiar with XQuery.

Inline XQueries can be used effectively to:

- Create variable structures by using the Inline XQuery Expression Editor. See [Using Variable Structures](#).
- Extract or access a business document or RPC parameter from the SOAP envelope elements in `$header` or `$body`.
- Extract or access an attachment document in `$attachments`.
- Set up the parameters of a service callout action by extracting it from the SOAP envelope.
- Insert the result parameter of a service callout action into the SOAP envelope.
- Extract a sequence from the SOAP envelope to drive a `for` loop.
- Update an item in the sequence in a `for` loop with an Update action.

Note:

You can also use the Inline XQuery Expression Editor to create variable structures. For more information, see [Using Variable Structures](#).

Inline XQueries

The inline XQuery and XPath editors allow you to declare a variable's structure by mapping it to a type or element and then creating path expressions with a drag and drop action from the graphical representation of the structure. You can also enter the path expressions manually.

You can use this feature directly for all user-defined variables, as well as `$inbound`, `$outbound`, and `$fault`. However, you cannot use it directly to access XML attachments in `$attachments`, headers in `$header`, or documents and RPC parameters in `$body`, with one exception—you can use it directly to access documents and parameters in `$body` for request messages received by a WSDL-based proxy service.

To learn more about creating variable structures, see [Creating Variable Structure Mappings](#).

To learn more about XQuery engine support and the relationship with Oracle functions and operators, see [Service Bus XQuery Functions](#).

Uses of the Inline XQuery Expression Editor

You typically use the Inline XQuery Expression Editor to enter simple XQueries that consist of the following:

- Fragments of XML with embedded XQueries.
- Simple variable paths along the child axis.

Note:

For more complex XQueries, we recommend that you use the Oracle XQuery Mapper, an editor with drag-and-drop functionality. See "Creating Transformations with the XQuery Mapper" in the *Developing SOA Applications with Oracle SOA Suite*.

Examples of good uses of inline XQueries are:

- Extract or access a business document or RPC parameter from the SOAP envelope elements in `$header` or `$body`.
- Extract or access an attachment document in `$attachments`.
- Set up the parameters of a service callout by extracting it from the SOAP envelope.
- Fold the result parameter of a service callout into the SOAP envelope.
- Extract a sequence from the SOAP envelope to drive a `for` loop.
- Update an item in the sequence in a `for` loop with an Update action.

You can also use the Inline XQuery Expression Editor to create variable structures. For more information, see [Using Variable Structures](#).

Best Practices for Type-Dependent Expressions

To help ensure expected results when using type-dependent expressions, manually cast values to the desired types. For example, the following statement casts `counter` as an integer for the XQuery compiler, which ensures a single return value:

```
<Body><result>{$foo/item[xs:int($counter)]}</result></Body>
```

Using Variable Structures

You can use the Inline XQuery Expression Editor to create variable structures, with which you define the structure of a given variable for design purposes. For example, it is easier to browse the XPath variable in the Administration Console rather than viewing the XML schema of the XPath variable. For examples of using variable structures in the Oracle Service Bus Console, see [Creating Variable Structure Mappings](#).

Note:

It is not necessary to create variable structures for your runtime to work. Variable structures define the structure of the variable or the variable path but do not create the variable. Variables are created at runtime as the target of the assign action in the stage.

In a typical programming language, the scope of variables is static. Their names and types are explicitly declared. The variable can be accessed anywhere within the static scope.

In Service Bus, there are some predefined variables, but you can also dynamically create variables and assign value to them using the assign action or using the loop variable in the for-loop. When a value is assigned to a variable, the variable can be accessed anywhere in the pipeline. The variable type is not declared but the type is essentially the underlying type of the value it contains at any point in time.

Note:

The scope of the for-loop variable is limited and cannot be accessed outside the stage.

When you use the Inline XQuery Expression Editor, the XQuery has zero or more inputs and one output. Because you can display the structure of the inputs and the structure of the output visually in the Expression Editor itself, you do not need to open the XML schema or WSDL resources to see their structure when you create the Inline XQuery. The graphical structure display also enables you to drag and drop simple variable paths along the child axis without predicates, into the composed XQuery.

Each variable structure mapping entry has a label and maps a variable or variable path to one or more structures. The scope of these mappings is the stage or route node. Because variables are not statically typed, a variable can have different structures at different points (or at the same point) in the stage or route node. Therefore, you can map a variable or a variable path to multiple structures, each with a different label. To view the structure, select the corresponding label with a list.

Note:

You can also create variable structure mappings in the Inline XPath Expression Editor. However, although the variable or a variable path is mapped to a structure, the XPaths generated when you select from the structure are XPaths relative to the variable. An example of a relative XPath is `./ctx:attachment/ctx:body`.

Quality of Service

The following sections discuss quality of service features in Service Bus messaging:

- [Delivery Guarantees](#)
- [Outbound Message Retries](#)

Delivery Guarantees

Service Bus supports reliable messaging. When messages are routed to another service from a route node, the default quality of service (QoS) is *exactly-once* if the proxy service is configured to be transactional; otherwise *best-effort* QoS is supported.

Quality of service is set in the `qualityOfService` element in the `$outbound` context variable.

The following delivery guarantee types are provided in Service Bus, shown in [Table 12-11](#).

Table 12-11 Delivery Guarantee Types

Delivery Reliability	Description
Exactly-once	<p><i>Exactly-once</i> reliability means that messages are delivered from inbound to outbound exactly-once, assuming a terminating error does not occur before the outbound message send is initiated. <i>Exactly-once</i> means reliability is optimized.</p> <p><i>Exactly-once</i> delivery reliability is a hint, not a directive. When <i>exactly-once</i> is specified, <i>exactly-once</i> reliability is provided if possible. If <i>exactly-once</i> is not possible, then <i>at-least-once</i> delivery semantics are attempted; if that is not possible, <i>best-effort</i> delivery is performed.</p> <p>Proxy services configured to be transactional have <i>exactly-once</i> quality of service.</p> <p>The default value of the <code>qualityOfService</code> element is also <i>exactly-once</i> for a route node action for the following inbound transports:</p> <ul style="list-style-type: none"> • email • FTP • SFTP • File • JMS (transactional) • Tuxedo (transactional) • MQ (with Backout Threshold set to zero) • WS <p>Note: Do not retry the outbound transport when the QoS is <i>exactly-once</i></p>
At-least-once	<p><i>At-least-once</i> semantics means the message is delivered to the outbound from the inbound at least once, assuming a terminating error does not occur before the outbound message send is initiated. Delivery is considered satisfied even if the target service responds with a transport-level error. However it is not satisfied in the case of a time-out, a failure to connect, or a broken communication link. If failover URLs are specified, <i>at-least-once</i> semantics is provided with respect to at least one of the URLs.</p> <p><i>At-least-once</i> delivery semantics is attempted if <i>exactly-once</i> is not possible but the <code>qualityOfService</code> element is <i>exactly-once</i>.</p>

Table 12-11 (Cont.) Delivery Guarantee Types

Delivery Reliability	Description
Best-effort	<p><i>Best-effort</i> means that there is no reliable messaging and there is no elimination of duplicate messages—however, performance is optimized. It is performed if the <code>qualityOfService</code> element is <code>best-effort</code>. <i>Best-effort</i> delivery is also performed if <i>exactly-once</i> and <i>at-least-once</i> delivery semantics are not possible but the <code>qualityOfService</code> element is <code>exactly-once</code>.</p> <p>The default value of the <code>qualityOfService</code> element for a route node is <code>best-effort</code> for the following inbound transports:</p> <ul style="list-style-type: none"> • HTTP • JMS (non-transactional) • Tuxedo (non-transactional) • MQ (with Backout Threshold set to greater than zero) <p>The default value of the <code>qualityOfService</code> element is always <code>best-effort</code> for the following:</p> <ul style="list-style-type: none"> • Service callout action – always <code>best-effort</code>, but can be changed if required. • Publish action – defaults to <code>best-effort</code>, modifiable <p>For more information on QoS behavior in publish actions, see Transformations and Publish Actions.</p> <p>Note: When the value of the <code>qualityOfService</code> element is <code>best-effort</code> for a publish action, all errors are ignored. However, when the value of the <code>qualityOfService</code> element is <code>best-effort</code> for a route node action or a Service Callout action, any error will raise an exception.</p>

For more detailed information about quality of service for other transports, see "Transports" in the *Developing Services with Oracle Service Bus*.

Overriding the Default Element Attribute

To override the default *exactly-once* quality of service attribute, you must set the `qualityOfService` in the outbound message context variable (`$outbound`). For more information, see [Message Context Schema](#).

You can override the default `qualityOfService` element attribute for the following pipeline actions:

- Publish
- Dynamic Publish
- Publish Table
- Service Callout
- Routing
- Dynamic Routing
- Routing Table

To override the `qualityOfService` element attribute, add a Routing Options action to any of the above actions, select the QoS option, and choose the override value.

Delivery Guarantee Rules

The delivery guarantee supported when a pipeline publishes a message or routes a request to a business service depends on the following conditions:

- The value of the `qualityOfService` element.
- The inbound transport (and connection factory, if applicable).
- The outbound transport (and connection factory, if applicable).

However, if the inbound proxy service is a Local Transport and is invoked by another proxy service, the inbound transport of the invoking proxy service is responsible for the delivery guarantee. That is because a proxy service that invokes another proxy service is optimized into a direct invocation if the transport of the invoked proxy service is a Local Transport. For more information on transport protocols, see [Creating and Configuring Proxy Services](#) and [Creating and Configuring Business Services](#).

Note:

No delivery guarantee is provided for responses from a proxy service.

The following rules govern delivery guarantees, shown in [Table 12-12](#).

Table 12-12 Delivery Guarantee Rules

Delivery Guarantee Provided	Rule
Exactly-once	The proxy service inbound transport is transactional and the value of the <code>qualityOfService</code> element is <code>exactly-once</code> to an outbound transport.
At-least-once	The proxy service inbound transport is file, FTP, or email and the value of the <code>qualityOfService</code> element is <code>exactly-once</code> .
At-least-once	The proxy service inbound transport is transactional and the value of the <code>qualityOfService</code> element, where applicable, is <code>exactly-once</code> to an outbound transport that is not transactional.
No delivery guarantee	All other cases, including all response processing cases.

Note:

To support *at-least-once* and *exactly-once* delivery guarantees with JMS, you must exploit JMS transactions and configure a retry count and retry interval on the JMS queue to ensure that the message is redelivered in the event of a server crash or a failure that is not handled in an error handler with a *Reply* or *Resume* action. File, FTP, and email transports also internally use a JMS/XA queue. The default retry count for a proxy service with a JMS/XA transport is 1.

The following are additional delivery guarantee rules:

- If the transport of the inbound proxy service propagates or starts a transaction, the request processing is performed in a transaction.
 - When the `qualityOfService` element is set to `exactly-once`, any route node actions executed in the request flow to a transactional destination are performed in the same transaction. Publish and Service Callout actions in a transaction context are *best-effort* by default and therefore execute outside of the transaction context. Setting those actions to *exactly-once* causes them to execute in the transaction context.
 - When the `qualityOfService` element is set to `best-effort` for any action in a route node, service callout or publish actions are executed outside of the request flow transaction. Specifically, for JMS, Tuxedo, Transactional Tuxedo, or EJB transport, the request flow transaction is suspended and the Transactional Tuxedo work is done without a transaction or in a separate transaction that is immediately committed.
 - If an error occurs during request processing, but is caught by a user error handler that manages the error (by using the `resume` or `reply` action), the message is considered successfully processed and the transaction commits. A transaction is aborted if the system error handler receives the error—that is, if the error is not handled before reaching the system level. The transaction is also aborted if a server failure occurs during request pipeline processing.
- If a response is received by a proxy service that uses a JMS/XA transport to business service (and the proxy inbound is not Transactional Tuxedo), the response processing is performed in a single transaction.
 - When the `qualityOfService` element is set to `exactly-once`, all route, service callout, and publish actions are performed in the same transaction.
 - When the `qualityOfService` element is set to `best-effort`, all publish actions and service callout actions are executed outside of the response flow transaction. Specifically, for JMS, EJB, or transactional Tuxedo types of transports, the response flow transaction is suspended and the service is invoked without a transaction or in a separate transaction that is immediately committed.
 - Proxy service responses executed in the response flow to a JMS/XA destination are always performed in the same transaction, regardless of the `qualityOfService` element setting.
- If the proxy service inbound transport is transactional Tuxedo, or if you set the "Same Transaction for Response" option on a proxy service, both the request processing and response processing are done in this transaction.

Note:

You will encounter a runtime error when the inbound transport is transactional Tuxedo and the outbound is an asynchronous transport, for example, JMS/XA.

Threading Model

The Service Bus threading model works as follows:

- Proxy service request and response pipelines always execute in separate threads.
- When invoking an external service, threads may be blocking or non-blocking depending on the pipeline action, the Quality of Service option and the transport used.

Service callouts are always blocking. An HTTP route or publish action is non-blocking (for request/response or one-way invocation), if the value of the `qualityOfService` element is `best-effort`.

JMS route actions or publish actions are always non-blocking, but the response is lost if the server restarts after the request is sent because Service Bus has no persistent message processing state.

Note:

In a request or response flow publish action, responses are always discarded because publish actions are inherently a one-way message send.

- When using blocking calls, a work manager having a Min Thread constraint must be associated with the response to prevent server deadlock. A Min Thread constraint guarantees a minimum number of threads for processing.

For general information about Work Managers, see "Using Work Managers to Optimize Scheduled Work" in *Administering Server Environments for Oracle WebLogic Server*. For information about Work Managers and threading in Service Bus, see [Work Managers and Threading](#).

Splitting Proxy Services

You may want to split a proxy service in the following situations:

- When HTTP is the inbound and outbound transport for a proxy service, you may want to incorporate enhanced reliability into the middle of the pipeline. To enable enhanced reliability in this way, split the proxy service into a front-end HTTP proxy service and a back-end JMS (one-way or request/response) proxy service with an HTTP outbound transport. In the event of a failure, the first proxy service must quickly place the message in the queue for the second proxy service, in order to avoid loss of messages.
- To disable the direct invocation optimization for a non-JMS transport when a proxy service, say `loanGateway1` invokes another proxy service, say `loanGateway2`. Route to the proxy service `loanGateway2` from the proxy service `loanGateway1` where the proxy service `loanGateway2` uses JMS transport.
- To have an HTTP proxy service publish to a JMS queue but have the publish action rollback if there is an exception later on in the request processing, split the proxy service into a front-end HTTP proxy service and a back-end JMS proxy service. The publish action specifies a `qualityOfService` element of `exactly-once` and uses an XA connection factory.

Outbound Message Retries

In addition to configuring inbound retries for messages using JMS, you can configure outbound retries and load balancing. Load balancing, failover, and retries work in conjunction to provide performance and high availability. For each message, the list of URLs you provide as failover URLs is automatically ordered based on the load

balancing algorithm into a failover sequence. If the retry count is N, the entire sequence is retried N times before stopping. The system waits for the specified retry interval before commencing subsequent loops through the sequence. After completing the retry attempts, if there is still an error, the error handler pipeline for the route node is invoked.

Note:

For HTTP transports, any HTTP status other than 200 or 202 is considered an error by Service Bus and must be retried. Because of this algorithm, it is possible that Service Bus retries errors like authentication failure that may never be rectified for that URL within the time period of interest. On the other hand, if Service Bus also fails over to a different URL for subsequent attempts to send a given message, the new URL may not give the error.

For quality of service=exactly-once, failover or retries will not be executed.

Using Work Managers with Service Bus

WebLogic Server helps you optimize the performance of your applications and web services environment as well as maintain service-level agreements with a feature called Work Managers. You create Work Manager resources and configure them by defining work execution rules. WebLogic Server uses the rules in a Work Manager to help prioritize work and allocate threads in whatever application or component the Work Manager is placed.

For general information about Work Managers, see "Using Work Managers to Optimize Scheduled Work" in *Administering Server Environments for Oracle WebLogic Server*. For information about Work Managers and threading in Service Bus, see [Work Managers and Threading](#).

In Service Bus, several transports for proxy and business services provide a configuration option called "Dispatch Policy" that lets you associate a Work Manager with a service to prioritize service work. This section describes how proxy and business services use Work Managers.

By configuring the Dispatch Policy of a Service Bus proxy service, the startup and execution of its request pipeline are governed by the rules of the Work Manager. For example, given a proxy service using a Dispatch Policy with a Max Constraint of 5, the proxy service will have no more than 5 request pipeline tasks executing simultaneously.

While the request pipeline is governed by the proxy service's Dispatch Policy, the response pipeline is governed by the business service or Split-Join Dispatch Policy. The RouteTo action specifies a business service or Split-Join to route the message to, and the response is subject to any Dispatch Policy on that business service or Split-Join.

When a RouteTo action specifies a local proxy service, the Dispatch Policy of the original proxy applies to work done in the local proxy's request pipeline. When the local proxy or chain of local proxies reaches a RouteTo action that invokes a business service or Split-Join, the business service, Split-Join, and all the following response pipelines are governed by the Dispatch Policy of that business service or Split-Join.

If an error occurs in the request, the error response is handled in the same thread as the request pipeline.

The quality of service (QoS) specified in the outbound metadata can also impact the way requests are threaded and thus impact what you see when monitoring Work Managers. The QoS on the RouteTo action defaults to the inbound QoS. For example, some inbound transactional transports set QoS to "exactly-once," such as JMS/XA, SB, Tuxedo, and WS (WS-RM). Other inbound transports, such as HTTP, set QoS to "best-effort" by default. The QoS on the RouteTo action is inherited from the inbound unless overridden by user settings in the RouteTo action.

When "best-effort" is used, the Route node invokes the business service asynchronously. In the case of "best-effort," the work thread returns to the pool and does not wait for the response; therefore, there is not a thread counting against a Work Manager constraint even though there is pending work due back asynchronously. But if "exactly-once" is selected, the request thread sends the request, blocks waiting for the response, and counts against the Work Manager constraints. In the case of "exactly-once," the waiting thread applies to the Work Manager assigned to the proxy service. Once a positive response arrives, a new thread processes the response pipeline using the Dispatch Policy assigned to the business service or Split-Join.

While using "exactly-once" is more expensive from performance, memory, and threading standpoints, "exactly-once" is necessary to maintain integrity on transactional inbound and outbound resources.

For more information on QoS, see [Quality of Service](#).

Content Types, JMS Type, and Encoding

To support interoperability with heterogeneous endpoints, Service Bus allows you to control the content type, the JMS type, and the encoding used.

Service Bus does not make assumptions about what the external client or service needs, but uses the information configured for this purpose in the service definition. Service Bus derives the content type for outbound messages from the service type and interface. Content type is a part of the email and HTTP protocols.

If the service type is:

- XML or SOAP with or without a WSDL file, the content type is text/XML.
- Messaging and the interface is MFL or binary, the content type is binary/octet-stream.
- Messaging and the interface is text, the content type is text/plain.
- Messaging and the interface is XML, the content type is text/XML.
- Messaging and the interface is Java, the content type is a Java Object.

Additionally, there is a JMS type, which can be byte or text for non-Java-type messages. You configure the JMS type to use when you define the service in Oracle Service Bus Console or in Oracle JDeveloper.

You can override the content type in the outbound context variable (`$outbound`) for proxy services invoking a service, and in the inbound context variable (`$inbound`) for a proxy service response. For more information on `$outbound` and `$inbound` context variables, see [Inbound and Outbound Variables](#).

Encoding is also explicitly configured in the service definition for all outbound messages. For more information on service definitions, see [Creating and Configuring Proxy Services](#) and [Creating and Configuring Business Services](#).

Throttling Pattern

In Service Bus, you can restrict the message flow to a business service. This technique of restricting a message flow to a business service is known as throttling. For information, see "Configuring Business Services for Message Throttling" in *Administering Oracle Service Bus*.

WS-I Compliance

Service Bus provides Web Service Interoperability (WS-I) compliance for SOAP 1.1 services in the runtime environment. The WS-I basic profile has the following goals:

- Disambiguate the WSDL and SOAP specifications wherever ambiguity exists.
- Define constraints that can be applied when receiving messages or importing WSDL files so that interoperability is enhanced. When messages are sent, construct the message so that the constraints are satisfied.

The WS-I basic profile is available at the following URL: <http://www.ws-i.org/Profiles/BasicProfile-1.1.html>

When you configure a proxy service or business service based on a WSDL file, you can use the Oracle Service Bus Console or Oracle JDeveloper to specify whether you want Service Bus to enforce WS-I compliance for the service. For more information, see [Security and Security Policies for Business Services](#).

When you configure WS-I compliance for a proxy service, checks are performed on inbound request messages received by that proxy service. When you configure WS-I compliance for an invoked service, checks are performed when any proxy receives a response message from that invoked service. Oracle recommends that you create an error handler for these errors, since by default, the proxy service SOAP client receives a system error handler-defined fault.

For messages sent from a proxy service, whether as outbound request or inbound response, WS-I compliance checks are not explicitly performed. That is because the pipeline designer is responsible for generating most of the message content. However, the parts of the message generated by Service Bus should satisfy all of the supported WS-I compliance checks. This includes the following content:

- Service invocation request message.
- System-generated error messages returned by a proxy service.
- HTTP status codes generated by a proxy service.

The Enforce WS-I Compliance check box is displayed as shown in [Figure 12-3](#).

Figure 12-3 Enforce WS-I Compliance Check Box

Edit a Proxy Service - Operation Selection Configuration (Path - default)

Enforce WS-I Compliance

Selection Algorithm

- Transport Header
- SOAPAction Header
- WS-Addressing
- SOAP Header
- SOAP Body Type

<< Back Next >> Finish Cancel

WS-I Compliance Checks

Note:

WS-I compliance checks require that the system knows what operation is being invoked on a service. For request messages received by a proxy service, that means that the context variable `$operation` should not be null. That depends upon the operation selection algorithm being configured properly. For response messages received from invoked services, the operation should be specified in the action configurations for route, publish, and service callout.

When you configure WS-I compliance checking for a proxy service or a business service, Service Bus carries out the following checks, shown in [Table 12-13](#).

Table 12-13 Service Bus WS-I Compliance Checks

Check	WS-I Basic Profile Details	Service Bus Description
3.1.1 SOAP Envelope Structure	R9980 An Envelope must conform to the structure specified in SOAP 1.1, Section 4, "SOAP Envelope" (subject to amendment).	This check applies to request and response messages. If a response message is checked and the message does not possess an outer <code>Envelope</code> tag, a <code>soap:client</code> error is generated. If the message is an <code>Envelope</code> tag but possesses a different namespace, it is handled by the 3.1.2 SOAP Envelope Namespace.
3.1.2 SOAP Envelope Namespace	R1015 A Receiver must generate an error if they encounter an envelope whose document element is not <code>soap:Envelope</code> .	This check applies to request and response messages and is related to the 3.1.1 SOAP Envelope Structure. If a request message has a local name of <code>Envelope</code> , but the namespace is not SOAP 1.1, a <code>soap:VersionMismatch</code> error is generated.
3.1.3 SOAP Body Namespace Qualification	R1014 The child elements of the <code>soap:body</code> element in an Envelope must be namespace qualified.	This check applies to request and response messages. All request error messages generate a <code>soap:Client</code> error.

Table 12-13 (Cont.) Service Bus WS-I Compliance Checks

Check	WS-I Basic Profile Details	Service Bus Description
3.1.4 Disallowed Constructs	R1008 An Envelope must not contain a Document Type Declaration.	This check applies to request and response messages. All request error messages generate a <code>soap:Client</code> error.
3.1.5 SOAP Trailers	R1011 An Envelope must not have any child elements of <code>soap:Envelope</code> following the <code>soap:body</code> element.	This check applies to request and response messages. All request error messages generate a <code>soap:Client</code> error.
3.1.9 SOAP attributes on SOAP 1.1 elements	R1032 The <code>soap:Envelope</code> , <code>soap:header</code> , and <code>soap:body</code> elements in an Envelope must not have attributes in the namespace <code>http://schemas.xmlsoap.org/soap/envelope/</code>	This check applies to request and response messages. Any request error messages generate a <code>soap:client</code> error.
3.3.2 SOAP Fault Structure	R1000 When an Envelope is a fault, the <code>soap:Fault</code> element must not have element children other than <code>faultcode</code> , <code>faultstring</code> , <code>faultactor</code> , and <code>detail</code> .	This check only applies to response messages.
3.3.3 SOAP Fault Namespace Qualification	R1001 When an Envelope is a Fault, the element children of the <code>soap:Fault</code> element must be unqualified.	This check only applies to response messages.
3.4.6 HTTP Client Error Status Codes	R1113 An instance should use a "400 Bad Request" HTTP status code if a HTTP request message is malformed. R1114 An instance should use a "405 Method not Allowed" HTTP status code if a HTTP request message is malformed. R1125 An instance must use a 4xx HTTP status code for a response that indicates a problem with the format of a request.	Only applies to responses for a proxy service where you cannot influence the status code returned due to errors in the request.
3.4.7 HTTP Server Error Status Codes	R1126 An instance must return a "500 Internal Server Error" HTTP status code if the response envelope is a fault.	This check applies differently to request and response messages. For request messages, any faults generated have a 500 Internal Server Error HTTP status code. For response messages, an error is generated if fault responses are received that do not have a 500 Internal Server Error HTTP status code.
4.7.19 Response Wrappers	R2729 An envelope described with an rpc-literal binding that is a response must have a wrapper element whose name is the corresponding <code>wsdl:operation</code> name suffixed with the string <code>Response</code> .	This check only applies to response messages. Service Bus never generates a non-fault response from a proxy service.

Table 12-13 (Cont.) Service Bus WS-I Compliance Checks

Check	WS-I Basic Profile Details	Service Bus Description
4.7.20 Part Accessors	<p>R2735 An envelope described with an rpc-literal binding must place the part accessor elements for parameters and return value in no namespace.</p> <p>R2755 The part accessor elements in a message described with an rpc-literal binding must have a local name of the same value as the name attribute of the corresponding <code>wsdl:part</code> element.</p>	This check applies to request and response messages. Any request error messages generate a <code>soap:client</code> error.
4.7.22 Required Headers	<p>R2738 An envelope must include all <code>soapbind:headers</code> specified on a <code>wsdl:input</code> or <code>wsdl:output</code> of a <code>wsdl:operation</code> of a <code>wsdl:binding</code> that describes it.</p>	This check applies to request and response messages. Any request error messages generate a <code>soap:client</code> error.
4.7.25 Describing SOAPAction	<p>R2744 A HTTP request message must contain a SOAPAction a HTTP header field with a quoted value equal to the value of the <code>soapAction</code> attribute of <code>soap:operation</code>, if present in the corresponding WSDL description.</p> <p>R2745 A HTTP request message must contain a SOAP action a HTTP header field with a quoted empty string value, if in the corresponding WSDL description, the SOAPAction of <code>soapbind:operation</code> is either not present, or present with an empty string as its value.</p>	This check applies to request messages and a <code>soap:client</code> error is returned.

Converting Between SOAP 1.1 and SOAP 1.2

Service Bus supports SOAP 1.1 and SOAP 1.2. A SOAP 1.1 proxy service can invoke a SOAP 1.2 business service or vice versa. However, due to differences between SOAP 1.1 and 1.2, Service Bus cannot automatically convert between the two in every situation. Use the following guidance to ensure proper conversion between SOAP 1.1 and 1.2:

- The SOAP namespace is automatically changed by Service Bus before invoking the business service. If a fault comes back from the business service it is automatically changed to the SOAP version of the proxy service. It is, however, up to the pipeline actions to map the SOAP header-related XML attributes (like `MustUnderstand`) between the two versions. It is also up to the pipeline actions to change the SOAP encoded namespace for encoded envelopes.
- Automatic conversion from SOAP 1.1 to SOAP 1.2 will work correctly only if the payload uses `doc/` or `rpc/literal` encoding.
- In SOAP 1.1, the `encodingStyle` attribute is allowed on any element in the envelope. In SOAP 1.2, the `encodingStyle` attribute is allowed only on child elements of the `Body`. Automatic conversion from SOAP 1.1 to SOAP 1.2 may result in an invalid envelope if the `encodingStyle` attribute in SOAP 1.1 is present outside of child

elements of Body, Header, and Fault detail. You must perform SOAP conversion in the proxy service pipeline to ensure a valid envelope.

- If the SOAP 1.1 and SOAP 1.2 services use different encoding styles, such as rpc/encoded to doc/literal, you must perform SOAP conversion in the proxy service pipeline.

Working with Pipelines in Oracle Service Bus Console

This chapter describes how to create and configure pipelines, or message flows, using the Oracle Service Bus Console. Sections include adding and configuring pipeline pairs, conditional branches, stages, operational branches, and route nodes.

- [Introduction to the Oracle Service Bus Console Pipeline Designer](#)
- [Viewing and Editing Pipelines in the Console](#)
- [Cutting, Copying, and Pasting Stages and Route Nodes](#)
- [Configuring the Resequencer in the Console](#)
- [Creating Variable Structure Mappings](#)

For more detailed information on pipelines and their components, see [Modeling Message Flow in Oracle Service Bus](#).

Introduction to the Oracle Service Bus Console Pipeline Designer

The pipeline designer provides a graphical representation of the message flow as you create and configure actions.

- [Edit Message Flow Page on the Console](#)
- [Edit Stage Configuration Page on the Console](#)

Edit Message Flow Page on the Console

Use the Edit Message Flow page to construct a message flow for a pipeline.

The left navigation pane of the Edit Message Flow page shows a tree view of the nodes and objects in the pipeline. When the details of an object are defined on a separate page, you can click the name of the object to display the associated page.

The right pane provides a field upon which to construct the pipeline. When a message flow has not yet been defined, the pane includes a single Pipeline icon that signifies the starting node for the pipeline. Click the icon to add pipeline pair nodes, route nodes, conditional branches, operational branches, and error handling for the service.

When you add objects to the page, icons are displayed on the page to represent the objects. The relationships among the objects are shown with lines and bounding boxes. Click an icon on the **Edit Message Flow** page to display a menu of the actions you can perform on that object. The options available on the menu may differ, depending on context. See [Table 13-1](#) for a complete list of icons and options.

Table 13-1 Edit Message Flow Page Icons and Options







Icon	Description	Menu Options
 Pipeline	The starting node for the pipeline.	<ul style="list-style-type: none"> • Add Pipeline Pair - See How to Add Pipeline Pairs to Pipelines. • Add Route - See How to Add Route Nodes to Pipelines in the Console. • Add Conditional Branch - See How to Add Conditional Branches to Pipelines in the Console. • Add Operational Branch - See How to Add Operational Branches to Pipelines in the Console. • Add Service Error Handler - See Adding Pipeline Error Handlers in the Console.
 Pipeline Pair Node	A pipeline pair node consists of a request pipeline and a response pipeline.	<ul style="list-style-type: none"> • Edit Name and Comments • Add Pipeline Pair - See How to Add Pipeline Pairs to Pipelines. • Add Route - See How to Add Route Nodes to Pipelines in the Console. • Add Conditional Branch - See How to Add Conditional Branches to Pipelines in the Console. • Add Operational Branch - See How to Add Operational Branches to Pipelines in the Console. • Paste Route - This option is available only if you have cut or copied a route node and it is on the Clipboard.
 Response Pipeline	See pipeline pair node above.	<ul style="list-style-type: none"> • Add Stage - See How to Add Stages to Pipelines in the Console. • Add Pipeline Error Handler - See Adding Pipeline Error Handlers in the Console.
 Request Pipeline	See pipeline pair node above.	<ul style="list-style-type: none"> • Add Stage - See How to Add Stages to Pipelines in the Console. • Add Pipeline Error Handler - See Adding Pipeline Error Handlers in the Console.
 Pipeline with Error Handler	A pipeline with an error handler defined for it.	<ul style="list-style-type: none"> • Edit Pipeline Error Handler - See Adding Pipeline Error Handlers in the Console. • Delete Pipeline Error Handler
 Route Node	Route node actions define the handling of messages as they flow through the route node.	<ul style="list-style-type: none"> • Edit Route - See How to Add Route Nodes to Pipelines in the Console. • Edit Name and Annotation • Add Route Error Handler - See Adding Route Node Error Handlers in the Console.

Table 13-1 (Cont.) Edit Message Flow Page Icons and Options

Icon	Description	Menu Options
 Route Node with Error Handler	A route node with an error handler defined for it.	<ul style="list-style-type: none"> • Edit Route Error Handler - See Adding Route Node Error Handlers in the Console. • Delete Route Error Handler
 Stage Node	A stage node is a container of actions.	<ul style="list-style-type: none"> • Edit Stage - See How to Add Stages to Pipelines in the Console. • Edit Name and Annotation • Add Stage - See How to Add Stages to Pipelines in the Console. • Add Stage Error Handler - See Adding Stage Error Handlers in the Console.
 Stage Node with Error Handler	A stage node with an error handler defined for it.	<ul style="list-style-type: none"> • Edit Stage Error Handler - See Adding Stage Error Handlers in the Console. • Delete Stage Error Handler
 Conditional Branch Node	A conditional branch node allows processing to proceed down exactly one of several possible paths.	<ul style="list-style-type: none"> • Edit Branch - See How to Add Conditional Branches to Pipelines in the Console. • Edit Name and Annotation
 Operational Branch Node	An operational branch node determines what branch to follow based on specified operations.	<ul style="list-style-type: none"> • Edit Branch - See How to Add Operational Branches to Pipelines in the Console. • Edit Name and Annotation
 Branch Node	A branch node is one of the alternative nodes defined by a conditional branch node or an operational branch node.	<ul style="list-style-type: none"> • Add Pipeline Pair - See How to Add Pipeline Pairs to Pipelines. • Add Route - See How to Add Route Nodes to Pipelines in the Console. • Add Conditional Branch - See How to Add Conditional Branches to Pipelines in the Console. • Add Operational Branch - See How to Add Operational Branches to Pipelines in the Console. • Paste Route - This option is available only if you have cut or copied a route node and it is on the Clipboard.
 Error Handler	An error handler provides the logic for resending errors in the pipeline.	<ul style="list-style-type: none"> • Add Service Error Handler - See Adding Pipeline Error Handlers in the Console.

Edit Stage Configuration Page on the Console

Use the Edit Stage Configuration page to add actions to pipeline stages, error handler stages, and route nodes in a pipeline.

- When nothing has yet been defined on the Edit Stage Configuration page, the only object displayed is the Add an Action icon. Click that icon to get started.
- When a stage or a route node has already been configured, the actions and objects defined for that stage or route node appear on the page. Edit the existing actions, as appropriate, or click any of the icons representing actions to add more actions to the stage.

See [Adding and Editing Pipeline Actions in the Console](#) for instructions on working with all the kinds of actions you can add to a stage.

Viewing and Editing Pipelines in the Console

The pipeline designer opens in a new browser window when you launch it from the Oracle Service Bus Console.

- [How to View and Edit Pipelines in the Console](#)
- [How to Add Shared Variables to Pipelines in the Console](#)
- [How to Add Pipeline Pairs to Pipelines](#)
- [How to Add Conditional Branches to Pipelines in the Console](#)
- [How to Add Operational Branches to Pipelines in the Console](#)
- [How to Add Stages to Pipelines in the Console](#)
- [How to Add Route Nodes to Pipelines in the Console](#)

How to View and Edit Pipelines in the Console

Perform the following steps to access the pipeline designer from the Oracle Service Bus Console.

To view and edit pipelines in the console:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.
2. In the left pane of the Oracle Service Bus Console window, click the **Resources** tab. The Project Navigator appears.
3. Click **All Projects**, then click the name of your service bus project.
4. Click the pipeline resource for which you wish to edit the pipeline.
5. Click the **Open Message Flow** icon near the top right corner of the window.
 - If no message flow has yet been created for the selected pipeline, the Edit Message Flow page is displayed with a single icon on the page, the Pipeline icon. This is the starting node for the pipeline message flow. Click this icon to begin constructing the message flow.

- If the pipeline already has a message flow, the page contains a graphic representation of the flow. Click the icons to view or edit the parts of the message flow.

For information on constructing the message flow, see [Working with Pipeline Actions in Oracle Service Bus Console](#).

6. Click the **Save** icon to commit the updates in the current session.
7. To end the session and deploy the configuration to the runtime, click **Activate** in the top right corner of the Oracle Service Bus Console window.

How to Add Shared Variables to Pipelines in the Console

If two pipelines in a single call chain declare the same shared variable, then they read and modify the same variable in the scope of the invocation call chain. In other words, if pipeline P1 declares a shared variable `var`, and pipeline P1 invokes pipeline P2, which also declares a shared variable `var`, then any changes to `var` in P1 are visible in P2, and vice versa. A shared variable must be of the String, Boolean, or XML data type.

When a pipeline receives and processes a message, all invoked pipelines that use a shared variable, read and write the same value for the variable. A subsequent message received by the proxy creates a new instance of the shared variable in the invoked pipelines.

Shared variables work across local proxy invocations and split-join component invocations. For example, say pipelines P1 and P2 declare a shared variable. Now, if P1 invokes a local proxy service or split-join component, which in turn invokes P2, then P1 and P2 continue to share the shared variable.

The following restrictions apply to using shared variables:

- System variables (such as `$body`, `$attachments`, `$operation`, `$inbound`, `$outbound`) cannot be shared.
- Variables cannot be shared across non-local proxy invocations. For example, say a pipeline invokes an HTTP proxy service, the shared variable is not propagated across this call.
- Variables cannot be shared between pipeline and split-join resources.
- Variables with Java and binary content types are not supported. For example, an XML-typed variable that has `<ctx: java-content />` in its XML structure, is not supported as a shared variable.

Before you begin

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a shared variable to a pipeline:

1. Click **Expand Shared Variables section** to expand the Shared Variables section at the top left hand corner of the Edit Message Flow page.
2. Enter a name for the shared variable in the **Variable** field.
3. Click **Add** to add the shared variable to the pipeline.

How to Add Pipeline Pairs to Pipelines

A pipeline can include zero or more pipeline pair nodes: request and response pipelines for the pipeline (or for the operations on the service), and error handler pipelines that can be defined for stages, pipelines, and the service. Pipelines can include one or more stages, which in turn include actions.

Before you begin

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a pipeline pair to a pipeline:

1. Click the **Pipeline** icon, then click **Add Pipeline Pair**.
2. To change the default name and add a description for the pipeline pair node, do the following:
 - a. Click the **Pipeline Pair Node** icon, then click **Edit Name and Comments**.
 - b. Change the name and description, as desired.
 - c. Click **Save**.

Note:

When you rename a pipeline or a route node, the number of messages displayed on the Dashboard page in the Monitoring module may not correlate with those of other components due to the pipeline counters being reset to zero. This is because Service Bus treats the rename as a delete and recreate action. The numbers should correlate again after a time period equal to the service's monitoring interval has elapsed.

3. To add stages to the pipeline, see [How to Add Stages to Pipelines in the Console](#).
4. To add actions to stages in the pipeline. See [Adding and Editing Pipeline Actions in the Console](#).
5. Click **Save** to commit the updates in the current session.
6. To end the session and deploy the configuration to the runtime, click **Activate** in the top right corner of the Oracle Service Bus Console window.

How to Add Conditional Branches to Pipelines in the Console

A branch node allows processing to proceed along exactly one of several possible paths. Branching is driven by an XPath-based switch table. Each branch in the table specifies a condition (for example, <500) that is evaluated in order down the pipeline against a single XPath expression (for example, `./ns:PurchaseOrder/ns:totalCost on $body`). Whichever condition is satisfied first determines which branch is followed. If no branch condition is satisfied, then the default branch is followed. A branch node may have several descendants in the pipeline: one for each branch, including the default branch.

If the proxy service is not based on a WSDL file and receives multiple document types as input, consider using a conditional branch node.

Conditional branching is driven by a lookup table with each branch tagged with a simple, but unique, string value. A variable in the message context is designated as the lookup variable for that node, and at runtime, its value is used to determine which branch to follow. If no branch matches the value of the lookup variable, the default branch is followed. You should design the pipeline in such a way that the value of the lookup variable is set before reaching the branch node.

Before you begin

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a conditional branch to a pipeline:

1. Click a **Pipeline Pair Node** icon or a **Branch Node** icon, then click **Create Conditional Branch**. The conditional branch node is added, and any existing nodes after the inserted branch node are moved to the default branch of the new conditional branch node.
2. To change the default name and add a description for the branch node, do the following:
 - a. Click the **Conditional Branch** icon, then click **Edit Name and Comments**.
 - b. Change the name and description, as desired.
 - c. Click **Save**.
3. To add branch definitions, click the **Conditional Branch** icon, then click **Edit Branch**. The **Edit Branch Node** page is displayed.
4. Do the following:
 - a. In the **Selected Path** field, click **Edit** to add an XPath expression for specifying the path. See [Creating and Editing Inline XQuery and XPath Expressions](#).
 - b. In the **Variable** field, enter a context variable.
 - c. From the **Operator** field, select a comparison operator.
 - d. In the **Value** field, enter a value for the branch.
 - e. In the **Label** field, enter a label for the branch.
5. Optionally, under Options:
 - Click **Add a New Branch** to add a new branch definition to this branch node.
 - Click **Delete this Branch** to delete a branch definition.
 - Click **Move Branch Up** or click **Move Branch Down** to change the positions of branch definitions. This option displays only when more than one branch definition exists.
6. Click **Save** to commit the updates in the current session.
7. On the Edit Message Flow page, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in the Console](#).
8. Click **Save** to commit the updates in the current session.

9. To end the session and deploy the configuration to the runtime, click **Activate** in the top right corner of the Oracle Service Bus Console window.

How to Add Operational Branches to Pipelines in the Console

When pipelines define Web Services Description Language (WSDL)-based proxy services, operation-specific processing is required. Instead of configuring a branching node based on operations manually, Service Bus provides a minimal configuration branching node that automatically branches based on operations. In other words, when you create an operational branch node in a pipeline, you can quickly build your branching logic based on the operations defined in the WSDL file because the Oracle Service Bus Console presents those operations in the branch node configuration page.

A branch node allows processing to proceed along exactly one of several possible paths. Branching is driven by an XPath-based switch table. Each branch in the table specifies a condition (for example, `<500`) that is evaluated in order down the pipeline against a single XPath expression (for example, `./ns:PurchaseOrder/ns:totalCost on $body`). Whichever condition is satisfied first determines which branch is followed. If no branch condition is satisfied, then the default branch is followed. A branch node may have several descendants in the pipeline: one for each branch, including the default branch.

Before you begin

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add an operation branch to a pipeline:

1. Click a **Pipeline Pair Node** icon or a **Branch Node** icon, then click **Create Operational Branch**. The operational branch node is added, and any existing nodes after the inserted branch node are moved to the default branch of the new operational branch node.
2. To change the default name and add a description for the branch node, do the following:
 - a. Click the **Operational Branch** icon, then click **Edit Name and Comments**.
 - b. Change the name and description, as desired.
 - c. Click **Save**.
3. To add branch definitions, click the **Operational Branch** icon, then click **Edit Branch**. The Edit Branch Node page is displayed.
4. In the Operation Branch Definitions panel, select a service operation.
5. Optionally, under Options:
 - Click **Add a New Branch** to add a new branch definition to this branch node.
 - Click **Delete this Branch** to delete a branch definition.
 - Click **Move Branch Up** or click **Move Branch Down** to change the positions of branch definitions. This option displays only when more than one branch definition exists.
6. Click **Save**.

7. On the Edit Message Flow page, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in the Console](#).
8. Click **Save** to commit the updates in the current session.
9. To end the session and deploy the configuration to the runtime, click **Activate** in the top right corner of the Oracle Service Bus Console window.

How to Add Stages to Pipelines in the Console

Before you begin

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a stage to a pipeline:

1. If necessary, click the plus sign to the left of the **Pipeline Pair Node** icon to expand it. A pipeline pair contains a Request Pipeline and a Response Pipeline.
2. Click the pipeline to which you want to add the stage, then click **Add Stage**.
3. To change the default name and add a description for the stage, do the following:
 - a. Click the **Stage** icon, then click **Edit Name and Comments**.
 - b. Change the name and description, as desired.
 - c. Click **Save**.
4. To add actions to the stage, click the **Stage** icon, then click **Edit Stage**. See [Adding and Editing Pipeline Actions in the Console](#).
5. To add error handling to the stage, click the **Stage** icon, then click **Add Stage Error Handler**. See [Adding Error Handlers in the Console](#). The Edit Message Flow page is displayed.
6. Continue to construct the pipeline, as described in [Viewing and Editing Pipelines in the Console](#).
7. Click **Save** to commit the updates in the current session.
8. To end the session and deploy the configuration to the runtime, click **Activate** in the top right corner of the Oracle Service Bus Console window.

How to Add Route Nodes to Pipelines in the Console

Before you begin

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a route node to a pipeline:

1. Click the **Pipeline Pair Node** icon of a pipeline pair, then click **Add Route**.
2. To change the default name and add a description for the route node, click the **Route Node** icon, then click **Edit Name and Comments**. Change the name and description, as desired, then click **Save**.

Note:

When you rename a pipeline or a route node, the number of messages displayed on the Dashboard page in the Monitoring module may not correlate with those of other components due to the pipeline counters being reset to zero. This is because Service Bus treats the rename as a delete and re-create action. The numbers should correlate again after a time period equal to the service's monitoring interval has elapsed.

3. To add actions to the route node, click the **Route Node** icon, then click **Edit Route**. The Edit Message Flow page is displayed. See the following sections for information about the actions you can add to route nodes:
 - [Adding If-Then Actions in the Console](#)
 - [Adding Dynamic Routing to Route Nodes in the Console](#)
 - [Adding Routing Actions to Route Nodes in the Console](#)
 - [Adding Routing Tables to Route Nodes in the Console](#)
 - [Adding Error Handlers in the Console](#)
4. On the Edit Message Flow page, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in the Console](#).
5. Click **Save** to commit the updates in the current session.
6. To end the session and deploy the configuration to the runtime, click **Activate** in the top right corner of the Oracle Service Bus Console window.

Cutting, Copying, and Pasting Stages and Route Nodes

You can cut, copy, and paste stages and route nodes.

- To cut a stage or a route node, click its icon and select **Cut** or **Copy**.
- To paste a stage that you cut or copied from a different pipeline pair within the message flow of this pipeline or from the message flow of a different pipeline, do one of the following:
 - Click the **Request Pipeline or Response Pipeline** icon, then click **Paste Stage**.
 - Click a **Stage** icon in a pipeline, then click **Paste**.
- To paste a route node that you cut or copied from another pipeline, click the **Pipeline Pair Node** icon for the pipeline pair, then click **Paste Route**.

Configuring the Resequencer in the Console

The resequencer in Service Bus rearranges a stream of related but out-of-sequence messages into a sequential order. When incoming messages arrive, they may be in a random order. The resequencer orders the messages based on sequential or chronological information, and then sends the messages to the target services in an orderly manner. The sequencing is performed based on the sequencing strategy selected.

You can configure the resequencer inside a pipeline component. Pipelines with the following service types are supported:

- **WSDL:** Resequencing is available for operations with only request type.
- **Message Type:** The request message type should be XML, and the response message type should be None.

Note:

The resequencer does not support **Any XML** and **Any SOAP** service types. For WSDL-based services, the WSDL file must be one-way; that is, it cannot contain output elements. For information about using generated WSDL files with resequencing pipelines, see [How to Export a WSDL File in the Console](#).

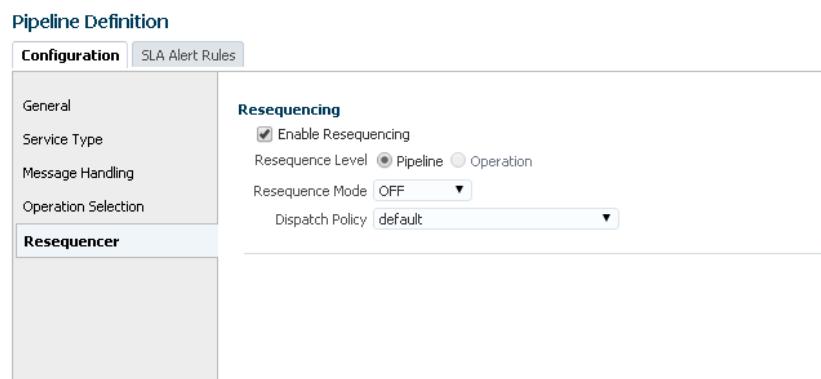
How to Configure Resequencing in a Pipeline in the Console

This section describes how to configure the resequencer in a pipeline using Oracle Service Bus Console.

To enable resequencing in a pipeline component:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.
2. In the left pane of the Oracle Service Bus Console window, click the **Resources** tab. The Project Navigator appears.
3. Click **All Projects**, then click the name of your service bus project.
4. Click the pipeline resource for which you wish to configure the resequencer. The Pipeline Definition page appears.
5. Make sure that the **Configuration** tab is selected. Click **Resequencer** in the left pane of the Pipeline Definition page.
6. Select **Enable Resequencing** to enable resequencing for the pipeline. [Figure 13-1](#) shows the Enable Resequencing option on the Pipeline Definition Resequencing page.

Figure 13-1 Enabling the Resequencer in Oracle Service Bus Console



7. Select the **Resequencing Level**. Choose **Pipeline** to configure resequencing at the component level. Choose **Operations** to configure resequencing at the operation

level. See [How to Select the Resequence Level in the Console](#) for more information on resequence levels.

If you select **Operations**, you get the option to configure resequencing for each operation separately.

8. Select the **Resequence Mode**. If you are configuring resequencing at the **Operations** level, then you can select a **Resequence Mode** corresponding to each operation. See [How to Configure the Resequencing Mode in the Console](#) for more information on the various resequencing options.

Depending on the **Resequence Mode** you select, you get options corresponding to that mode. For example, selecting the **Standard** mode requires you to select values for **Group Expression**, **Id Expression**, and so on. [Figure 13-2](#) shows the options displayed for the **Standard** mode.

Figure 13-2 Configuration Options Displayed for Standard Mode Resequencing

The screenshot shows the 'Pipeline Definition' console with the 'Configuration' tab selected. The 'Resequencer' section is active, displaying the following configuration options:

- Resequencing**
 - Enable Resequencing
 - Resequence Level: Pipeline Operation
 - Resequence Mode: Standard (dropdown menu)
 - Dispatch Policy: default (dropdown menu)
- Group Expression: <Edit an Expression> (with edit icon)
- * ID Expression: <Edit an Expression> (with edit icon)
- Start: 1 (spin button)
- Increment: 1 (spin button)
- Timeout: 0 sec. (spin button)

9. Select a **Dispatch Policy**, which specifies the Work Manager to use. The default Work Manager is used if no other Work Manager exists.
10. Click the **Save** icon to commit your changes.

How to Select the Resequence Level in the Console

You can define resequencing either at the pipeline level or the operation level. The Resequence Level can have the following values:

- **Pipeline:** A common configuration specified at the component level is used to resequence all messages. If a component has multiple operations, then messages for each operation are sequenced separately using the common component configuration.

Component-level resequencing is allowed only when all the operations of the pipeline component support request one-way messages. If only a subset of operations support request one-way messages, then you can individually specify operation-level resequencing for these operations.

- **Operation:** For a WSDL-based pipeline, resequencing can be configured at the operation level. Each operation can have a different resequencer configuration.

Only operations supporting request one-way messages can be resequenced. Non-WSDL pipelines cannot have resequencer configured at the operation level.

How to Configure the Resequencing Mode in the Console

This section provides instructions on how to configure various resequencing modes. See "Resequencing Order" in *Developing SOA Applications with Oracle SOA Suite* to learn about the various resequencing modes. By default, the group ID has a character limit of 1000; the ID has a character limit of 100.

Configuring a Standard Resequencer

To configure a standard resequencer:

1. In the Pipeline Definition Resequencing page, select **Standard** from the **Resequence Mode** drop-down list. If you are configuring resequencing at the operation level, select **Standard** from the **Resequence Mode** drop-down list for the operation.

The fields related to standard resequencing configuration appear on the page. See [Figure 13-2](#) for more details.

2. Fill in the fields listed in [Table 13-2](#).

Table 13-2 Standard Resequencing Options

Field Name	Description	Default Value	Mandatory
Group Expression	An XQuery expression that points to the field in the incoming message on which grouping is done. If you do not enter a value, then all messages are put in one default group. Click the Expression Builder icon on the right to invoke the XQuery/XSLT Expression Editor.	N/A	N
ID Expression	An XQuery expression that points to the field in the incoming message on which resequencing is done. Click the Expression Builder icon on the right to invoke the XQuery/XSLT Expression Editor.	N/A	Y
Start	The starting number of the ID sequence.	1	N
Increment	The increment of the ID sequence.	1	N
Timeout	The time period in seconds to wait for an expected message. The resequencer locks the group as timed-out if a time out occurs. The default value of 0 means that the timeout never happens for a group by default.	0	N

Configuring a FIFO Resequencer

To configure a FIFO resequencer:

1. In the Pipeline Definition Resequencing page, select **FIFO** from the **Resequencing Mode** drop-down list. If you are configuring resequencing at the operation level, select **FIFO** from the **Resequencing Mode** drop-down list for the operation.

The fields related to FIFO resequencing configuration appear on the page.

2. In the **Group Expression** field, enter an XQuery expression pointing to the field in the incoming message on which grouping is performed.

Click the Expression Builder icon on the right to invoke the XQuery/XSLT Expression Editor.

Configuring a Best Effort Resequencer

To configure a best effort resequencer:

1. In the Pipeline Definition Resequencing page, select **Best Effort** from the **Resequencing Mode** drop-down list. If you are configuring resequencing at the operation level, select **Best Effort** from the **Resequencing Mode** drop-down list for the operation.

The fields related to Best Effort resequencing configuration appear on the page.

2. Fill in the fields listed in [Table 13-3](#) to configure the best effort resequencer.

Table 13-3 Best Effort Resequencing Options

Field Name	Description	Default Value	Mandatory
Group Expression	An XQuery expression that points to the field in the incoming message on which grouping is performed. If no value is entered here, then all messages are considered to be in one default group. Click the Expression Builder icon on the right to invoke the XQuery/XSLT Expression Editor.	N/A	N
ID Expression	An XQuery expression that points to the field in the incoming message that contains the ID on which resequencing is performed. Click the Expression Builder icon on the right to invoke the XQuery/XSLT Expression Editor.	N/A	Y
Data Type	The data type of the sequence ID. The ordering process is based on the data type. Supported values are Date/Time and Numeric.	Numeric	Y

Table 13-3 (Cont.) Best Effort Resequencing Options

Field Name	Description	Default Value	Mandatory
Max Rows	Number of in-sequence messages that the resequencer should pick from the data store at a time. This must be a positive integer value. You must specify Max Rows or Time Window (explained below), but not both.	5	N
Time Window (sec)	The length of time in seconds to wait after a message arrives before selecting messages from the data store for resequencing. The default value of 0 means no wait. You must specify a Time Window or Max Rows (described above), but not both.	0	N

Creating Variable Structure Mappings

The following sections describe how to create several types of variable structure mappings:

- [Sample WSDL Document](#)
- [Creating the Resources You Need for the Examples](#)
- [Example 1: Selecting a Predefined Variable Structure](#)
- [Example 2: Mapping a Variable to a Type](#)
- [Example 3: Mapping a Variable to an Element](#)
- [Example 4: Mapping a Variable to a Child Element](#)
- [Example 5: Mapping a Variable to a Business Service](#)
- [Example 6: Mapping a Child Element to Another Child Element](#)

Sample WSDL Document

This sample WSDL document is used in most of the examples in this section. You need to save this WSDL document as a resource in your configuration. For more information, see [Creating the Resources You Need for the Examples](#).

Example - Sample WSDL Document

```
<definitions
  name="samplewsdl"
  targetNamespace="http://example.org"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:s0="http://www.oracle.com"
  xmlns:s1="http://example.org"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
<types>
  <xs:schema
```

```

attributeFormDefault="unqualified"
elementFormDefault="qualified"
targetNamespace="http://www.oracle.com"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
<xs:element name="PO" type="s0:POType"/>
<xs:complexType name="POType">
  <xs:all>
    <xs:element name="id" type="xs:string"/>
    <xs:element name="name" type="xs:string"/>
  </xs:all>
</xs:complexType>
<xs:element name="Invoice" type="s0:InvoiceType"/>
<xs:complexType name="InvoiceType">
  <xs:all>
    <xs:element name="id" type="xs:string"/>
    <xs:element name="name" type="xs:string"/>
  </xs:all>
</xs:complexType>
</xs:schema>
</types>
<message name="POTypeMsg">
  <part name="PO" type="s0:POType"/>
</message>
<message name="InvoiceTypeMsg">
  <part name="InvReturn" type="s0:InvoiceType"/>
</message>

<portType name="POPortType">
  <operation name="GetInvoiceType">
    <input message="s1:POTypeMsg"/>
    <output message="s1:InvoiceTypeMsg"/>
  </operation>
</portType>
<binding name="POBinding" type="s1:POPortType">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetInvoiceType">
    <soap:operation soapAction="http://example.com/GetInvoiceType"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
</definitions>

```

Creating the Resources You Need for the Examples

To make use of the examples that follow, save the sample WSDL document as a resource in your configuration and create the sample business service and proxy service that use the sample WSDL document.

The instructions that follow tell how to accomplish the tasks in the Oracle Service Bus Console:

- [Save the WSDL File as a Resource](#)
- [Create a Proxy Service and Pipeline](#)
- [Build a Message Flow for the Sample Pipeline](#)

- [Create a Business Service](#)

Save the WSDL File as a Resource

Perform the following steps:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.
2. In the left pane of the Oracle Service Bus Console window, click the **Resources** tab. The Project Navigator appears.
3. Expand the **All Projects** node by clicking the **Expand** (arrow) icon before it.
4. Click the project name to which you wish to add the WSDL file.
5. From the **Create** drop-down list, select **WSDL**. The Create WSDL dialog appears.
6. In the **Resource Name** field, enter `SampleWSDL`. This is a required field.
7. In the **Description** field, enter a description for the WSDL resource. This is optional.
8. Click **Choose File** and select the WSDL sample file.
9. Click **Create** to create the WSDL resource.

Create a Proxy Service and Pipeline

Perform the following steps to create a proxy service and pipeline that use the sample WSDL document:

1. In the Project Navigator, select the project to which you want to add a pipeline, and then click the down arrow next to the **Create** icon.
2. Select **Pipeline** from the list of options.
The Create Pipeline dialog appears.
3. In the **Pipeline Name** field of the **General** section, enter `PipelinewithSampleWSDL`. This field is mandatory.
Optionally, specify a **Description** for the pipeline.
4. Under Service Type, select **WSDL Based Service**.
5. Click the **Browse** icon and select `SampleWSDL` from the list of WSDL files. You may need to click Search to search for the WSDL file.
6. Select **Expose as Proxy Service** to create a proxy service corresponding to the pipeline.
7. In the Name field, enter `ProxywithSampleWSDL`.
8. Click **Create** to create the pipeline and proxy service. The pipeline is created and opened up for editing.

Build a Message Flow for the Sample Pipeline

Perform the following steps:

1. In the Project Navigator, click the pipeline `PipelineWithSampleWSDL` to open it.
2. Click the **Open Message Flow** icon, in the top right corner, to start editing the message flow for the pipeline. The Edit Message Flow page appears.
3. In the **Edit Message Flow** page, click the `PipelineWithSampleWSDL` icon, then click **Add Pipeline Pair**. `PipelinePairNode1` is displayed, which includes request and response pipelines.
4. Click the **Request Pipeline** icon, then click **Add Stage**. The Stage **Stage1** is displayed.
5. Click **Save**. The basic message flow is created for the `PipelineWithSampleWSDL` pipeline.

Create a Business Service

Perform the following steps to create a business services that uses the sample WSDL document:

1. In the Project Navigator, select the project to which you want to add the business service.
2. From the **Create** drop-down list, select **Business Service**. The Create Business Service wizard appears.
3. In the **Resource Name** field, enter `BusinessWithSampleWSDL`. This is a required field.
4. Under Service Definition, select **WSDL Based Service**.
5. Click the **Browse** icon, next to the **Name** field, and select `SampleWSDL` from the list of WSDL files. You may need to click Search to search for the WSDL file.
6. Click **Next**. Click **Create**. The business service is created.

You are now ready to use the examples—continue in [Example 1: Selecting a Predefined Variable Structure](#).

Example 1: Selecting a Predefined Variable Structure

In this example, you select a predefined variable structure using the proxy service `ProxyWithSampleWSDL`, which has a service type **WSDL Web Service** that uses the binding **POBinding** from `SampleWSDL`.

The pipeline message flow needs to know the structure of the message in order to manipulate it. To achieve this, Service Bus automatically provides a predefined structure that maps the **body** variable to the SOAP body structure as defined by the WSDL file of the proxy service for all the messages in the interface. This predefined structure mapping is labeled **body**.

Note:

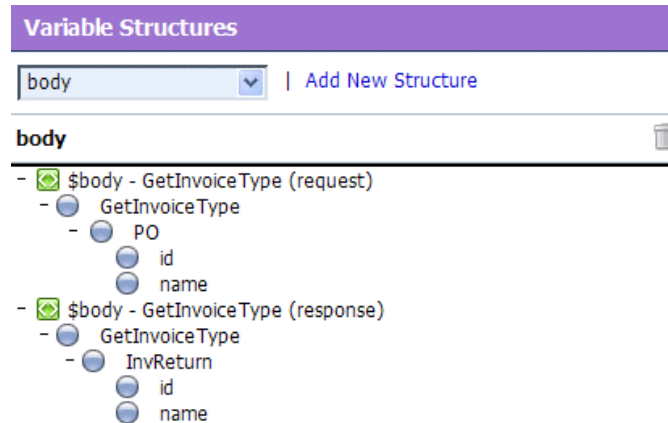
This predefined structure is also supported for messaging services with a typed interface.

To select a predefined variable structure:

In the Variable Structures panel on the XQuery Expression Editor page, select **body** from the list of built-in structures.

The variable structure **body** is displayed in [Figure 13-3](#).

Figure 13-3 Variable Structures—body



Example 2: Mapping a Variable to a Type

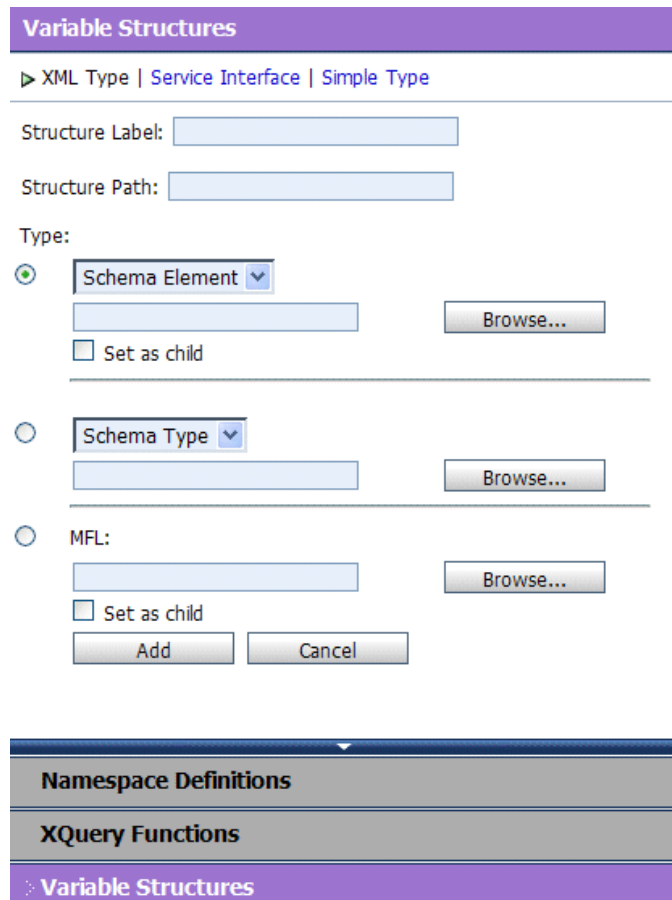
Suppose the proxy service `ProxyWithSampleWSDL` invokes a service callout to the business service `BusinessWithSampleWSDL`, which also has a service type WSDL Web Service that uses the binding **POBinding** from `SampleWSDL`. The operation `GetInvoiceType` is invoked.

In this example, the pipeline needs to know the structure of the response parameter in order to manipulate it. To achieve this, you can create a new variable structure that maps the response parameter variable to the type `InvoiceType`.

To map a variable to a type:

1. In the Variable Structures panel, click **Add New Structure**. Additional fields are displayed in [Figure 13-4](#).

Figure 13-4 Variable Structures—Add a New Structure



2. Select the **XML Type**.
3. In the **Structure Label** field, enter `InvoiceType` as the display name for the variable structure you want to create. This display name enables you to give a meaningful name to the structure so you can recognize it at design time but it has no impact at runtime.
4. In the **Structure Path** field, enter `$InvoiceType` as the path of the variable at runtime.
5. To select the type **InvoiceType**, do the following:
 - a. Under the **Type** field, select the appropriate radio button, then select **WSDL Type** from the list.
 - b. Click **Browse**. The **WSDL Browser** is displayed.
 - c. In the **WSDL Browser**, select **SampleWSDL**, then select **InvoiceType** under **Types** in the **Select WSDL Definitions** pane.
 - d. Click **Submit**. **InvoiceType** is displayed under your selection **WSDL Type**.
6. Click **Add**. The new variable structure **InvoiceType** is included under **XML Type** in the list of variable structures.

The variable structure **InvoiceType** is displayed in [Figure 13-5](#).

Figure 13-5 Variable Structures—InvoiceType

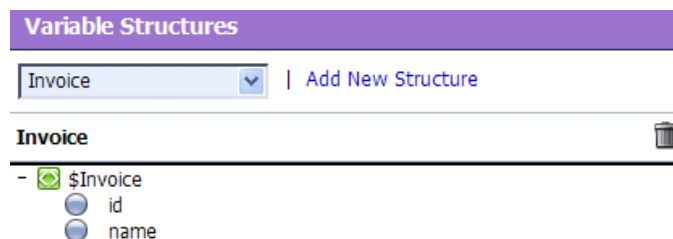
Example 3: Mapping a Variable to an Element

Suppose a temporary variable has the element **Invoice** described in the `SampleWSDL` WSDL file. In this example, the `ProxyWithSampleWSDL` pipeline needs to access this variable. To achieve this, you can create a new variable structure that maps the variable to the element **Invoice**.

To map a variable to an element:

1. In the Variable Structures panel, click **Add New Structure**.
2. Make sure you select the **XML Type**.
3. In the **Structure Label** field, enter `Invoice` as the meaningful display name for the variable structure you want to create.
4. In the **Structure Path** field, enter `$Invoice` as the path of the variable structure at runtime.
5. To select the element **Invoice**, do the following:
 - a. For the **Type** field, make sure you select the appropriate radio button. Then select **WSDL Element**.
 - b. Click **Browse**.
 - c. In the **WSDL Browser**, select `SampleWSDL`, then select **Invoice** under **Elements** in the **Select WSDL Definitions** pane.
 - d. Click **Submit**. **Invoice** is displayed under your selection **WSDL Element**.
6. Click **Add**. The new variable structure **Invoice** is included under **XML Type** in the list of variable structures.

The variable structure **Invoice** is displayed in [Figure 13-6](#).

Figure 13-6 Variable Structures—Invoice

Example 4: Mapping a Variable to a Child Element

The `ProxyWithSampleWSDL` proxy service routes to the document style Any SOAP business service that returns the Purchase Order in the SOAP body. In this example, the `ProxyWithSampleWSDL` pipeline must then manipulate the response. To achieve this, you can create a new structure that maps the **body** variable to the **PO** element, and specify the **PO** element as a child element of the variable. You need to specify it as a child element because the **body** variable contains the SOAP Body element and the **PO** element is a child of the Body element.

To map a variable to a child element:

1. In the Variable Structures panel, click **Add New Structure**.
2. Make sure you select the **XML Type**.
3. In the **Structure Label** field, enter `body to PO` as the meaningful display name for the variable structure you want to create.
4. In the **Structure Path** field, enter `$body` as the path of the variable structure at runtime.
5. To select the **PO** element:
 - a. Under the **Type** field, make sure you select the appropriate radio button, and then select **WSDL Element**.
 - b. Click **Browse**.
 - c. In the **WSDL Browser**, select `SampleWSDL`, then select **PO** under **Elements** in the **Select WSDL Definitions** pane.
 - d. Click **Submit**.
6. Select the **Set as child** check box to set the **PO** element as a child of the **body to PO** variable structure.
7. Click **Add**. The new variable structure **body to PO** is included under **XML Type** in the list of variable structures.

The variable structure **body to PO** is displayed in [Figure 13-7](#).

Figure 13-7 Variable Structures—*body to PO*



Example 5: Mapping a Variable to a Business Service

The `ProxyWithSampleWSDL` proxy service routes the message to the `BusinessWithSampleWSDL` business service, which also has a service type WSDL

Web Service that uses the binding **POBinding** from `SampleWSDL`. In this example, the pipeline must then manipulate the response. To achieve this, you can define a new structure that maps the **body** variable to the `BusinessWithSampleWSDL` business service. This results in a map of the **body** variable to the SOAP body for all the messages in the WSDL interface of the service.

Note:

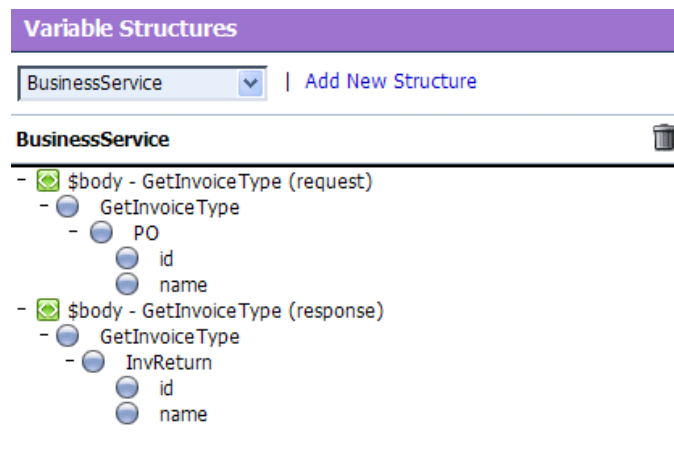
This mapping is also supported for messaging services with a typed interface.

To map a variable to a business service:

1. In the Variable Structures panel, click **Add New Structure**.
2. Select **Service Interface**.
3. In the **Structure Label** field, enter `BusinessService` as the meaningful display name for the variable structure.
4. In the **Structure Path** field, `$body` is already set as the default. This is the path of the variable structure at runtime.
5. To select the business service, do the following:
 - a. Under the **Service** field, click **Browse**. The **Service Browser** is displayed.
 - b. In the **Service Browser**, select the `BusinessWithSampleWSDL` business service, then click **Submit**. The business service is displayed under the **Service** field.
 - c. In the **Operation** field, select **All**.
6. Click **Add**. The new variable structure `BusinessService` is included under **Service Interface** in the list of variable structures.

The variable structure `BusinessService` is displayed in [Figure 13-8](#).

Figure 13-8 Variable Structures—Business Service



Example 6: Mapping a Child Element to Another Child Element

Modify the `SampleWSDL` so that the `ProxyWithSampleWSDL` proxy service receives a single attachment. The attachment is a Purchase Order. In this example, the pipeline must then manipulate the Purchase Order. To achieve this, you can define a new structure that maps the `body` element in `$attachments` to the `PO` element, which is specified as a child element. The `body` element is specified as a variable path of the form:

```
$attachments/ctx:attachment/ctx:body
```

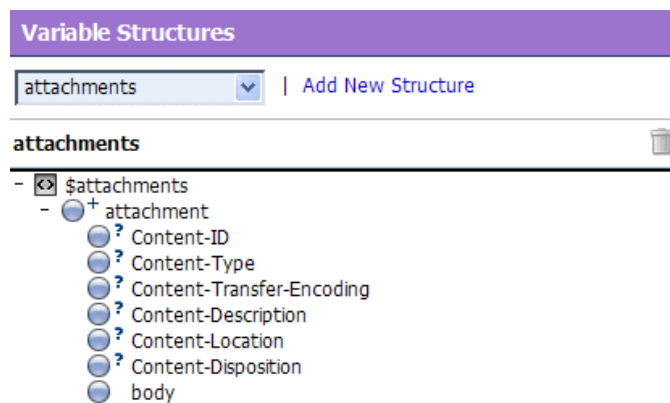
You can select and copy the `body` element from the predefined `attachments` structure, paste this element as the variable path to be mapped in the new mapping definition.

To map a child element to another child element:

1. In the Variable Structures panel, select `attachments` from the list of built-in structures.

The variable structure `attachments` is displayed in [Figure 13-9](#).

Figure 13-9 Variable Structures—*attachments*



2. Select the `body` child element in the `attachments` structure. The variable path of the `body` element is displayed in the Property Inspector on the right side of the page:

```
$attachments/ctx:attachment/ctx:body
```

3. Copy the variable path of the `body` element.
4. In the Variable Structures panel, click **Add New Structure**.
5. Select the **XML Type**.
6. In the **Structure Label** field, enter `PO` attachment as the meaningful display name for this variable structure.
7. In the **Structure Path** field, paste the variable path of the `body` element:

```
$attachments/ctx:attachment/ctx:body
```

This is the path of the variable structure at runtime.

8. To select the `PO` element:

- a. Under the **Type** field, make sure the appropriate radio button is selected, then select **WSDL Element**.
 - b. Click **Browse**.
 - c. In the **WSDL Browser**, select `SampleWSDL`, then select **PO** under **Elements** in the **Select WSDL Definitions** pane.
 - d. Click **Submit**.
9. Select the **Set as child** check box to set the PO element as a child of the **body** element.
 10. Click **Add**. The new variable structure **PO attachment** is included under XML Type in the list of variable structures.
 11. If there are multiple attachments, add an index to the reference when you use fields from this structured variable in your XQueries. For example, if you drag the PO field to the XQuery field, but as PO will be the second attachment, change the inserted value from

```
$attachments/ctx:attachment/ctx:body/oracle:PO/oracle:id
```

to

```
$attachments/ctx:attachment[2]/ctx:body/oracle:PO/oracle:id
```

Working with Pipeline Actions in Oracle Service Bus Console

This chapter describes how to add different types of actions to pipelines, using the Oracle Service Bus Console, such as route, publish, service callout, transport headers, conditional actions, error actions, and message transformation actions.

Actions are the elements of pipeline stages, error handler stages, route nodes, and branch nodes that determine how messages are to be defined as they flow through a pipeline.

This chapter includes the following sections:

- [Adding and Editing Pipeline Actions in the Console](#)
- [Adding Publish Actions in the Console](#)
- [Adding Publish Table Actions in the Console](#)
- [Adding Dynamic Publish Actions in the Console](#)
- [Adding Routing Options Actions in the Console](#)
- [Adding Service Callout Actions in the Console](#)
- [Adding Transport Header Actions in the Console](#)
- [Adding Dynamic Routing to Route Nodes in the Console](#)
- [Adding Routing Actions to Route Nodes in the Console](#)
- [Adding Routing Tables to Route Nodes in the Console](#)
- [Adding For-Each Actions in the Console](#)
- [Adding If-Then Actions in the Console](#)
- [Adding Raise Error Actions in the Console](#)
- [Adding Reply Actions in the Console](#)
- [Adding Resume Actions in the Console](#)
- [Adding Skip Actions in the Console](#)
- [Adding Assign Actions in the Console](#)
- [Adding Delete Actions in the Console](#)
- [Adding Insert Actions](#)
- [Adding Java Callout Actions in the Console](#)

- [Adding MFL Translate Actions in the Console](#)
- [Adding nXSD Translate Actions](#)
- [Adding Rename Actions in the Console](#)
- [Adding Replace Actions in the Console](#)
- [Adding Validate Actions in the Console](#)
- [Adding Alert Actions in the Console](#)
- [Adding Log Actions in the Console](#)
- [Adding Report Actions in the Console](#)
- [Adding Error Handlers in the Console](#)
- [Disabling an Action or a Stage in the Console](#)

Adding and Editing Pipeline Actions in the Console

Actions are the elements of pipeline stages, error handler stages, route nodes, and branch nodes that determine how messages are to be defined as they flow through a proxy service.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

They also assume you have already added a pipeline stage, a route node, and/or an error handler stage. See:

- [How to Add Pipeline Pairs to Pipelines](#)
- [How to Add Stages to Pipelines in the Console](#)
- [Adding Pipeline Error Handlers in the Console](#)

To add an action to a pipeline:

1. Select the component to which you want to add an action. For example, click the **Stage** icon, then click **Edit Stage**, or click the **Route Node** icon, then click **Edit Route**.
2. Depending on whether actions have already been added to the stage or to the route node, do one of the following:
 - If no actions have yet been added, the Edit Stage Configuration page displays only the Add an Action icon. Click that icon, then select an action type.
 - If one or more actions have already been added, the Edit Stage Configuration page displays one or more icons representing those actions, for example, a Publish icon or a Routing icon. Click the appropriate icon, click **Add an Action**, then select an action type.
 - Some actions, such as request and response actions in publish actions, include an **Add an Action** link where an action is appropriate. Click that icon, then select an action type.

There are no restrictions on what actions may be chained together in a pipeline. [Table 14-1](#) through [Table 14-4](#) list the actions you can configure for pipelines.

Table 14-1 Pipeline - Communication Actions

Action	Description	More Information
 Dynamic Publish	Publish a message to a service identified by an XQuery expression	Adding Dynamic Publish Actions in the Console
 Publish	Publish a message to a statically specified service.	Adding Publish Actions in the Console
 Publish Table	Publish a message to zero or more statically specified services. Switch-style condition logic is used to determine at runtime which services will be used for the publish.	Adding Publish Table Actions in the Console
 Routing Options	Modify any or all of the following properties in the outbound request: URI, Quality of Service, Mode, Retry parameters, Message Priority.	Adding Routing Options Actions in the Console
 Service Callout	Configure a synchronous (blocking) callout to a Service Bus-registered proxy or business service.	Adding Service Callout Actions in the Console
 Transport Headers	Set the transport header values in messages	Adding Transport Header Actions in the Console
 Dynamic Routing	Assign a route for a message based on routing information available in an XQuery resource.	Adding Dynamic Routing to Route Nodes in the Console
 Routing	Identify a target service for the message and configure how the message is routed to that service:	Adding Routing Actions to Route Nodes in the Console

Table 14-1 (Cont.) Pipeline - Communication Actions


Action	Description	More Information
 Routing Table	Assign a set of routes wrapped in a switch-style condition table. Different routes are selected based upon the results of a single XQuery expression.	Adding Routing Tables to Route Nodes in the Console

Table 14-2 Pipeline - Flow Control Actions







Action	Description	More Information
 For each	Iterate over a sequence of values and execute a block of actions	Adding For-Each Actions in the Console
 If...then...	Perform an action or set of actions conditionally, based on the Boolean result of an XQuery expression.	Adding If-Then Actions in the Console
 Raise error	Raise an exception with a specified error code (a string) and description.	Adding Raise Error Actions in the Console
 Reply	Specify that an immediate reply be sent to the invoker.	Adding Reply Actions in the Console
 Resume	Resume pipeline after an error is handled by an error handler.	Adding Resume Actions in the Console
 Skip	Specify that at runtime, the execution of the current stage is skipped and the processing proceeds to the next stage in the pipeline.	Adding Skip Actions in the Console

Table 14-3 Pipeline - Message Processing Actions


Action	Description	More Information
 Assign	Assign the result of an XQuery expression to a context variable.	Adding Assign Actions in the Console

Table 14-3 (Cont.) Pipeline - Message Processing Actions









Action	Description	More Information
 Delete	Delete a context variable or a set of nodes specified by an XPath expression.	Adding Delete Actions in the Console
 Insert	Insert the result of an XQuery expression at an identified place relative to nodes selected by an XPath expression.	Adding Insert Actions
 Java callout	Invoke a Java method from the pipeline.	Adding Java Callout Actions in the Console
 MFL transform	Convert non-XML to XML or XML to non-XML in the pipeline.	Adding MFL Translate Actions in the Console
 nXSD translate	Convert native data format (nXSD) to XML or XML to native data format (nXSD) in the pipeline.	Adding nXSD Translate Actions
 Rename	Rename elements selected by an XPath expression without modifying the contents of the element.	Adding Rename Actions in the Console
 Replace	Replace a node or the contents of a node specified by an XPath expression.	Adding Replace Actions in the Console
 Validate	Validate elements selected by an XPath expression against an XML schema element or a WSDL resource.	Adding Validate Actions in the Console

Table 14-4 Pipeline - Reporting Actions




Action	Description	More Information
 Alert	Send an alert notification based on pipeline message context.	Adding Alert Actions in the Console

Table 14-4 (Cont.) Pipeline - Reporting Actions

Action	Description	More Information
 Log	Construct a message to be logged.	Adding Log Actions in the Console
 Report	Enable message reporting for a proxy service.	Adding Report Actions in the Console

3. When you have finished adding actions, you can further configure the actions in stage or route node, as described in [Table 14-5](#).

Table 14-5 Edit Stage Configuration Tasks

To...	Complete This Step...
Delete an action	Click the appropriate icon, then click Delete this Action .
Move an action down (demote)	Click the appropriate icon, then click Move Action Down . The action is moved below the next action contained in this stage. This option is displayed only when a stage contains two or more actions.
Move an action up (promote)	Click the appropriate icon, then click Move Action Up . The action is moved above the previous action contained in this stage. This option is displayed only when the stage contains two or more actions.
Cut an action	Click the appropriate icon, then click Cut .
Copy an action	Click the appropriate icon, then click Copy .
Paste an action that you have cut or copied	Click the appropriate icon, then click Paste Action . You can copy and paste actions across stages. However, in the case of Assign, Replace or Insert actions, note the following: <ul style="list-style-type: none"> • All variable-related and user-defined namespaces from the source (copied) stage are added as user-defined namespaces in the target (pasted) stage. • Duplicate namespaces (identical namespaces in both source and target stage) are not copied. • Conflicting namespaces (namespace declarations that use the same prefix but different URIs) are copied. Users will be able to save the configuration, but will not be able activate it until the conflicting namespace declarations in stage B are removed.

Table 14-5 (Cont.) Edit Stage Configuration Tasks

To...	Complete This Step...
Validate a stage	In the Edit Stage Configuration page, click Validate to validate all the actions configured in that stage. Only actions that are enabled are validated. The Validate button is not available if the stage is disabled.

4. Click **Save** to commit the updates in the current session.
5. On the Edit Message Flow page, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in the Console](#).
6. Click **Save** to commit the updates in the current session.
7. To end the session and deploy the configuration to the runtime, click **Activate** in the top right corner of the Oracle Service Bus Console.

Adding Publish Actions in the Console

Use a publish action to identify a statically specified target service for a message and to configure how the message is packaged and sent to that service. For more information on publish behavior, see [Performing Transformations in Pipelines](#).

To add a publish action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Select **Add an Action > Communication > Publish**.
3. Click **Service**. The Select Service page is displayed.
4. Select a service from the list, then click **Submit**. This is the target service for the message.
5. If the service has operations defined, you can specify an operation to be invoked by selecting it from the **Operation** list.
6. To make the outbound operation the same as the inbound operation, select the **Use inbound operation for outbound** check box.
7. To configure how the message is packaged and sent to the service, in the **Request Actions** field, click **Add an Action**. Then select an action to associate with the service. You can add more than one action. See [Adding and Editing Pipeline Actions in the Console](#).
8. Click **Save** to commit the updates in the current session.

Adding Publish Table Actions in the Console

Use a publish table action to publish a message to zero or more statically specified services. Switch-style condition logic is used to determine at runtime which services will be used for the publish.

For more information on publish behavior, see [Performing Transformations in Pipelines](#).

To add a publish table action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Select **Add an Action > Communication > Publish Table**.
3. Click **Expression**. The XQuery Expression Editor page is displayed. Create an XQuery expression, which at runtime returns the value upon which the routing decision will be made. See [Creating and Editing Inline XQuery and XPath Expressions](#).
4. From the **Operator** list, select a comparison operator. Then, in the adjacent field, enter a value against which the value returned from the XQuery expression will be evaluated.
5. Click **Service** to select a service to which messages are to be published if the expression evaluates true for the value you specified. The Select Service page appears.
6. Select a service from the list, then click **Submit**. This is the target service for the message.
7. If the service has operations defined, you can specify the operation to be invoked by selecting it from the **invoking** list.
8. If you want the outbound operation to be the same as the inbound operation, select the **Use inbound operation for outbound** check box.
9. In the **Request Actions** field, to configure how the message is packaged and sent to the service, click **Add an Action**, then select one or more actions that you want to associate with the service. To learn more about the type of action you want to add, see [Adding and Editing Pipeline Actions in the Console](#).
10. To insert a new case, click the **Case** icon, then select **Insert New Case**.
11. Repeat steps 4-8 for the new case.
12. Add additional cases as dictated by your business logic.
13. Click the **Case** icon of the last case you define in the sequence, then select **Insert Default Case** to add a default case at the end.
14. Configure the default case—the configuration of this case specifies the routing behavior in the event that none of the preceding cases is satisfied.
15. Click **Save** to commit the updates in the current session.

Adding Dynamic Publish Actions in the Console

Use a dynamic publish action to publish a message to a service specified by an XQuery expression. For more information on publish behavior, see [Performing Transformations in Pipelines](#).

To add a dynamic publish action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).

2. Click the appropriate icon, then select **Add an Action > Communication > Dynamic Publish**.
3. Click **Expression**.
4. In the XQuery Expression Editor, enter an XQuery expression or select an XQuery resource that provides a result similar to:

```
<ctx:route>
  <ctx:service isProxy="false">project/folder/businessservicename</ctx:service>
  <ctx:operation>foo</ctx:operation>
</ctx:route>
```

Note:

If a proxy service is being invoked, set `isProxy` to true. If a business service is being invoked, set `isProxy` to false.

The element `operation` is optional.

Alternatively, the following code routes to a pipeline:

```
<ctx:route>
  <ctx:pipeline>project/folder/pipeline</ctx:pipeline>
</ctx:route>
```

Alternatively, the following code routes to a split join:

```
<ctx:route>
  <ctx:splitjoin>project/folder/splitjoin</ctx:splitjoin>
</ctx:route>
```

5. Click **Save**.
6. In the **Request Actions** field, click **Add an Action** to add an action, then select an action that you want to associate with the service. You can add more than one action. To learn more about the type of actions you can add, see the table of actions in [Adding and Editing Pipeline Actions in the Console](#).
7. Click **Save** to commit the updates in the current session.

Adding Routing Options Actions in the Console

Use the routing options action to modify any or all of the following properties for the outbound request in `$outbound`: URI, Quality of Service, Mode, Retry parameters. Although these properties can be modified using assign, insert, replace, or delete actions on `$outbound`, using routing options provides a simpler way to perform this task, without requiring knowledge of XPath, XQuery, or the structure of the `$outbound` context variable.

The routing options action can only be used where the context variable `$outbound` is valid. It can be added to the following actions:

- Publish
- Dynamic Publish
- Publish Table

- Service Callout
- Routing
- Dynamic Routing
- Routing Table

For more information on routing, see [Modeling Message Flow in Oracle Service Bus](#).

To configure a routing options action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Communication > Routing Options**.
3. Complete any or all of the following steps:
 - To set the URI for the outbound message: Select **URI**, and click the **XQuery Expression Editor**. Enter an expression that returns a URI. This overrides the URI for the invoked service.

Note:

When routing to another proxy service, the URI override has no effect.

- To set the Quality of Service element: Select **Quality of Service**, and select the Quality of Service option from the list. This overrides the default that is auto computed.
- To set the Mode: Select **Mode**, and select either request, or request-response from the list.

Note:

This is normally already automatically set, based on the interface of the service invoked. However, in some cases like Any Soap or Any XML services, this is not so.

- To set the Retry Interval: Select **Retry Interval**, and specify the number of seconds between retries. This overrides the default configured with the invoked service.
 - To set the Retry Count: Select **Retry Count**, and specify the number of retries the system must attempt before discontinuing the action. This overrides the default configured with the invoked service.
 - To set the Message Priority: Select **Priority**, and click the **XQuery Expression Editor**. Enter an expression that returns a positive integer.
4. Click **Save** to commit the updates in the current session.

Adding Service Callout Actions in the Console

Use a service callout action to configure a synchronous (blocking) callout to a Service Bus-registered proxy or business service. For more information on service callout actions, see [Constructing Service Callout Messages](#).

To add a service callout action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action** > **Communication** > **Service Callout**.
3. Click **Service**. The Service Browser is displayed.
4. Select a service from the list of registered proxy or business services, then click **Submit**.
5. If the service you chose in step 3, above, is WSDL-based and has operations that can be invoked on the service, those operations are listed in the **invoking Operation** list. Select an operation to be invoked on the service.

Note:

Selecting an operation, which Service Bus requires for many reasons, does not guarantee that only the selected operation is invoked. For example, if you select OperationA, but a message also contains an invocation for Operation B, then OperationB will be invoked as well.

6. Specify how you want to configure the request and response messages by selecting one of the following options:
 - Select **Configure SOAP Body** to configure the SOAP Body. Selecting this option allows you to use \$body directly.

Note:

This option supports SOAP-RPC encoded, which is not supported when configuring payload parameters or document.

- Select **Configure Payload Parameters** or **Configure Payload Document** to configure the payload.
7. Subsequent configuration options depend on the kind of service you selected and on the kind of configuration options you chose for that service.

[Table 14-6](#) provides instructions for each option.

Table 14-6 Service Callout Configuration Options

For These Options...	Follow These Steps...
SOAP Request Body and SOAP Response Body	<p>To configure these options,</p> <ul style="list-style-type: none"> In the SOAP Request Body field, enter the name of a variable to hold the XML of the SOAP Body element for the callout request. In the SOAP Response Body field, enter the name of a variable to which the XML of the SOAP Body element on the response will be bound.
Request Attachments Variable and Response Attachments Variable	<p>To configure these options (optional),</p> <ul style="list-style-type: none"> In the Request Attachments Variable field, enter the name of a variable to hold the XML corresponding to the outbound request attachments. In the Response Attachments Variable field, enter the name of a variable to hold the XML corresponding to the outbound response attachments.
SOAP Request Header and SOAP Response Header	<p>To configure these options,</p> <ul style="list-style-type: none"> In the SOAP Request Header field, enter the name of a variable to hold the XML of the SOAP Header element for the callout request You must wrap the input document for the SOAP Request Header with <code><soap-env:Header> . . . </soap-env:Header></code>. In the SOAP Response Header field, enter the name of a variable to which the XML of the SOAP Headers on the response, if any, will be bound.
Request Parameters and Response Parameters	<p>To configure options,</p> <ul style="list-style-type: none"> In the Request Parameters fields, enter names for the variables that will be evaluated at runtime to provide values for the request parameters. You must provide only the core payload documents in the input variable—the SOAP package is created for you by Service Bus. In other words, do not wrap the input document with <code><soap-env:Body> . . . </soap-env:Body></code>. For example, when creating a body input variable that is used for this request parameter, you would define that variable's contents using the XPath statement <code>body/*</code> (to remove the wrapper <code>soap-env:Body</code>), not <code>\$body</code> (which results in keeping the <code>soap-env:Body</code> wrapper). In the Response Parameters fields, enter the names of the variables to which the responses will be assigned at runtime.

Table 14-6 (Cont.) Service Callout Configuration Options

For These Options...	Follow These Steps...
Request Document and Response Document	<p>To configure these options,</p> <ul style="list-style-type: none"> In the Request Document Variable field, enter the name of a variable to assign a request document to. For SOAP Document-type services, the variable is evaluated at runtime to form the body of the SOAP message sent to the service. For <i>Any XML</i> services, the variable is evaluated at runtime to form the body of the XML message sent to the service. For SOAP Document-type services and for Any XML services, you provide only the core payload documents in the input variable—the SOAP package is created for you by Service Bus. In other words, do not wrap the input document with <code><soap-env:Body> . . . </soap-env:Body></code>. For example, when creating a body input variable that is used for this request parameter, you would define that variable's contents using the XPath statement <code>body/*</code> (to remove the wrapper <code>soap-env:Body</code>), not <code>\$body</code> (which results in keeping the <code>soap-env:Body</code> wrapper). For <i>Messaging</i> services, the variable is evaluated to form the body of the message, based on the type of data expected by the service. The following restrictions apply to variables used with Messaging services: <ul style="list-style-type: none"> - For services that expect binary data, the variables must have a <code>ctx:binary-content</code> element. - For services that expect MFL data, the variable must have the XML equivalent. - For services that expect text data, the variable is a string. In the Response Document Variable field, enter the name of the variable to which a response document will be assigned at runtime.

- Optionally, add one or more transport headers. For more information, see [Adding Transport Header Actions in the Console](#).

Note:

In addition to the transport headers you specify, headers are added by the Service Bus binding layer. For more information, see [Configuring Transport Headers in Pipelines](#).

- Click **Save** to commit the updates in the current session.

Adding Transport Header Actions in the Console

Use a transport header action to set the header values in messages.

To add a transport header action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Communication > Transport Headers**.
3. From the **Set Transport Headers for** list, select one of the following, to specify to the runtime which of the message context locations are to be modified:
 - **Outbound Request** - Select this option to set header values for outbound requests (the messages sent out by a proxy service in route, publish, or service callout actions). This header element is located in the message context as follows:

```
$outbound/ctx:transport/ctx:request/tp:headers
```
 - **Inbound Response** - Select this option to set header values for inbound responses (the response messages a proxy service sends back to clients). This header element is located in the message context as follows:

```
$inbound/ctx:transport/ctx:response/tp:headers
```
4. Optionally, select **Pass all Headers through Pipeline** to pass all headers through from the inbound message to the outbound message or vice versa. Every header in the source set of headers will be copied to the target header set, overwriting any existing values in the target header set.

For information about using this option in conjunction with the header-specific pass through option, see [Configuring Transport Headers in Pipelines](#).
5. Complete the following steps for each Header you want to add:
 - a. In the Transport Headers table, click **Add Header** to display fields for configuring the header.
 - b. Specify a header by doing either of the following:
 - From the list in the Name column, select a header name. The list contains all of the predefined header names for the target transport (for example, **Content-Type** for HTTP transports, **JMSCorrelationID** for JMS transports, and so on).
 - Enter a header name in the **Other** field. If that header name is not one of the predefined headers for this service's transport, it becomes a user-header, as defined by the transport specification.
 - c. Select one of the options in the Action column to specify how to set the headers value:

Set Header to Expression

Selecting this option allows you to use an XQuery or XSLT expression to set the value of the header. The expression can be simple (for example, "text/xml") or a complex XQuery or XSLT expression.

Because the Service Bus transport layer defines the XML representation of all headers as string values, the result of any expression is converted to a string before the header value is set. Expressions that return nothing result in the header value being set to the empty string. You cannot delete a header using an expression.

Caution:

Not all of the header settings you can specify in this action are honored at runtime. For information about which of the headers for a given transport you can set and which of those set are honored at runtime, see [Configuring Transport Headers in Pipelines](#).

Delete Header

Specifies that the header is removed from the request or response metadata.

Copy Header from Inbound Request (if you are setting transport headers for the Outbound Request)

or

Copy Header from Outbound Response (if you are setting transport headers for the Inbound Response)

Specifies that this header is copied directly from the corresponding header of the same name from the inbound message to the outbound message and vice versa. For example, if you want to set the SOAPAction header for an outbound request, selecting **Copy Header from Inbound Request** causes the runtime to copy the value from the SOAPAction request header of `$.inbound`. In the case of inbound response headers, the source of the header to copy is the response headers of `$.outbound`.

If the **Copy Header** option is selected for a header that does not exist in the source, this option is ignored and no action is performed on the target for this header.

For information about using this option in conjunction with the global **Pass all Headers through Pipeline** option, see [Configuring Transport Headers in Pipelines](#).

6. To add additional Headers to the table, click the **Header** icon, then click **Add Header**.

The table is expanded to include an additional row, which includes a new set of options that you can use to configure another transport header. You can add as many headers as necessary to this table. You do not have to order the headers in the table, because the runtime declares namespaces and places header elements in their proper order when generating the corresponding XML.

7. Click **Save** to commit the updates in the current session.

Setting Cookies in Outbound HTTP Transport Headers

In an HTTP pipeline you can set cookies on the transport header in the following ways:

- [Setting a Cookie as a Complex XML Expression](#)
- [Setting a Cookie with a String Expression](#)

Setting a Cookie as a Complex XML Expression

To set a cookie using a complex XML expression, which is the Service Bus default format, configure the value of the HTTP Cookie header in the outbound request using the following expression syntax:

```
<cookie-values xmlns="http://www.bea.com/wli/sb/transports/http">
  <value>{fn:concat("cookie_name", "=", "cookie_value")}</value>
</cookie-values>
```

Setting a Cookie with a String Expression

To set the Cookie header with a string in the outbound request, you must add the following option to your domain startWebLogic command:

```
-Dcom.bea.osb.http.cookieAsNoComplexElement=true
```

After you restart the server with this option, you can set an HTTP Cookie header with a string expression. For example:

```
$cookie_name = "cookie_value"
```

Adding Dynamic Routing to Route Nodes in the Console

Assign a route for a message based on routing information available in an XQuery resource. This is a terminal action, which means you cannot add another action after this one. However, this action can contain request and response actions. For more information on dynamic routing, see [Using Dynamic Routing](#).

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add dynamic routing to a route node:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the **Route Node** icon, then click **Edit Route**. The Edit Stage Configuration page is displayed.
3. Click the **Add an Action** icon, then select **Communication > Dynamic Routing**.
4. Click **Expression**. The XQuery Expression Editor is displayed.
5. In the XQuery Expression Editor, enter an XQuery expression, the result of which is similar to:

```
<ctx:route>
  <ctx:service isProxy='true'>{$service}</ctx:service>
  <ctx:operation>{$operation}</ctx:operation>
</ctx:route>
```


Note:

If a proxy service is being invoked, set `isProxy` to true. If a business service is being invoked, set `isProxy` to false.

- The service name is the fully qualified service name.
- The operation element is optional.

Alternatively, the following code routes to a pipeline:

```
<ctx:route>
  <ctx:pipeline>project/folder/pipeline</ctx:pipeline>
</ctx:route>
```

Alternatively, the following code routes to a split join:

```
<ctx:route>
  <ctx:splitjoin>project/folder/splitjoin</ctx:splitjoin>
</ctx:route>
```

6. Click **Save**.
7. In the **Request Actions** field, click **Add an Action** to add an action, then select an action that you want to associate with the service. You can add more than one action. To learn more about the type of actions you want to add, see the table of actions in [Adding and Editing Pipeline Actions in the Console](#).
8. In the **Response Actions** field, click **Add an Action** to add an action, then select an action that you want to associate with the service. You can add more than one action. To learn more about the type of actions you want to add, see the table of actions in [Adding and Editing Pipeline Actions in the Console](#).
9. Click **Save**.
10. On the Edit Message Flow page, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in the Console](#).
11. Click **Save** to commit the updates in the current session.
12. To end the session and deploy the configuration to the runtime, click **Activate** in the top right corner of the Oracle Service Bus Console.

Adding Routing Actions to Route Nodes in the Console

Identify a target service for the message and configure how the message is routed to that service. This is a terminal action, which means you cannot add another action after this one. However, this action can contain request and response actions. For more information on routing, see [Modeling Message Flow in Oracle Service Bus](#).

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a routing action to a route node:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).

2. Click the **Route Node** icon, then click **Edit Route**. The Edit Stage Configuration page is displayed.
3. Click the **Add an Action** icon, then select **Communication > Routing**.
4. Click **Service**. The Service Browser is displayed.
5. Select a service from the list, then click **Submit**. The service is displayed instead of the default link.
6. If you want the outbound operation to be the same as the inbound operation, select the **Use inbound operation for outbound** check box.
7. In the **Request Actions** field, click **Add an Action** to add an action, then select an action that you want to associate with the service. You can add more than one action. To learn more about the type of actions you can add, see the table of actions in [Adding and Editing Pipeline Actions in the Console](#).
8. In the **Response Actions** field, click **Add an Action** to add an action, then select an action that you want to associate with the service. You can add more than one action. To learn more about the type of actions you can add, see the table of actions in [Adding and Editing Pipeline Actions in the Console](#).
9. Click **Save**.
10. On the Edit Message Flow page, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in the Console](#).
11. Click **Save** to commit the updates in the current session.
12. To end the session and deploy the configuration to the runtime, click **Activate** under Change Center.

Adding Routing Tables to Route Nodes in the Console

A routing table is a set of routes wrapped in a switch-style condition table. It is a short-hand construct that allows different routes to be selected based upon the results of a single XQuery expression. You can nest multiple levels in the stage editor. Identify target services for messages and configure how the messages are routed to these services.

This is a terminal action, which means you cannot add another action after this one. However, this action can contain request and response actions. For more information on routing, see [Modeling Message Flow in Oracle Service Bus](#).

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a routing table to a route node:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the **Route Node** icon, then click **Edit Route**. The Edit Stage Configuration page is displayed.
3. Click the **Add an Action** icon, then select **Communication > Routing Table**. The routing table action is displayed.

4. From the **Operator** list, select a comparison operator, then enter a value expression in the adjacent field.
5. Click **Service**. The Select Service page is displayed.
6. Select a service from the list, then click **Submit**.
7. If you want to invoke an operation on the service, select an operation from the **Operation** list
8. If you want the outbound operation to be the same as the inbound operation, select the **Use inbound operation for outbound** check box.
9. In the **Request Actions** field, click **Add an Action** to add an action, then select an action that you want to associate with the service. You can add more than one action.
10. In the **Response Actions** field, click **Add an Action** to add an action, then select an action that you want to associate with the service. You can add more than one action.

To learn more about the types of request and response actions you can add, see [Adding and Editing Pipeline Actions in the Console](#).
11. To insert a new case, click the **Case** icon, then select **Insert New Case**.
12. Repeat steps 2-7 for the new case. You can click the **Case** icon, then select **Insert Default Case** to add a default case at the end whose routes are selected if none of the preceding cases is satisfied.
13. Click **Save**.
14. On the Edit Message Flow page, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in the Console](#).
15. Click **Save** to commit the updates in the current session.
16. To end the session and deploy the configuration to the runtime, click **Activate** under Change Center.

Adding For-Each Actions in the Console

Use the for-each action to iterate over a sequence of values and execute a block of actions.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a for-each action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Flow Control > For Each**.
3. Enter variable names in the **variable** fields, click **XPath** to open the XPath editor to create an XPath expression, and configure the actions in the **Do ()** loop.

4. Click **Save** to commit the updates in the current session.

Adding If-Then Actions in the Console

Use an if-then action to perform an action or set of actions conditionally, based on the Boolean result of an XQuery expression.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add an if-then action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Flow Control > If...Then...**
3. Click **Condition** to display the XQuery Condition Editor page.

The condition you create is used as the test that is executed before the `then ()` clause is entered, per standard if-then logic. See [Creating and Editing Inline XQuery and XPath Expressions](#).

4. When you finish editing the XQuery condition, click **Add an Action**, then select an action that you want to associate with the condition. To learn more about the type of action you want to add, see [Adding and Editing Pipeline Actions in the Console](#).

In the route node, you can select only the routing, dynamic routing, or routing table actions. However, these actions can contain request and response actions inside of them.

5. As your logic requires, click the **If...Then...** icon, then click **Add else-if Condition** or **Add else Condition** to add `else-if` conditions or `else` conditions. Click **Add an Action** to associate actions with these conditions.

Condition actions can be nested.

6. Click **Save** to commit the updates in the current session.

Adding Raise Error Actions in the Console

Use the raise error action to raise an exception with a specified error code (a string) and description.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a raise error action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Flow Control > Raise Error**.

3. In the **error code** field, enter the error code you want to raise.
4. In the **error message** field, enter a description of the error code.
5. Click **Save** to commit the updates in the current session.

Transactions

If a service is transactional, a triggered Raise Error action aborts the transaction in the request (asynchronous) or in either the request or response (synchronous). For example, you may introspect messages and determine conditions under which a Raise Error action should occur even if no SOAP fault occurs, and Raise Error causes the transaction to be aborted.

Adding Reply Actions in the Console

Use the reply action to specify that an immediate reply be sent to the invoker. The reply action can be used in the request, response or error pipeline. You can configure it to result in a reply with success or failure. In the case of reply with failure where the inbound transport is HTTP, the reply action specifies that an immediate reply is sent to the invoker.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a reply action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Flow Control > Reply**.
3. Select **With Success** to reply that the message was successful, or select **With Failure** to reply that the message has a fault.

Reply With Failure will cause a transaction, if started by Service Bus, to be aborted.

4. Click **Save** to commit the updates in the current session.

Adding Resume Actions in the Console

Use the resume action to resume message flow after an error is handled by an error handler. This action has no parameters and can only be used in error pipelines.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a resume action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Flow Control > Resume**.

After you finish:

When you complete the configuration of this action, continue by configuring other actions or by saving your configuration, as described in [Adding and Editing Pipeline Actions in the Console](#).

Adding Skip Actions in the Console

Use the skip action to specify that at runtime, the execution of this stage is skipped and the processing proceeds to the next stage in the pipeline. This action has no parameters and can be used in the request, response or error pipelines.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a skip action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Flow Control > Skip**.

Adding Assign Actions in the Console

Use the assign action to assign the result of an XQuery expression to a context variable.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add an assign action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Message Processing > Assign**.
3. Click **Expression**. The XQuery Expression Editor page is displayed. The XQuery expression is used to create the data that will be assigned to the named variable. See [Creating and Editing Inline XQuery and XPath Expressions](#).
4. When you finish editing the expression, enter a context variable in the variable field. To learn more about context variables, see [Inbound and Outbound Variables and Constructing Messages to Dispatch](#).
5. Click **Save** to commit the updates in the current session.

Adding Delete Actions in the Console

Use the delete action to delete a context variable or a set of nodes specified by an XPath expression. The delete action is one of a set of update actions.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a delete action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. To delete a context variable, select the **Variable** option, then enter the name of a context variable in the **Variable** field.

Alternatively, to delete all nodes selected by an XPath expression, select the **XPath** radio button, then click **XPath**. The XPath Expression Editor page is displayed. See [Creating and Editing Inline XQuery and XPath Expressions](#). After you save the expression, enter a context variable in the **variable** field.

Adding Insert Actions

Use the insert action to insert the result of an XQuery expression at an identified place relative to nodes selected by an XPath expression. The insert action is one of a set of update actions.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add an insert action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Message Processing > Insert**.
3. Click **Expression** to edit an XQuery expression. The XQuery expression is used to create the data that will be inserted at a specified location in a named variable. The XQuery Expression Editor page is displayed. See [Creating and Editing Inline XQuery and XPath Expressions](#).
4. When you finish editing the expression, select the relative location from the list. The relative location is used to control where the insert is performed relative to the result of the XPath expression:
 - **Before:** As sibling before each element or attribute selected by the XPath expression
 - **After:** As sibling after each element or attribute selected by the XPath expression
 - **As first child of:** As first child of each element identified by the XPath expression. An error occurs if the result of the XPath returns attributes.
 - **As last child of:** As last child of each element identified by the XPath expression. An error occurs if the XPath returns attributes.
5. Click **XPath**. The XPath Expression Editor page is displayed. See [Creating and Editing Inline XQuery and XPath Expressions](#).

Valid configurations include those in which:

- XQuery and XPath expressions both return elements.
 - The XQuery and XPath expressions both return attributes—in which case, the XQuery expression must return attributes.
6. When you finish editing the XPath expression, enter a context variable in the in variable field. The XPath evaluates the contents of this variable.
 7. Click **Save** to commit the updates in the current session.

Adding Java Callout Actions in the Console

Use the Java callout action to invoke a Java method, or EJB business service, from within the pipeline.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a Java callout action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Message Processing > Java Callout**.
3. Click **Method**. The Select a JAR page is displayed. Select a JAR resource from the list. The Select a Class and Method page is displayed.
4. From the list of Java classes listed, click the + beside the appropriate class, to display a list of methods. Select a method and click **Submit**. The Java callout action is displayed on the Edit Stage page, as follows:

- **Method** is replaced by the name of the Java method you selected in steps 2 and 3. This name is a link to the Select a Class and Method page. You can click this link to change your selection of Java method.

The method must be a static method.

- **Parameters:** An **Expression** link to the XQuery Expression Editor page is provided for each argument the Java method requires. A label for each link indicates the data type for the argument, which will be one of the following:
 - `java.lang.String`
 - Primitive types, and their corresponding class types (for example, `int` versus `java.lang.Integer`)
 - `java.lang.BigDecimal`, and `java.lang.BigInteger` (these types are used in financial calculations where round-off errors or overflows are not tolerable)
 - only `org.apache.xbeans.XmlObject` and no typed xml beans.
 - `byte[]`
 - `java.lang.String[]` (INPUT ONLY)

- `XmlObject[]` (INPUT ONLY)
- `javax.activation.DataSource`
- **Result:** A **Result** field in which you enter the variable to which the result is to be assigned. The label for the field indicates the data type of the result.

Note:

If the result is a byte array (the only possible array returned), the binary-content XML element is returned.

- **Return Parameter as Reference:** This option makes the return value of a Java Callout invocation a `<java-content ref="jcid">` reference element regardless of its actual type, where `jcid` is the key to the object in the pipeline object repository. In the Result value field, enter the name of the variable to contain the java-content reference. This option lets you work with a referenced object in the pipeline in addition to the pipeline XML for providing passthrough, performing message enrichment with Java Callout and inline actions, or performing message transformation between Java and non-Java transports. For more information, see [Sending and Receiving Java Objects in Messages](#).
- **Attach a Service Account:** A **Service Account** link allows you to specify an optional Service Account if there is a security context for this Java method. To learn more about security contexts and service accounts, see [Working with Service Accounts](#).

In the case of fixed and mapped service accounts, the `userid/password` from the service account is authenticated in the local system and the security context propagated to the Java callout. In the case of `passthru`, the security context is propagated to the Java callout. This context is the message level context if defined (with WS-Security). Else it is the transport level context.

5. Under Parameters, click **Expression**. The XQuery Expression Editor page is displayed. Use the XQuery Expression Editor to provide the arguments required by the Java method. See [Creating and Editing Inline XQuery and XPath Expressions](#).

If the type of the input value you enter does not match the declared input argument type, Service Bus tries to automatically typecast input values to the declared type of the input argument. For example a string value of "123" will be converted to integer 123 if the declared type of the input argument is java primitive `int`.

6. In the **Result** field, assign a variable for the result returned by the Java method.
7. If there is a security context for the Java method, select the check box and click **Service Account**. The Select Service Account page is displayed. Select the required service account from the list, and click **Submit**.
8. Click **Save** to commit the updates in the current session.

Adding MFL Translate Actions in the Console

Use the MFL (Message Format Language) translate action to convert message content from XML to non-XML, or vice versa, in the message pipeline. An MFL is a specialized

XML document used to describe the layout of binary data. It is an Oracle proprietary language used to define rules to translate formatted binary data into XML data, or vice versa. See [Defining Data Structures with Message Format Language](#).

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add an MFL translate action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Message Processing > MFL Translate**.
3. From the **Apply MFL Translation** list, select **XML to Non-XML** or **Non-XML to XML**, according to your requirement.
4. Click **Expression**. Using the XQuery Expression Editor, specify the variable on which the MFL translation action is to be performed. This input must be text or binary when translating to XML, and must be XML when translating to non-XML. Binary content in the message context is represented by the binary-content XML element. This XML should be the result of the XQuery expression when the input needs to be binary. See [Creating and Editing Inline XQuery and XPath Expressions](#).
5. Select one of the following options:
 - **MFL Resource**: click the **resource** link. The Select MFL page is displayed. Select the static MFL resource that will perform the MFL translate action.
 - **MFL Resource from**: click the **Expression** link. The XQuery Expression Editor page is displayed. Using the XQuery Expression Editor, create or edit an XQuery expression to dynamically specify an MFL resource that will perform the translate action, in the format project/folder/MFLresourceName. See [Creating and Editing Inline XQuery and XPath Expressions](#).
6. In the **Assign to Variable** field, enter the name of the variable to which the result of this translate action is to be assigned. The result will be a binary-content XML element.
7. Click **Save** to commit the updates in the current session.

Adding nXSD Translate Actions

Use the nXSD translate action to convert message content from XML to native format data, or vice versa, in the message pipeline. See "Native Format Builder Wizard" in *Understanding Technology Adapters* for information on creating native schemas used for translation.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add an nXSD translate action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Message Processing > nXSD Translate**.
3. From the **Apply nXSD Translation** list, select **XML to Native** or **Native to XML**, according to your requirement.
4. Click **Expression**. Using the XQuery/XSLT Expression Editor, specify the variable on which the nXSD translation action is to be performed. This input would be native format when translating to XML, and XML when translating to native data format. See [Creating and Editing Inline XQuery and XPath Expressions](#).
5. Select one of the following options:
 - **nXSD Resource**: click the **resource** link. The Select a XML Schema page appears. Select the schema (.xsd) file corresponding to the native schema.
 - **nXSD Resource from**: click the **Expression** link. The XQuery Expression Editor page is displayed. Using the XQuery Expression Editor, create or edit an XQuery expression to dynamically specify a native schema. See [Creating and Editing Inline XQuery and XPath Expressions](#).
6. In the **Assign output to** field, select **variable**, and enter the name of the variable to which the result of this translate action is to be assigned. You can alternatively assign the output to **content of \$body**.
7. Click **Save** to commit the updates in the current session.

Adding Rename Actions in the Console

Use the rename action to rename elements selected by an XPath expression without modifying the contents of the element. The rename action is one of a set of update actions.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a rename action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Message Processing > Rename**.
3. Click **XPath**. The XPath Expression Editor page is displayed. The XPath expression is used to specify the data (in the named variable) that will be renamed. See [Creating and Editing Inline XQuery and XPath Expressions](#).
4. In **variable** field, enter the context variable that holds the element you want to rename.
5. Do one of the following:

- To rename selected elements using a localname, select the first **localname** option, then enter a local name in the **localname** field.
 - To rename selected elements using a namespace, select the first **namespace** option, then enter a namespace in the **namespace** field.
 - To rename selected elements using a local name and namespace, select the **localname and namespace** radio button, then enter a local name and namespace in the **localname and namespace** fields.
6. Click **Save** to commit the updates in the current session.

Adding Replace Actions in the Console

Use a replace action to replace a node or the contents of a node specified by an XPath expression. The node or its contents are replaced with the value returned by an XQuery expression. A replace action can be used to replace simple values, elements and even attributes. An XQuery expression that returns nothing is equivalent to deleting the identified nodes or making them empty, depending upon whether the action is replacing entire nodes or just node contents. The replace action is one of a set of update actions.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a replace action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Message Processing > Replace**.
3. Click **XPath**. The XPath Expression Editor page is displayed. The XPath expression is used to specify the data (in the named variable) that will be replaced. See [Creating and Editing Inline XQuery and XPath Expressions](#).
4. When you finish editing the XPath expression, enter a context variable in the **variable** field.
5. Click **Expression**. The XQuery Expression Editor page is displayed. The XQuery expression is used to create the data that replaces the data specified by the XPath in the named variable. See [Creating and Editing Inline XQuery and XPath Expressions](#).
6. When you finish editing the XQuery expression, select one of the options:
 - **Replace entire node**—to specify that the nodes selected by the XPath expression you defined are replaced along with all of its contents
 - **Replace node contents**—to specify that the node is not replaced; only the contents are replaced.

Note:

Selecting the **Replace node contents** option and leaving the **XPath** field blank is more efficient than selecting the **Replace entire node** option and setting the XPath to `./*`

7. Click **Save** to commit the updates in the current session.

Adding Validate Actions in the Console

Use a validate action to validate elements selected by an XPath expression against an XML schema element or a WSDL resource. You can validate global elements only; Service Bus does not support validation against local elements. You can also choose to dynamically select the XML schema element or WSDL resource, at runtime, based on the result of an XQuery expression.

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a validate action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Message Processing > Validate**.
3. Click **XPath**. to construct an XPath expression that specifies the elements to be validated. See [Creating and Editing Inline XQuery and XPath Expressions](#). When you are finished constructing the expression in the XPath Expression Editor, click **Save** to insert the expression on the Edit Stage Configuration page.
4. In the **in variable** field, enter the name of the variable to hold the element to be validated.
5. Select one of the following options:
 - **Against Resource:** click the **resource** link, then select **WSDL** or **Schema**. From the WSDL Browser or XML Schema Browser, do the following:
 - a. Select the WSDL file or XML schema
 - b. Select the WSDL file or XML schema type or element
 - c. Click **Submit**.
 - **Against Resource from Expression:** click the **Expression** link. The XQuery Expression Editor page is displayed. Using the XQuery Expression Editor, create or edit an XQuery expression to dynamically specify a WSDL file or schema resource. See [Creating and Editing Inline XQuery and XPath Expressions](#).

The following is an example of dynamically specifying a WSDL resource:

```
<validate xmlns="http://www.bea.com/wli/sb/context">
  <wsdl>default/MyWSDL</wsdl>
  <schemaType>
    <namespaceURI>http://openuri.org</namespaceURI>
    <localname>MyType</localname>
```

```

    </schemaType>
  </validate>

```

The following is an example of dynamically specifying a schema resource:

```

<validate xmlns="http://www.bea.com/wli/sb/context">
  <schema>{dvm:lookup('SBProject/dvm/DVM_Validation', 'operation',
$operationName, 'validationSchema','default')}</schema>
  <schemaElement>
    .....<namespaceURI>"http://www.server.com/date</namespace>
      <localname>{dvm:lookup('SBProject/dvm/DVM_Validation', 'operation',
$operationName, 'validationElement','default')}</localname>
    </schemaElement>
  </validate>

```

Note that the schema or WSDL selected has to be a resource and so must have been imported into OSB as a Schema/WSDL resource. You cannot point directly to a file or URL. The XML fragment that is required is created via an xquery expression, letting you enter any kind of xquery expression to create that XML. In the examples above, the XML provided is a constant XML, illustrating what kind of XML is expected.

For example, the name of the WSDL could be coming in a variable and you extract that name to find the resource name. Another example is to have one xquery module resource with a function that can do a lookup of the XML fragment given one or multiple keys. In this case, in the validate expression you can evaluate the keys required and make a call to that xquery module function. This allows you to externalize and manage all those dynamic entries in that one xquery module along with the lookup algorithm.

6. To save the result of this validation (a boolean result), select **Save result of validation in variable** and enter the name of the variable in which you want to save the result.

Alternatively, to raise an error if the element fails validation against the WSDL file or XML schema element, select **Raise Error** on validation failure.

7. Click **Save** to commit the updates in the current session.

Adding Alert Actions in the Console

Use the alert action to generate alerts based on message context in a pipeline, to send to an alert destination. Unlike SLA alerts, notifications generated by the alert action are primarily intended for business purposes, or to report errors, and not for monitoring system health. Alert destinations should be configured and chosen with this in mind. To learn more about alert destinations, see [Working with Alert Destinations](#).

If pipeline alerting is not enabled for the service or at the domain level, the configured alert action is bypassed during message processing.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add an alert action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).

2. Click the appropriate icon, then select **Add an Action > Reporting > Alert**.
3. Click **Destination**. The Select Alert Destination page is displayed. Select the required alert destination from the list and click **Submit**.

By default, the alert will always go to the Administration Console.

4. Click **Expression**. The XQuery Expression Editor page is displayed. You specify the message context to be added to the alert message through XQuery expressions on context variables. See [Creating and Editing Inline XQuery and XPath Expressions](#).
5. In the **alert summary** field, enter a short description of the alert. This will be the subject line in the case of an Email notification, and can contain no more than 80 characters. If no description is provided, a predefined subject line that reads, "Oracle Service Bus Alert", will be used instead.
6. In the **severity level** list, select a severity level for this alert from among: **Normal**, **Warning**, **Minor**, **Major**, **Critical**, and **Fatal**.
7. Click **Save** to commit the updates in the current session.

Adding Log Actions in the Console

Use the log action to construct a message to be logged and to define a set of attributes with which the message is logged.

Before you begin

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

Note:

To see log data in the log file or standard out (server console), WebLogic Server logging must be set to the following severity levels:

- Minimum severity to log: Info
- Log file: Info
- Standard out: Info

For information on setting log severity levels, see "Using Log Severity Levels" in *Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server*.

To add a log action:

1. Be sure Logging is enabled globally. For more information, see "Configuring Operational Settings at the Global Level" in *Administering Oracle Service Bus*.
2. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
3. Click the appropriate icon, then select **Add an Action > Reporting > Log**.
4. Click **Expression**. The XQuery Expression Editor page is displayed. You specify the message context to be logged through XQuery expressions on context variables. See [Creating and Editing Inline XQuery and XPath Expressions](#).

5. In the **Annotation** field, enter notes for this log action. These notes are logged along with the result of the previously defined expression.
6. In the **severity level** list, select one of the options.

Table 14-7 Log Action Severity Levels

Severity Level	Typical Usage
Info	Used for reporting normal operations; a low-level informational message.
Warning	A suspicious operation or configuration has occurred but it might not affect normal operation.
Error	A user error has occurred. The system or application can handle the error with no interruption and limited degradation of service.
Debug	While your application is under development, you might find it useful to create and use messages that provide verbose descriptions of low-level activity within the application.

Make sure the Log severity level on the proxy service's operational settings is the same as the log action severity level. For information on proxy service operational settings, see "Available Operational Settings" in *Administering Oracle Service Bus*.

7. Click **Save** to commit the updates in the current session.

Adding Report Actions in the Console

Use the report action to enable message reporting for a proxy service.

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To add a report action:

1. Navigate to where you want to add the action, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the appropriate icon, then select **Add an Action > Reporting > Report**.
3. Click **Expression**. The XQuery Expression Editor page is displayed. See [Creating and Editing Inline XQuery and XPath Expressions](#). The XQuery expression is used to create the data that will be reported to the Service Bus dashboard.
4. When you finish editing the XQuery expression, click **Add a Key**. Two fields are displayed: a **Key Name** field and a **Key Value** field, which includes an XPath link that you can click to edit an XPath expression and an in variable field in which you can enter a context variable.

You use key value pairs to extract key identifiers from any message context variable or message payload, and ignore the rest of the message. The keys are a convenient way to identify a message. They are displayed as report indexes in the Reporting module. See "Working with Message Reports" in *Administering Oracle Service Bus*.

- a. Enter a key name in the **Key Name** field.
- b. Click **XPath**. The Edit an XPath Expression page is displayed. See [Creating and Editing Inline XQuery and XPath Expressions](#).
- c. Enter a context variable in the **variable** field.
- d. To add more key values, click the **Key** icon, then select **Add a Key**. To delete a key, click the **Key** icon, then select **Delete this Key**.

For example, consider a report action configured on an error handler in a stage. The action reports the contents of the `fault` context variable in the event of an error. The report action is configured as follows:

- Key name = `errorCode`
- Key value = `./ctx:errorCode` in variable `fault`

Each time this action is executed at runtime, a message is reported through the Reporting Data Stream. The following table shows the results after the report action is executed twice.

Report Index	DB TimeStamp	Inbound Service	Error Code
errorCode=OSB-38250 5	04/26/07 9:45 AM	MortgageBroker/ProxySvc3/ loanGateway3	OSB-382505
errorCode=OSB-38250 5	04/26/07 9:45 AM		OSB-382505

Adding Error Handlers in the Console

You can configure error handling at the message flow, pipeline, route node, and stage level. Configure error handlers on the Edit Error Handler page. You must always add at least one stage to the page to specify how the error handler will work.

Adding Pipeline Error Handlers in the Console

Before you begin

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

The instructions also assume you have created a pipeline pair node, as explained in [How to Add Pipeline Pairs to Pipelines](#).

To add a pipeline error handler:

1. Navigate to the pipeline pair node containing the pipeline to which you want to add an error handler. If the pipeline pair is not already expanded, click the plus sign next to the icon to display the pipelines.
2. Click the **Request Pipeline** icon or the **Response Pipeline** icon, then click **Add Pipeline Error Handler**. The Edit Error Handler page is displayed.
3. Click the **Error Handler** icon, then click **Add Stage**.

4. Click the **Stage** icon, click **Edit Stage**. The Edit Stage Configuration page is displayed.
5. Click **Add an Action**, then select the action you want to add.

An error handler is a pipeline and is therefore configured like any other pipeline. For example, you can use the Publish action to send error notifications to other services, use the Assign action to modify the context variables, and so on. To learn more about the type of action you want to add, see the appropriate procedure in [Adding and Editing Pipeline Actions in the Console](#). There is no restriction on what actions may be chained together.

Three commonly-used error actions are Raise Error, Reply, and Resume.

6. Add other actions and make other edits on the Edit Stage Configuration page, as desired.
7. On the Edit Stage Configuration page, click **Save** to commit the updates in the current session.
8. On the Edit Error Handler page, click **Save** to commit the updates in the current session.

Adding Stage Error Handlers in the Console

Before you begin

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#). The instructions also assume you have added a stage to a pipeline, as explained in [How to Add Stages to Pipelines in the Console](#).

To add a stage error handler:

1. Navigate to the stage to which you want to add error handling.
2. Click the **Stage** icon, then click **Add Stage Error Handler**. The Edit Error Handler page is displayed.
3. Click the **Error Handler** icon, then click **Add Stage**.
4. Click the **Stage** icon, then click **Edit Stage**. The Edit Stage Configuration page is displayed.
5. Click **Add an Action**, then select the action you want to add.

An error handler is a pipeline and is therefore configured like any other pipeline. For example, you can use the Publish action to send error notifications to other services, use the Assign action to modify the context variables, and so on. To learn more about the type of action you want to add, see the appropriate procedure in [Adding and Editing Pipeline Actions in the Console](#). There is no restriction on what actions may be chained together.

Three commonly-used error actions are Raise Error, Reply, and Resume.

6. Add other actions and make other edits on the Edit Stage Configuration page, as desired.
7. Click **Save** to commit the updates in the current session.

Adding Route Node Error Handlers in the Console

Before you begin

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

The instructions also assume you have created a route node, as explained in [How to Add Route Nodes to Pipelines in the Console](#).

To add a route node error handler:

1. Click the **Route Node** icon, then click **Add Error Handler**. The Edit Error Handler page is displayed.
2. Click the **Error Handler** icon, then click **Add Stage**.
3. Click the **Stage** icon, then click **Edit Stage**. The Edit Stage Configuration page is displayed.
4. Click **Add an Action**, then select the action you want to add.

Since an error handler is another pipeline, it is configured like any other pipeline. For example, the Publish action may be used to send error notifications to other services, the Assign action may be used to modify the context variables, and so on. To learn more about the type of action you want to add, see the appropriate procedure in [Adding and Editing Pipeline Actions in the Console](#). There is no restriction on what actions may be chained together.

Three commonly-used error actions are Raise Error, Reply, and Resume.

5. Add other actions and make other edits on the Edit Stage Configuration page, as desired.
6. On the Edit Stage Configuration page, click **Save** to commit the updates in the current session.
7. On the Edit Error Handler page, click **Save** to commit the updates in the current session.

Editing Error Handlers in the Console

Before you begin

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To view and change an error handler:

Do one of the following:

Table 14-8 Viewing and Changing the Error Handler

To...	Complete This Step...
View and change the pipeline error handler	Click the appropriate Request Pipeline icon or the Response Pipeline icon, then click Edit Pipeline Error Handler . The Edit Error Handler page is displayed. See Adding Pipeline Error Handlers in the Console .
View and change the route node error handler	Click the appropriate Route Node icon, then click Edit Route Error Handler . The Edit Error Handler page is displayed. See Adding Route Node Error Handlers in the Console .
View and change the stage error handler	Click the appropriate Stage icon, then click Edit Stage Error Handler . The Edit Error Handler page is displayed. See Adding Stage Error Handlers in the Console .

Disabling an Action or a Stage in the Console

You can choose to disable an action or a stage in a pipeline. A disabled action or stage is skipped from the pipeline execution. When you disable an action or a stage, all the nested actions, if any, are disabled automatically. A disabled stage or action is not validated at design time.

Note:

If a disabled stage has an error handler, then the error handler is also disabled.

You can still edit the configuration of a disabled action or stage. Refactoring also takes place for disabled actions and stages. This means that if there is a call to a service in the disabled action or stage, and the service gets renamed, then the service callout is automatically updated.

You can re-enable a disabled stage or action at any time, and the action or stage is no longer skipped in the pipeline.

Disabling an Action on the Pipeline

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To disable an action in the pipeline:

1. Navigate to the action that you wish to edit, as described in [Adding and Editing Pipeline Actions in the Console](#).

2. Click the appropriate action icon, then select **Disable this Action**.

The action icon is dimmed, and a Disabled icon appears next to the action.

3. Click **Save** to save the changes.

Re-Enabling an Action in the Pipeline

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To re-enable an action in the pipeline:

1. Navigate to the action that you wish to edit, as described in [Adding and Editing Pipeline Actions in the Console](#).
2. Click the disabled action icon, then select **Enable this Action**.

The dimmed action icon returns to normal, and the action is enabled.

3. Click **Save** to save the changes.

Disabling a Stage in the Pipeline

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To disable a stage in the pipeline:

1. On the Edit Message Flow page, click the stage icon that you wish to disable.
2. Select **Disable Stage** from the context menu that appears.

The stage icon is dimmed, and a Disabled icon appears next to the stage.

3. Click **Save** to save the changes.

Re-Enabling a Stage in the Pipeline

Before you begin:

These instructions assume you are already editing a pipeline in the Edit Message Flow page, as explained in [Viewing and Editing Pipelines in the Console](#).

To re-enable a stage in the pipeline:

1. On the Edit Message Flow page, click the disabled stage icon.
2. Select **Enable Stage** from the context menu that appears.

The dimmed stage icon returns to normal, and the stage is enabled.

3. Click **Save** to save the changes.

Working With Expression Editors in Oracle Service Bus Console

This chapter describes how to use XQuery and XPath expressions in pipelines, or message flows.

In the pipeline, you can assign XQuery expressions to message context variables, assign if-then actions based on the Boolean result of an XQuery expression, insert the result of an XQuery expression at an identified place relative to an XPath expression, specify the message context that you want to log through XQuery expressions on context variables, and so on.

The XQuery Expression Editor, the XQuery Condition Editor, and the XPath Expression Editor are available in the appropriate context in message flows to construct the kind of expression called for in the context.

This chapter contains the following sections:

- [Creating and Editing Inline XQuery and XPath Expressions](#)
- [Understanding XQuery Editor Layouts and Tasks](#)
- [Building Expressions in the Editor Workspace Text Fields](#)
- [Creating Namespaces to Use in Inline Expressions](#)
- [Creating Variable Structures in the XQuery Editors](#)
- [Creating Custom XPath Functions in the XQuery Editors](#)
- [Binding External XQuery Resources to Inline XQueries](#)
- [Binding External XSLT Resources to Inline XQueries](#)
- [Binding Dynamic XQuery Expressions to Inline XQueries](#)
- [Binding Dynamic XSLT Expressions to Inline XQueries](#)
- [Entering XQuery Comparison Expressions Using the Builder Option](#)
- [Entering Unary Expressions Using the Builder Option](#)

Creating and Editing Inline XQuery and XPath Expressions

When you add actions to stages or route nodes on the Edit Stage Configuration page, a skeleton structure is displayed on the page that prompts for configuration details.

[Figure 15-1](#) shows an example.

Figure 15-1 Example of Action Configuration Skeleton

Replace `<XPath>` in variable with `<Expression>`

Whenever it is appropriate for the context, the skeleton provides links for accessing the XQuery editors, where you can construct expressions that will be executed inline, as required by the context in the action.

Before you begin:

These instructions assume you are creating or editing an action in the Edit Stage Configuration page of a pipeline stage, an error handler stage, or a route node. See [Viewing and Editing Pipelines in the Console](#). and [Adding and Editing Pipeline Actions in the Console](#).

To create or modify an inline expression:

1. On the Edit Stage Configuration page, locate the place in the action where you want to add or edit the expression.
2. Click the expression link to open the editor that is appropriate for the context. When an expression has not yet been defined, the link tells what kind of expression you can use in that position:
 - Click **Expression** to create an XQuery expression. You can also import an XQuery or XSLT resource created outside Service Bus, then bind it to the inline XQuery.
 - Click **Condition** to create an XQuery conditional expression for an if-then action.
 - Click **XPath** to create an XPath expression for a message context variable.

When an expression has already been defined in a position, the **Expression**, **Condition**, or **XPath** link is replaced by a link that shows expression itself, for example `true()` icon, or a fragment of the expression it is too long to fit, for example `$body/urn:POSTa...` icon. Click the expression (or expression fragment) to open the expression in the appropriate editor.

3. Build the expression, as described in the following topics:
 - [Building Expressions in the Editor Workspace Text Fields](#)
 - [Creating Namespaces to Use in Inline Expressions](#)
 - [Creating Variable Structures in the XQuery Editors](#)
 - [Creating Custom XPath Functions in the XQuery Editors](#)
 - [Binding External XQuery Resources to Inline XQueries](#)
 - [Binding External XSLT Resources to Inline XQueries](#)
 - [Binding Dynamic XQuery Expressions to Inline XQueries](#)
 - [Entering XQuery Comparison Expressions Using the Builder Option](#)
 - [Entering Unary Expressions Using the Builder Option](#)

4. Optionally, do either or both of the following:
 - Click **Validate** to validate the expression.
 - Click **Test** to test the expression. See [XQuery Transformation Testing Prerequisites and Guidelines](#).
5. Click **Save** to close the editor and insert the expression in the action.

Understanding XQuery Editor Layouts and Tasks

The XQuery Expression Editor, the XQuery Condition Editor, and the XPath Expression Editor are each composed of the following components:

- [Palettes](#)
- [Workspace](#)
- [Property Inspector](#)

Palettes

The left panel of the each editor contains the palettes listed below. In any of the editors, click the name of a palette to display it. Each palette contains entities that you can insert into expressions in the editors.

- The Namespace Definitions palette lists default Service Bus namespaces, variable namespaces, and user-defined namespaces. You can define new namespaces, which are then added to the list of user-defined namespaces.

For more information, see:

- [Building Expressions in the Editor Workspace Text Fields](#)
- [Creating Namespaces to Use in Inline Expressions](#)

- The XQuery Functions palette lists a set of standard XQuery functions. When you insert a function into an expression, placeholders are used for parameter values you must supply.

For more information, see:

- [Building Expressions in the Editor Workspace Text Fields](#)
- [Creating Custom XPath Functions in the XQuery Editors](#)
- [Binding External XQuery Resources to Inline XQueries](#)

- The Variable Structures palette provides a set of tools for inserting variables and paths to the variables, using XPath expressions.

Variable structures are graphical representations of variables or variable paths that are displayed in the editor. They can help you visualize the variable structure, and you can use them to construct inline XQuery expressions that reference the content of the variable.

Note:

Variable structures do not create variables. Variables are created at runtime as the target of the Assign action in the stage.

Service Bus provides several predefined message context variables (*attachments*, *body*, *header*, *outbound*, and *inbound*), whose contents you can display as variable structures. You can also define your own variable structures.

For more information, see

- [Building Expressions in the Editor Workspace Text Fields](#)
- [Creating Variable Structures in the XQuery Editors](#)

Workspace

The right side of the page provides a workspace for constructing the XQuery expression, XQuery condition, or XPath. The workspace is different in the three editors.

Property Inspector

In all three editors, the Property Inspector is displayed on the bottom right of the page. When you select an item from one of the palettes to add to the expression, that item appears in the Property Inspector. You can then paste the item into the workspace. See [Building Expressions in the Editor Workspace Text Fields](#).

Building Expressions in the Editor Workspace Text Fields

The XQuery Expression Editor, the XQuery Condition Editor, and the XPath Expression Editor all provide text fields in which you can build expressions by typing directly or by pasting items from the palettes.

Before you begin:

These instructions assume you are creating or editing an expression in the XQuery Expression Editor, XQuery Condition Editor, or XPath Expression Editor, as described in [Creating and Editing Inline XQuery and XPath Expressions](#).

To build an expression in a text field:

1. Display the panel containing the text field. Depending on the editor, do one of the following:
 - In the XQuery Expression Editor, select **XQuery Text** (located under the workspace button bar), if it is not already selected.
 - In the XQuery Condition Editor, select **Text** (located under the workspace button bar), if it is not already selected.
 - In the XPath Expression Editor, you do not have to select anything, because there are no options for selecting other tools.

Note:

Selecting any of the above links displays a text field where you can create a complete expression appropriate for the context. However, the tools and techniques described in this topic can be used wherever text fields are provided in the editors, for example when binding variables from imported resources to the inline expression, as described in [Binding External XQuery Resources to Inline XQueries](#) and [Binding External XSLT Resources to Inline XQueries](#).

2. If desired, type or paste an expression or expression fragment into the field. If you create the complete expression this way, skip to step 7, below. Otherwise, proceed to the next step.
3. Select the palette containing the item(s) you want to add to the expression, and locate the item you want to add, as described in [Table 15-1](#), below.

Table 15-1 Palettes

Palette	Description and Use
Namespace Definitions	<p>Lists default Service Bus namespaces, variable namespaces, and user-defined namespaces. Namespace abbreviations are listed when defined.</p> <p>Scroll through the lists to find the desired namespace.</p> <p>You can also define a namespace. See Creating Namespaces to Use in Inline Expressions.</p>
XQuery Functions	<p>Contains a set of standard XQuery functions, organized alphabetically and by type, including any custom functions you have developed. To expand or collapse nodes in the tree, click the plus sign (+) or minus sign (-).</p> <p>See also:</p> <ul style="list-style-type: none"> • Transforming Data with XQuery.

Table 15-1 (Cont.) Palettes

Palette	Description and Use
Variable Structures	<p>Displays variables and their contents as trees, which can help you to visualize.</p> <p>You can view a variable structure and its contents.</p> <ul style="list-style-type: none"> • Select the name of the structure from the list at the top of the palette. The list displays Built-in message context variables (<code>attachments</code>, <code>body</code>, <code>header</code>, <code>outbound</code>, and <code>inbound</code>), as well as any user-defined structures, organized by type (XML Type, Service Interface, and Simple Type). • To expand or collapse nodes in the tree, click the plus sign (+) or minus sign (-). <p>You can also define your own variable structures. See Creating Variable Structures in the XQuery Editors.</p> <p>Variable structures do not create variables. Variables are created at runtime as the target of the Assign action in the stage.</p> <p>When you insert an item from the a variable structure tree into the text field, it is inserted as an XPath expression that describes the path.</p> <p>See also Transforming Data with XQuery.</p>

4. Paste the desired item into the text field using any of the methods shown below in [Table 15-2](#):

Table 15-2 Ways to Paste Items Into the Editor Text Fields

From this palette...	Do this...
Namespace Definitions palette	<p>Use standard mouse or keyboard select, copy, and paste a namespace, for example:</p> <ol style="list-style-type: none"> Select the entire namespace string (or its abbreviation, if one exists) by dragging the mouse pointer over the string. Press Ctrl-C to copy the string. Click the location in the text field where you want to insert the namespace. Press Ctrl-V to paste the string.
XQuery Functions palette or Variable Structures palette	<p>Drag an item from the palette to the text field.</p> <p>Note: Dragging from the palette to the workspace is supported only in Internet Explorer.</p>

Table 15-2 (Cont.) Ways to Paste Items Into the Editor Text Fields

From this palette...	Do this...
XQuery Functions palette or Variable Structures palette	<ol style="list-style-type: none"> a. Click an item in the palette. The item is displayed in the Property Inspector pane: Functions are displayed with placeholders for any values you have to supply. Variables and their attributes are displayed as XPath expressions. b. Click in the text field where you want to insert the item c. Click Copy Property to paste the item into the location selected in the text field.
XQuery Functions palette or Variable Structures palette	<ol style="list-style-type: none"> a. Click an item in the palette. The item is displayed in the Property Inspector pane. b. Select and copy the item in the Property Inspector, using standard keyboard or mouse actions. c. Select a location in the text field, and paste the item into the text field, using standard keyboard or mouse actions.

5. Continue to drag and drop functions to build the desired expression.
6. Edit the expression in the text field, as needed.
7. Optionally, do either or both of the following:
 - Click **Validate**. A message is displayed if the expression is validated successfully.
 - Click **Test** to test the expression. See [XQuery Transformation Testing Prerequisites and Guidelines](#).
8. Click **Save** to close the editor and insert the expression in the action.

Creating Namespaces to Use in Inline Expressions

The Namespace Definitions palette includes a list of default namespaces, but you can also define new ones.

Before you begin:

These instructions assume you are creating or editing an expression in the XQuery Expression Editor, XQuery Condition Editor, or XPath Expression Editor, as described in [Creating and Editing Inline XQuery and XPath Expressions](#).

To create and use a namespace in an inline expression:

1. Select **Namespace Definitions**. The Namespace Definitions palette includes a list of default namespaces, plus lists of variable namespaces and user defined namespaces, if any exist.
2. To define and add a user namespace,
 - a. Click **Add Namespace**.
 - b. In the **Prefix** field, enter a unique identifier for the namespace. You cannot use the same prefix more than once.
 - c. In the **URI** field, enter a URL for this namespace in the format `http://url/.../` or enter a URN in the format `uddi:server:`.
 - d. Click **Add** to add the namespace to the User Defined Namespaces list.
 - e. Copy and paste the user-defined namespace into the XQuery expression, XQuery condition, or XPath, as described in [Creating and Editing Inline XQuery and XPath Expressions](#).

After you finish:

Continue as described in [Creating and Editing Inline XQuery and XPath Expressions](#).

Creating Variable Structures in the XQuery Editors

The Variable Structures palette in the XQuery and XPath editors displays graphical representations of the contents of variables. It includes by default the built-in message context variables `attachments`, `body`, `header`, `outbound`, and `inbound`.

Each variable structure mapping entry has a label and maps a variable or variable path to one or more structures. The scope of these mappings is a stage or a route node.

You can also declare your own variable structures, based on:

- XML types, including:
 - Schema elements
 - WSDL elements
 - Schema types
 - WSDL types
 - MFLs
- Service interfaces
- Simple types (string or any XML)

You can use this feature directly for all user-defined variables, as well as `$inbound`, `$outbound`, and `$fault`. However, you cannot use it directly to access XML attachments in `$attachments`, headers in `$header`, or documents and RPC parameters in `$body`, with one exception— you can use it directly to access documents and parameters in `$body` for request messages received by a WSDL proxy service.

Before you begin:

These instructions assume you are creating or editing an expression in the XQuery Expression Editor, XQuery Condition Editor, or XPath Expression Editor, as described in [Creating and Editing Inline XQuery and XPath Expressions](#).

To create a variable structure:

1. Select **Variable Structures**.
2. In the Variable Structures palette, click **Add New Structure**.
3. Continue with any of the tasks listed in [Table 15-3](#).

Table 15-3 Create a New Variable Structure

To...	Complete these steps...
Create a variable structure that maps a variable to an XML Schema type	<ol style="list-style-type: none"> 1. Select XML Type at the top of the Variable Structures palette, if it is not already selected. 2. In the Structure Label field, enter a display name for the variable you want to create. This display name enables you to give a meaningful name to the structure so you can recognize it at design time but it has no impact at runtime. 3. In the Structure Path field, enter the path of the variable structure at runtime. The path must begin with \$. 4. Under the Type field, select the appropriate radio button, then select Schema Type. 5. Click Browse. The XML Schema Browser is displayed. Select an XML Schema from the list, select an XML Schema type from the Definitions pane, then click Submit. 6. Click Add to create the variable structure.

Table 15-3 (Cont.) Create a New Variable Structure

To...	Complete these steps...
<p>Create a variable structure that maps a variable to a WSDL type</p>	<ol style="list-style-type: none"> 1. Select XML Type at the top of the Variable Structures palette, if it is not already selected. 2. In the Structure Label field, enter a display name for the variable you want to create. This display name enables you to give a meaningful name to the structure so you can recognize it at design time but it has no impact at runtime. 3. In the Structure Path field, enter the path of the variable structure at runtime. 4. Under the Type field, select the appropriate radio button, then select WSDL Type. 5. Click Browse. The WSDL Browser is displayed. Select a WSDL file from the list of WSDL files, select a WSDL type from the Definitions pane, then click Submit. 6. Click Add to create the variable structure.
<p>Create a variable structure that maps a variable to an XML Schema element</p>	<ol style="list-style-type: none"> 1. At the top of the Variable Structures palette, select XML Type, if it is not already selected. 2. In the Structure Label field, enter a display name for the variable you want to create. This display name enables you to give a meaningful name to the structure so you can recognize it at design time but it has no impact at runtime. 3. In the Structure Path field, enter the path of the variable structure at runtime. 4. Under the Type field, select the appropriate radio button, then select Schema Element. 5. Click Browse. The XML Schema Browser is displayed. Select an XML Schema from the list, select an XML Schema type from the Definitions pane, then click Submit. 6. Click Add to create the variable structure.

Table 15-3 (Cont.) Create a New Variable Structure

To...	Complete these steps...
Create a variable structure that maps a variable to a WSDL element	<ol style="list-style-type: none"> 1. At the top of the Variable Structures palette, select XML Type, if it is not already selected. 2. In the Structure Label field, enter a display name for the variable you want to create. This display name enables you to give a meaningful name to the structure so you can recognize it at design time but it has no impact at runtime. 3. In the Structure Path field, enter the path of the variable structure at runtime. 4. Under the Type field, select the appropriate radio button, then select WSDL Element. 5. Click Browse. The WSDL Browser is displayed. Select a WSDL file from the list of WSDL files, select a WSDL element from the Definitions pane, then click Submit. 6. Click Add to create the variable structure.
Create a variable structure that maps a variable to a child element	<ol style="list-style-type: none"> 1. At the top of the Variable Structures palette, select XML Type, if it is not already selected. 2. In the Structure Label field, enter a display name for the variable you want to create. This display name enables you to give a meaningful name to the structure so you can recognize it at design time but it has no impact at runtime. 3. In the Structure Path field, enter the path of the variable structure at runtime. 4. Under the Type field, select the type of variable you want to create: To create an XML Schema element or WSDL element variable, select the radio button associated with this option, then select Schema Element or WSDL Element. To create an MFL variable, select the radio button associated with this option, then select MFL. 5. For the XML Schema, WSDL file, or MFL file, click Browse to select an object from the list that the browser displays, then click Submit. For example, select an MFL from a list of MFLs, then click Submit. 6. Select the Set as child check box to set the element as a child of the structure being created. 7. Click Add to create the variable structure.

Table 15-3 (Cont.) Create a New Variable Structure

To...	Complete these steps...
<p>Create a variable structure that uses an MFL resource</p>	<ol style="list-style-type: none"> 1. At the top of the Variable Structures palette, select XML Type, if it is not already selected. 2. In the Structure Label field, enter a display name for the variable you want to create. This display name enables you to give a meaningful name to the structure so you can recognize it at design time but it has no impact at runtime. 3. In the Structure Path field, enter the path of the variable structure at runtime. 4. Under the Type field, select the appropriate radio button, then click Browse. The MFL Browser is displayed. 5. Select an MFL from the list of MFLs, then click Submit. 6. Click Add to create the variable structure.
<p>Create a Service Interface variable structure</p>	<ol style="list-style-type: none"> 1. At the top of the Variable Structures palette, select Service Interface. 2. In the Structure Label field, enter a display name for the variable you want to create. This display name enables you to give a meaningful name to the structure so you can recognize it at design time but it has no impact at runtime. 3. In the Structure Path field, the default is already set as \$body. You cannot change this field. 4. In the WSDL Based Service field, select the Service Browser icon, select a service from the list of services the Service Browser displays, then click Submit. <p>The service you selected is displayed in the WSDL Based Service field.</p> <ol style="list-style-type: none"> 1. In the Operation field, select an operation or select None to not include an operation. 2. Click Add to create the variable.

Table 15-3 (Cont.) Create a New Variable Structure

To...	Complete these steps...
Create a Simple variable structure	<ol style="list-style-type: none"> 1. At the top of the Variable Structures palette, select Simple Type. 2. In the Structure Label field, enter a display name for the variable you want to create. This display name enables you to give a meaningful name to the structure so you can recognize it at design time but it has no impact at runtime. 3. In the Structure Name field, enter a name for the variable structure you want to create. 4. Under the Type field, select String or Any XML. 5. Click Add to create the variable.

After you finish:

Continue as described in [Creating and Editing Inline XQuery and XPath Expressions](#).

Creating Custom XPath Functions in the XQuery Editors

Creating custom XQuery functions is a development process that requires writing Java code. For more information, see [Creating Custom XPath Functions](#).

Binding External XQuery Resources to Inline XQueries

You can bind XQuery resources to inline XQuery expressions, so they will be executed inline as part of an action.

Before you begin:

These instructions assume you are creating or editing an expression in the XQuery/XSLT Expression Editor, XQuery Condition Editor, or XPath Expression Editor, as described in [Creating and Editing Inline XQuery and XPath Expressions](#).

To bind an XQuery Resource to an inline expression:

1. Click **Variable Structures**.
2. In the workspace (under the button bar), select **XQuery Resources**.
3. In the **1. Select an XQuery resource to execute** box, click **Browse**.
4. In the XQuery Browser, select the radio button associated with the XQuery you want to use, then click **Submit**.
5. In the **2. Bind Variables** box, define the input parameters for the transformation. For each variable listed under **Variable Name** enter an XQuery expression to be mapped to it. You must define a mapping for each parameter. For example, if an XQuery transformation has two input parameters named **one** and **two**, the **Variable Name** field has two labels—**one** and **two**. A text box, into which the XQuery expression is entered, is associated with each label.

The following XQuery expressions are examples of valid input to this field:

```
$body/*[1]  
$body/po:PurchaseOrder
```

Note:

The following variable name is not a valid entry for this field and results in an exception: `body`.

6. After you finish, continue with any of the following tasks.
 - Click **Validate**. A message is displayed if the expression is validated successfully.
 - Click **Test**. See [XQuery Transformation Testing Prerequisites and Guidelines](#).
 - Save or discard your changes.

Binding External XSLT Resources to Inline XQueries

The XQuery/XSLT Expression Editor page allows you to select an XSLT resource for execution. To learn more about this editor, see [Creating and Editing Inline XQuery and XPath Expressions](#).

To Select an XSLT Resource for Execution

1. Select the **XSLT Resources** option.
2. Under the **Select the XSLT resource to execute** field, select the **XSLT Browser** icon.
3. In the XSLT Browser, select the radio button associated with the XSLT you want to execute, then click **Submit**.
4. Under the **Bind Variables** field, a label and a corresponding text box is displayed for each input parameter of the transformation. Each label corresponds to the name of a parameter, and each text box is for defining an XQuery expression to be mapped to the parameter. You must define a mapping for each parameter. For example, if an XSL transformation has two input parameters named **one** and **two**, the **Variable Mapping** field has two labels—**one** and **two**—with a text box associated with each into which the XQuery expression is entered. In addition to the mapping for any input variables, you must also specify an XQuery expression for the Input Document to the transformation. The mapping is specified in the text box with the label Input Document.

The following XQuery expressions are examples of valid input to this field:

```
$body/*[1]  
$body/po:PurchaseOrder
```

Note:

The following variable name is not a valid entry for this field and results in an exception: `body`.

5. Continue with any of the following tasks:

- [Creating Namespaces to Use in Inline Expressions](#)
- [Creating Variable Structures in the XQuery Editors](#)
- [Building Expressions in the Editor Workspace Text Fields](#)
- [Binding External XQuery Resources to Inline XQueries](#)
- Click **Validate**. A message is displayed if the expression is validated successfully.
- Click **Test**. See [XQuery Transformation Testing Prerequisites and Guidelines](#).

Binding Dynamic XQuery Expressions to Inline XQueries

The XQuery/XSLT Expression Editor page allows you to specify a dynamic XQuery expression that evaluates at runtime to the name of a pre-registered XQuery resource. To learn more about this editor, see [Creating and Editing Inline XQuery and XPath Expressions](#).

To define a dynamic XQuery expression

1. Select the **Dynamic XQuery** option.
2. In the Enter Expression for XQuery Resource area, enter the XQuery expression that will evaluate at runtime to the name of a pre-registered XQuery resource.

The following shows the syntax for the XQuery resource (representing the full name of the resource):

```
Project/folder1/folder2/XQueryResourceName
```

3. In the Bind Variables using XQuery Template or Custom Variables area, define the input parameters for the transformation.
 - Click Browse in the **Select XQuery Template** field to select an existing registered resource to serve as a template for the shape of the query (the number and names of the variables). After selecting a template, the variables appear in the Bind Variables area. Note that the template is not persisted with the configuration. Instead, the template serves as a quick start to help you specify the variables for the query.
 - Type a variable name in the **Add Custom Variable** field, and click Add. For each variable listed under Variable Name, enter an XQuery expression to be mapped to it. You must define a mapping for each parameter. For example, if an XQuery transformation has two input parameters named **one** and **two**, the **Variable Name** field has two labels—**one** and **two**. A text box, into which the XQuery expression is entered, is associated with each label.

The following XQuery expressions are examples of valid input to the variable fields:

```
$body/*[1]
$body/po:PurchaseOrder
```

4. After you finish, continue with any of the following tasks.
 - Click **Validate**. A message is displayed if the expression is validated successfully.

- Click **Test**. See [XQuery Transformation Testing Prerequisites and Guidelines](#).
- Save or discard your changes.

Binding Dynamic XSLT Expressions to Inline XQueries

The XQuery/XSLT Expression Editor page allows you to specify a dynamic XPath/XQuery expression that evaluates at runtime to the name of a pre-registered XSLT resource. To learn more about this editor, see [Creating and Editing Inline XQuery and XPath Expressions](#).

To define a dynamic XSLT expression

1. Select the **Dynamic XSLT** option.
2. In the Enter Expression for XSLT Resource area, enter the XPath/XQuery expression that will evaluate at runtime to the name of a pre-registered XSLT resource. For example: `$body/*:name/text()`

The following shows the syntax for the XSLT resource (representing the full name of the resource):

```
Project/folder1/folder2/myXSLResourceName
```

3. In the Bind Input area, enter an inline XQuery/XPath expression to select the XML input document. For example: `$body/message`.
4. In the Bind Variables using XSLT Template or Custom Variables area, define the input parameters for the transformation.
 - Click Browse in the **Select XSLT Template** field to select an existing registered XSLT resource to serve as a template for the shape of the query (the number and names of the variables). After selecting a template, the variables appear in the Bind Variables area. Note that the template is not persisted with the configuration. Instead, the template serves as a quick start to help you specify the variables for the query.
 - To add a variable manually, type a variable name in the **Add Custom Variable** field, and click **Add**.
 - For each variable listed under Variable Name, enter an XQuery/XPath expression to be mapped to it. You must define a mapping for each parameter. For example, if an XQuery transformation has two input parameters named **one** and **two**, the **Variable Name** field has two labels—**one** and **two**. A text box, into which the XQuery expression is entered, is associated with each label.

The following XQuery expressions are examples of valid input to the variable fields:

```
$body/*[1]  
$body/po:PurchaseOrder
```

5. After you finish, continue with any of the following tasks.
 - Click **Validate**. A message is displayed if the expression is validated successfully.
 - Click **Test**. See [XQuery Transformation Testing Prerequisites and Guidelines](#).
 - Save or discard your changes.

Entering XQuery Comparison Expressions Using the Builder Option

Before you begin:

These instructions assume you are creating or editing an XQuery conditional expression in the XQuery Condition Editor, as described in [Creating and Editing Inline XQuery and XPath Expressions](#).

To enter an XQuery comparison expression:

1. In the XQuery Condition Editor, select **Builder** (located under the workspace button bar), if it is not already selected. option.
2. In the Expression Builder box, select **Comparison Expression** if it is not already selected.
3. In the **Operand** field, enter a context variable, namespace definition, or XQuery function.

To build the operand, you can paste XQuery functions from the XQuery Functions palette and namespaces from the Namespace Definitions palette. See [Building Expressions in the Editor Workspace Text Fields](#).

4. From the **Operator** list, select a comparison operator.
5. In the **Value** field, enter text or enter a context variable.
You must enter text in quotations—for example, "true" is valid; true is not.
6. Click **Add**. The text you entered is displayed in the Expressions pane.
7. Repeat steps 3-6 to build additional conditions. Each condition is added to the end of the list of conditions.

Consider the following when using multiple conditions:

- When you build additional expressions, make sure to select the **And** or the **Or** options in the **Conjunction** field.
 - You can select a condition and click the **Up** arrow to move it up in the list of conditions or click the **Down** arrow to move it down the list of conditions. You can also click the **Edit** icon to update a condition, or click the **Delete** icon to delete it.
 - Unary expressions may be intermixed with Comparison expressions in the overall definition of a condition.
8. Optionally, do either or both of the following:
 - Click **Validate**. A message is displayed if the expression is validated successfully.
 - Click **Test** to test the expression. See [XQuery Transformation Testing Prerequisites and Guidelines](#).
 9. Click **Save** to close the editor and insert the expression in the action.

After you finish:

Continue configuring the action, as described in [Adding and Editing Pipeline Actions in the Console](#).

Entering Unary Expressions Using the Builder Option

Before you begin:

These instructions assume you are creating or editing an XQuery conditional expression in the XQuery Condition Editor, as described in [Creating and Editing Inline XQuery and XPath Expressions](#).

To enter an XQuery comparison expression:

1. In the XQuery Condition Editor, select **Builder** (located under the workspace button bar), if it is not already selected. option.
2. In the Expression Builder box, select **Unary Expression** if it is not already selected.
3. Select the **Not** check box to make this a negative expression, or leave it blank.
4. Enter a context variable, namespace definition or XQuery function in the **Expression** field.

To build the expression, you can paste XQuery functions from the XQuery Functions palette and namespaces from the Namespace Definitions palette. See [Building Expressions in the Editor Workspace Text Fields](#).

5. Click **Add** to add the text to the Expressions pane.
6. Repeat steps 3-5 to build additional conditions. Each condition is added to the end of the list of conditions.

Consider the following when building unary expressions.

- When you build additional expressions, make sure to select the **And** or the **Or** options in the **Conjunction** field.
 - You can select a condition and click the **Up** arrow to move it up in the list of conditions, click the **Down** arrow to move it down the list of conditions, click the **Edit** icon to update it, or click the **Delete** icon to delete it.
 - Unary expressions may be intermixed with Comparison expressions in the overall definition of a condition.
7. Optionally, do either or both of the following:
 - Click **Validate**. A message is displayed if the expression is validated successfully.
 - Click **Test** to test the expression. See [XQuery Transformation Testing Prerequisites and Guidelines](#).
 8. Click **Save** to close the editor and insert the expression in the action.

Working with Pipelines in Oracle JDeveloper

This chapter describes how to create and configure pipelines, or message flows, using Oracle JDeveloper. Sections include adding and configuring pipeline pairs, conditional branches, stages, operational branches, route nodes, and error handlers.

For more detailed information on message flows, see [Modeling Message Flow in Oracle Service Bus](#).

This chapter includes the following sections:

- [Adding a Pipeline Component in JDeveloper](#)
- [Viewing and Editing Pipelines in JDeveloper](#)
- [Adding Shared Variables to Pipelines in JDeveloper](#)
- [Adding Pipeline Pair Nodes to Pipelines in JDeveloper](#)
- [Adding Conditional Branches to Pipelines in JDeveloper](#)
- [Adding Operational Branches to Pipelines in JDeveloper](#)
- [Adding Stages to Pipelines in JDeveloper](#)
- [Adding Route Nodes to Pipelines in JDeveloper](#)
- [Cutting, Copying, and Pasting Stages and Route Nodes in JDeveloper](#)
- [Configuring the Resequencer in JDeveloper](#)

Adding a Pipeline Component in JDeveloper

Add a pipeline to define the message flow and any data transformation or validation for the project. You can also define errors and reporting in a pipeline. When you create a new pipeline, you have the option to generate a proxy service from the pipeline configuration.

How to Add a Pipeline in JDeveloper

You can use the Service Bus Overview Editor in Oracle JDeveloper to add a pipeline component to the service bus project.

To add a pipeline:

1. Make sure you have the Service Bus project open in Oracle JDeveloper.

2. Double-click the project icon in the Application Navigator to open the Service Bus Overview Editor. For more information about the Service Bus Overview Editor, see [Introduction to the Service Bus Overview Editor](#).

Note:

You can also right-click the project folder in the Application Navigator and select **New > Pipeline** from the context menu that appears. Next, continue from Step 5 to configure the pipeline.

3. From the Components window, select **Service Bus**.
4. From the **Resources** list, drag a Pipeline into the Pipelines/Split Joins lane in the designer.

The Create Pipeline Service wizard appears.

5. Configure the settings for the pipeline.

For help with the configuration fields, click **Help** or press **F1**.

6. To generate a proxy service to associate with the pipeline, click **Expose as a Proxy Service** on the Type page of the wizard. Select a transport for the proxy service and, optionally, modify the name.
7. On the last page of the wizard, click **Finish**.

The pipeline file is added to the project, and the pipeline appears in the Pipelines/Split Joins section of the designer. If you exposed the pipeline as a proxy service, the proxy service also appears in the Proxy Services swim lane, and the components are automatically wired.

8. To define the message flow in the pipeline, see [Viewing and Editing Pipelines in JDeveloper](#)
9. Click **Save All** in the JDeveloper toolbar.

Viewing and Editing Pipelines in JDeveloper

The message flow corresponding to a proxy service is handled by a pipeline. This section describes how to view and pipelines using the Pipeline Editor.

How to View and Edit a Pipeline in JDeveloper

Use the Pipeline Editor in Oracle JDeveloper to view and edit the message flow for the pipeline.

To view and edit a pipeline:

1. Make sure you have the Service Bus project open in Oracle JDeveloper.
2. Use one of the following methods to edit the message flow for a pipeline:
 - In Application Navigator, locate the pipeline node. Right-click the pipeline node and select **Open**. You can alternatively double-click the pipeline node to open it.
 - In Application Navigator, click the project node (or overview.xml) to open the Overview Editor.

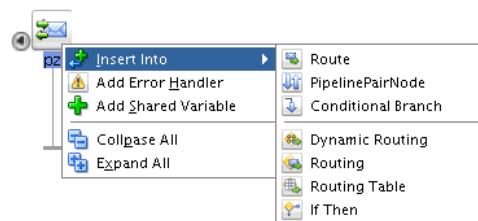
In the Overview Editor, double-click the pipeline component to open the Pipeline Editor.

The Pipeline Editor appears. Ensure that the **Design** tab is selected at the bottom left corner of the editor.

If no message flow has yet been created for the selected pipeline, the Pipeline Editor Design view shows a single icon on the page, the pipeline icon. This is the starting node for the message flow.

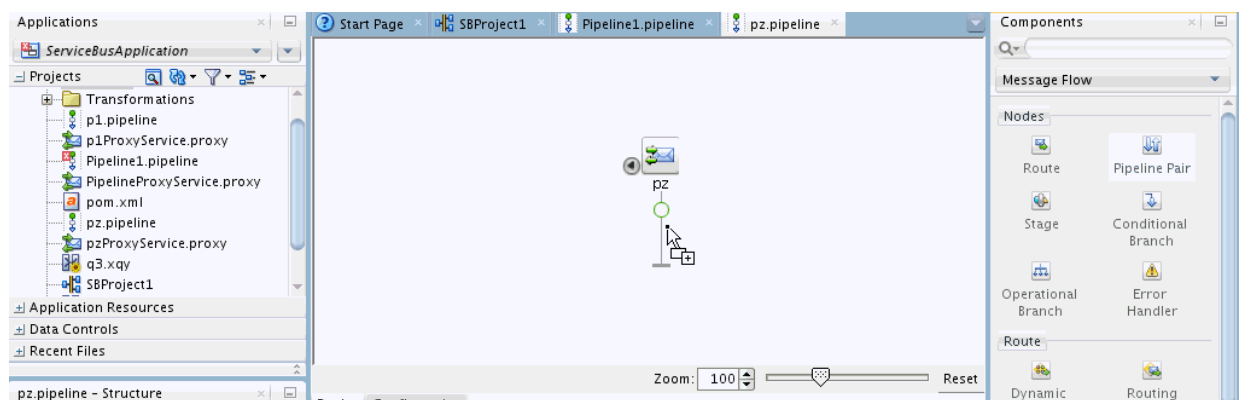
- Use one of the following methods to add a pipeline component (node):
 - Right-click the start node to get options for pipeline components that you can add. [Figure 16-1](#) shows the options available for the start node. You can add nodes like the Route node, PipelinePair node, and Conditional Branch.

Figure 16-1 Right-Clicking a Node to Add a Pipeline Component



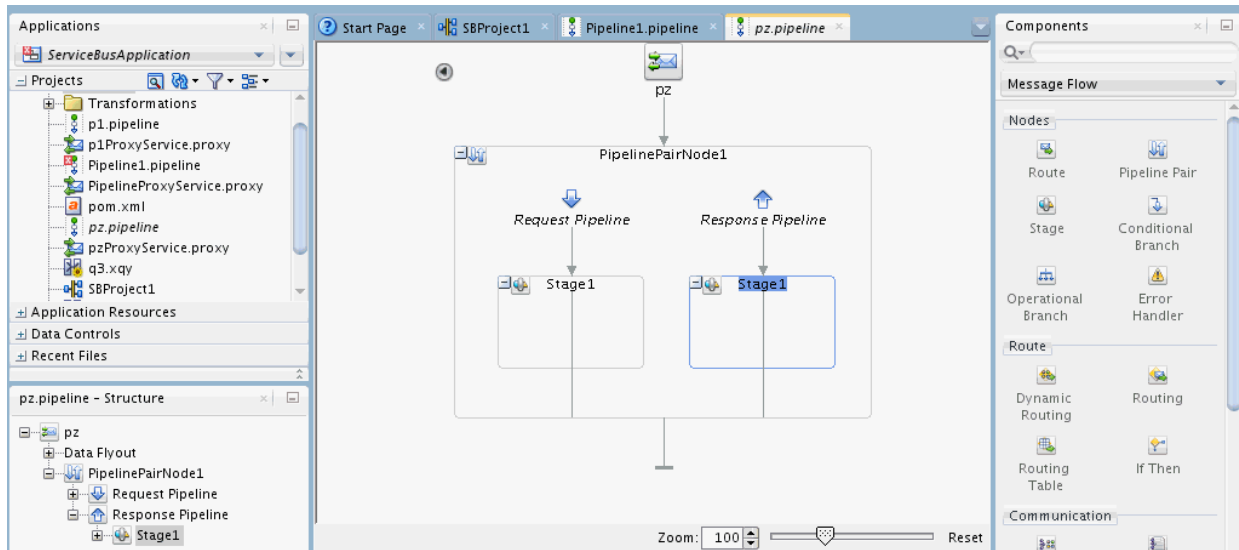
- Alternatively, select the pipeline component to add from the Components window, and drag the component to the Pipeline Editor window. Yellow circles appear indicating valid places to drop the component in the pipeline. Drag the component to a yellow circle. The yellow circle turns green. Release the component to add it. [Figure 16-2](#) shows a Pipeline Pair node being added to a start node.

Figure 16-2 Adding a Pipeline Pair to the Start Node



When you add components to the editor, icons are displayed on the editor to represent the components. The relationships among the components are shown with lines and bounding boxes.

[Figure 16-3](#) shows a pipeline where a Pipeline Pair node has been added to the flow. The Pipeline Pair comprises of a Request Pipeline and a Response Pipeline. The Request and Response pipelines have stages that can contain action nodes.

Figure 16-3 Pipeline with a Pipeline Pair Node

4. Continue to build the pipeline by adding more components to the editor. For example, to add a communication action to the stage node, you can drag the communication action from the Components window to the stage node in the editor. Alternatively, you can right-click the Stage node to get options for pipeline components that you can add to the Stage node. The options available for each component may differ, depending on context.
5. Click **Save** in the Oracle JDeveloper toolbar.

Adding Shared Variables to Pipelines in JDeveloper

If two pipelines in a single call chain declare the same shared variable, then they read and modify the same variable. In other words, if pipeline P1 declares a shared variable `var`, and pipeline P2 also declares a shared variable `var`, then any changes to `var` in P1 are visible in P2, and vice versa. A shared variable must be of the String, Boolean, or XML data type.

When a proxy service receives and processes a message, all invoked pipelines that use a shared variable, read and write the same value for the variable. A subsequent message received by the proxy creates a new instance of the shared variable in the invoked pipelines.

Shared variables work across local proxy invocations and split-join component invocations. For example, say pipelines P1 and P2 declare a shared variable. Now, if P1 invokes a local proxy service or split-join component, which in turn invokes P2, then P1 and P2 continue to share the shared variable.

The following restrictions apply to using shared variables:

- System variables (such as `$body`, `$attachments`, `$operation`, `$inbound`, `$outbound`) cannot be shared.
- Variables cannot be shared across non-local proxy invocations. For example, say a pipeline invokes an HTTP proxy service, the shared variable is not propagated across this call.
- Variables cannot be shared between pipeline and split-join resources.

- Variables with Java and binary content types are not supported. For example, an XML-typed variable that has `<ctx: java-content />` in its XML structure, is not supported as a shared variable.

How to Add a Shared Variable to a Pipeline in JDeveloper

Use the Pipeline Editor to add a shared variable to the message flow for the pipeline.

To add a shared variable to a pipeline

1. In the Pipeline Editor, right-click the start node.
2. Select **Add Shared Variable** from the context menu that appears.
3. Enter a name for the shared variable in the **Enter Shared Variable Name** field.
4. Click **OK** to add the shared variable to the pipeline.

Adding Pipeline Pair Nodes to Pipelines in JDeveloper

Pipelines can include zero or more pipeline pair nodes: request and response pipelines for the proxy service (or for the operations on the service), and error handler pipelines that can be defined for stages, pipelines, and proxy services. Pipelines can include one or more stages, which in turn include actions.

How to Add a Pipeline Pair Node to a Pipeline in JDeveloper

Use the Pipeline Editor to add a pipeline pair to the message flow for the pipeline.

To add a pipeline pair to a pipeline:

1. In the Pipeline Editor, right-click the start node icon.
2. Select **Insert Into > PipelinePairNode** from the context menu that appears. The pipeline pair node is inserted.

Note:

You can alternatively choose to drag a **Pipeline Pair** component from the Components window to the appropriate location in the Pipeline Editor. The Pipeline Pair component can be found under the Nodes section of the Message Flow category.

3. To change the default name and add a description for the pipeline pair node, do the following:
 - a. Click the **Pipeline Pair Node** to select it. The Properties window displays the properties for the selected component.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. In the Properties window, change the name and description, as desired.

- c. Click **Save** in the Oracle JDeveloper toolbar.
4. To add stages to the pipeline, see [Adding Stages to Pipelines in JDeveloper](#).
5. To add actions to stages in the pipeline. See [Adding and Editing Actions in Pipelines in JDeveloper](#).
6. On the Pipeline Editor, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in JDeveloper](#).
7. Click **Save** in the Oracle JDeveloper toolbar.

Adding Conditional Branches to Pipelines in JDeveloper

A branch node allows processing to proceed along exactly one of several possible paths. Branching is driven by an XPath-based switch table. Each branch in the table specifies a condition (for example, <500) that is evaluated in order down the pipeline against a single XPath expression (for example, `./ns:PurchaseOrder/ns:totalCost on $body`). Whichever condition is satisfied first determines which branch is followed. If no branch condition is satisfied, then the default branch is followed. A branch node may have several descendants in the pipeline: one for each branch, including the default branch.

If the proxy service is not based on a WSDL file and receives multiple document types as input, you can consider using a conditional branch node.

Conditional branching is driven by a lookup table with each branch tagged with a simple, but unique, string value. A variable in the message context is designated as the lookup variable for that node, and at runtime, its value is used to determine which branch to follow. If no branch matches the value of the lookup variable, the default branch is followed. You should design the proxy service in such a way that the value of the lookup variable is set before reaching the branch node.

How to Add a Conditional Branch to a Pipeline in JDeveloper

Use the Pipeline Editor to add a conditional branch to the message flow for the pipeline.

To add a conditional branch to a pipeline:

1. In the Pipeline Editor, right-click the start node icon) or a **Branch Node** icon.
2. Select **Insert Into > Conditional Branch** from the context menu that appears. The conditional branch node is added, and any existing nodes after the inserted branch node are moved to the default branch of the new conditional branch node.

Note:

You can alternatively choose to drag a **Conditional Branch** component from the Components window to the appropriate location in the Pipeline Editor. The Conditional Branch component can be found under the Nodes section of the Message Flow category.

3. To change the properties for the conditional branch node, do the following:
 - a. Click the **Conditional Branch** node to select it. The Properties window displays the properties for the selected component.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Under the **General** section, change the name and description, as desired.
 - c. Under the **Condition** section, specify an XPath expression to use as the condition. You can invoke the XPath Expression Builder by clicking the **fx** icon.
 - d. Click **Save** in the Oracle JDeveloper toolbar.
4. To change the properties for a branch, do the following:
- a. Click the branch node to select it. The Properties window displays the properties for the selected component.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Under **Name**, specify a name for the branch.
 - c. Under **Value**, specify an operator and an XPath expression to use as the value. You can invoke the XPath Expression Builder by clicking the **fx** icon.
 - d. Click **Save** in the Oracle JDeveloper toolbar.
5. Optionally:
- Click the **Add New Branch** icon to add a new branch node.
 - Right-click a branch, and select **Delete** to delete the branch.
6. Click **Save** in the Oracle JDeveloper toolbar.
 7. On the Pipeline Editor, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in JDeveloper](#).
 8. Click **Save** in the Oracle JDeveloper toolbar.

Adding Operational Branches to Pipelines in JDeveloper

When pipelines define Web Services Description Language (WSDL)-based proxy services, operation-specific processing is required. Instead of configuring a branching node based on operations manually, Service Bus provides a minimal configuration branching node that automatically branches based on operations. In other words, when you create an operational branch node in a pipeline, you can quickly build your branching logic based on the operations defined in the WSDL file because the Oracle Service Bus Console presents those operations in the branch node configuration page.

A branch node allows processing to proceed along exactly one of several possible paths. Branching is driven by an XPath-based switch table. Each branch in the table specifies a condition (for example, `<500`) that is evaluated in order down the pipeline against a single XPath expression (for example, `./ns: PurchaseOrder/`

`ns:totalCost` on `$body`). Whichever condition is satisfied first determines which branch is followed. If no branch condition is satisfied, then the default branch is followed. A branch node may have several descendants in the pipeline: one for each branch, including the default branch.

How to Add an Operational Branch to a Pipeline in JDeveloper

Use the Pipeline Editor to add an operational branch to the message flow for the pipeline.

To add an operational branch to a pipeline:

1. In the Pipeline Editor, right-click the start node or a **Branch Node** icon.
2. Select **Insert Into > Operational Branch** from the context menu that appears. The operational branch node is added, and any existing nodes after the inserted branch node are moved to the default branch of the new operational branch node.

Note:

You can alternatively choose to drag an **Operational Branch** component from the Components window to the appropriate location in the Pipeline Editor. The Operational Branch component can be found under the Nodes section of the Message Flow category.

3. To change the properties for the operational branch node, do the following:
 - a. Click the operational branch node to select it. The Properties window displays the properties for the selected component.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Change the name and description, as desired.
 - c. Click **Save** in the Oracle JDeveloper toolbar.
4. To change the properties for a branch, do the following:
 - a. Click the branch node to select it. The Properties window displays the properties for the selected component.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Select the **Operation** represented by the branch.
 - c. Click **Save** in the Oracle JDeveloper toolbar.
5. Optionally:

- Click the **Add New Branch** icon to add a new branch node.
 - Right-click a branch, and select **Delete** to delete the branch.
6. On the Pipeline Editor, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in JDeveloper](#).
 7. Click **Save** in the Oracle JDeveloper toolbar.

Adding Stages to Pipelines in JDeveloper

Request pipelines, response pipelines, and error handlers can contain stages, where you configure actions to manipulate messages passing through the pipeline.

How to Add a Stage to a Pipeline in JDeveloper

Use the Pipeline Editor to add a stage to the message flow for the pipeline.

To add a stage to a pipeline:

1. In the Pipeline Editor, right-click a Request Pipeline icon or Response Pipeline icon in a pipeline pair node.
2. Select **Insert Into > Stage** from the context menu that appears. A stage node is added.

Note:

You can alternatively choose to drag a **Stage** component from the Components window to the appropriate location in the Pipeline Editor. The Stage component can be found under the Nodes section of the Message Flow category.

3. To change the default name and add a description for the stage, do the following:
 - a. Click the stage node to select it. The Properties window displays the properties for the selected component.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Edit the **Name** and **Description** fields for the selected stage node.
 - c. Click **Save** in the Oracle JDeveloper toolbar.
4. To add actions to the stage, right-click the **Stage** icon, then click **Insert Into > ActionName** from the context menu that appears. See [Adding and Editing Actions in Pipelines in JDeveloper](#).
5. To add error handling to the stage, right-click the **Stage** icon, then click **Add Stage Error Handler** from the context menu that appears. See [Adding Error Handlers in JDeveloper](#).

6. Continue to construct the pipeline, as described in [Viewing and Editing Pipelines in JDeveloper](#).
7. Click **Save** in the Oracle JDeveloper toolbar.

Adding Route Nodes to Pipelines in JDeveloper

A route node performs request/response communication with another service. It represents the boundary between request and response processing for the proxy service. When the route node dispatches a request message, the request processing is considered complete. When the route node receives a response message, the response processing begins. The route node supports conditional routing as well as request and response transformations.

Because a route node represents the boundary between request and response processing, it cannot have any descendants in the pipeline.

How to Add a Route Node to a Pipeline in JDeveloper

Use the Pipeline Editor to add a route node to the message flow for the pipeline.

To add a route node to a pipeline:

1. In the Pipeline Editor, right-click the start node or a **Branch Node** icon.
2. Select **Insert Into > Route** from the context menu. A route node is added.

Note:

You can alternatively choose to drag a **Route** component from the Components window to the appropriate location in the Pipeline Editor. The Route component can be found under the Nodes section of the Message Flow category.

3. To add actions to the route node, right-click the **Route Node** icon, then select the appropriate option from the context menu. See the following sections for information about the actions you can add to route nodes:
 - [Adding If Then Actions in JDeveloper](#)
 - [Adding Dynamic Routing to Route Nodes in JDeveloper](#)
 - [Adding Routing Actions to Route Nodes in JDeveloper](#)
 - [Adding Routing Tables to Route Nodes in JDeveloper](#)
 - [Adding Error Handlers in JDeveloper](#)
4. In the Pipeline Editor, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in JDeveloper](#).
5. Click **Save** in the Oracle JDeveloper toolbar.

Cutting, Copying, and Pasting Stages and Route Nodes in JDeveloper

You can cut, copy, and paste stages and route nodes in the Pipeline Editor.

- To cut a stage or a route node, right-click its icon and select **Cut** or **Copy**.
- To paste a stage that you cut or copied from a different pipeline pair within the message flow of the pipeline or from the message flow of a different pipeline, do one of the following:
 - Right-click the target **Request Pipeline** or **Response Pipeline** icon, then click **Paste**.
 - Right-click an existing **Stage** icon in a pipeline, then click **Paste**.
- To paste a route node that you cut or copied from the message flow of another pipeline, click the target **Branch Node** icon, then click **Paste Route**.

Configuring the Resequencer in JDeveloper

The resequencer in Service Bus rearranges a stream of related but out-of-sequence messages into a sequential order. When incoming messages arrive, they may be in a random order. The resequencer orders the messages based on sequential or chronological information, and then sends the messages to the target services in an orderly manner. The sequencing is performed based on the sequencing strategy selected.

You can configure the resequencer inside a pipeline component. Pipelines with the following service types are supported:

- **WSDL:** Resequencing is available for operations with only request type.
- **Message Type:** The request message type should be XML, and the response message type should be None.

Note:

The resequencer does not support **Any XML** and **Any SOAP** service types. For WSDL-based services, the WSDL file must be one-way; that is, it cannot contain output elements. For information about using generated WSDL files with resequencing pipelines, see [How to Generate a WSDL File from a Service in JDeveloper](#).

How to Configure Resequencing in a Pipeline in JDeveloper

This section describes how to configure the resequencer in a pipeline using JDeveloper.

To enable resequencing in a pipeline:

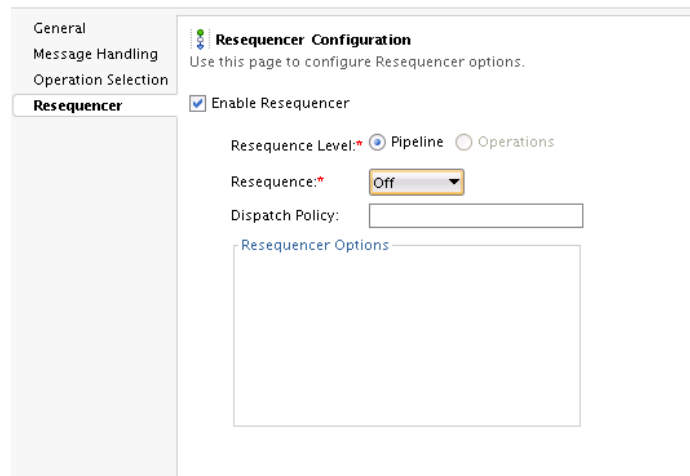
1. Make sure you have the Service Bus project open in JDeveloper.
2. Use one of the following methods to edit the pipeline configuration:
 - In Application Navigator, locate the pipeline node. Right-click the pipeline node and select **Open**. You can alternatively double-click the pipeline node to open it.
 - In Application Navigator, click the project node (or overview.xml) to open the Overview Editor.

In the Overview Editor, double-click the pipeline component to open the Pipeline Editor.

The Pipeline Editor appears. Select the **Configuration** tab from the bottom left corner of the editor.

3. Select **Resequencer** from the left pane in the editor. The Resequencer Configuration page appears. [Figure 16-4](#) shows the Resequencer Configuration page.

Figure 16-4 Resequencer Configuration Page



4. Select **Enable Resequencer** to enable resequencing for the pipeline.
5. Select the **Resequence Level**. Choose **Pipeline** to configure resequencing at the component level. Choose **Operations** to configure resequencing at the operation level. See [Selecting the Resequence Level in JDeveloper](#) for more information on resequence levels.

If you select **Operations**, you get the option to configure resequencing for each operation separately.

6. Select the **Resequence** mode. If you are configuring resequencing at the **Operations** level, then you can select a **Resequence Mode** corresponding to each operation. See [How to Configure the Resequencing Mode in JDeveloper](#) for more information on the various resequencing options.

Depending on the **Resequence** mode you select, you get options corresponding to that mode. For example, selecting the **Standard** mode requires you to select values for **Group**, **Id**, and so on. [Figure 16-5](#) shows the resequencer configuration options that appear when you select the **Standard** mode.

Figure 16-5 Resequencer Options for the Standard Resequence Mode

7. Enter a **Dispatch Policy**, which specifies the Work Manager to use. The default Work Manager is used if no other Work Manager exists.
8. Click **Save** in the JDeveloper toolbar.

Selecting the Resequence Level in JDeveloper

You can define resequencing either at the pipeline level or the operation level. The Resequence Level can have the following values:

- **Pipeline:** A common configuration specified at the component level is used to resequence all messages. If a component has multiple operations, then messages for each operation are sequenced separately using the common component configuration.

Component-level resequencing is allowed only when all the operations of the pipeline component support request one-way messages. If only a subset of operations support request one-way messages, then you can individually specify operation-level resequencing for these operations.

- **Operations:** For a WSDL-based pipeline, resequencing can be configured at the operation level. Each operation can have a different resequencer configuration. Only operations supporting request one-way messages can be resequenced. Non-WSDL pipelines cannot have resequencer configured at the operation level.

How to Configure the Resequencing Mode in JDeveloper

This section provides instructions on how to configure various resequencing modes. See "Resequencing Order" in *Developing SOA Applications with Oracle SOA Suite* to learn about the various resequencing modes. By default, the group ID has a character limit of 1000; the ID has a character limit of 100.

Configuring a Standard Resequencer

To configure a standard resequencer:

1. In the Resequencer Configuration page, select **Standard** from the **Resequence** drop-down list. If you are configuring resequencing at the operation level, select **Standard** from the **Resequence Mode** drop-down list for the operation.

The Resequencer Options or Operation Details area is populated with fields related to standard resequencing. See [Figure 16-5](#) for more details.

2. Fill in the fields listed in [Table 16-1](#).

Table 16-1 Standard Resequencing Options

Field Name	Description	Default Value	Mandatory
Group	An XQuery expression that points to the field in the incoming message on which grouping is done. If you do not enter a value, then all messages are put in one default group. Click the Expression Builder icon on the right to invoke the XQuery Expression Builder.	N/A	N
ID	An XQuery expression that points to the field in the incoming message on which resequencing is done. Click the Expression Builder icon on the right to invoke the XQuery Expression Builder.	N/A	Y
Start	The starting number of the ID sequence.	1	N
Increment	The increment of the ID sequence.	1	N
Timeout	The time period in seconds to wait for an expected message. The resequencer locks the group as timed-out if a time out occurs. The default value of 0 means that the timeout never happens for a group by default.	0	N

Configuring a FIFO Resequencer

To configure a FIFO resequencer:

1. In the Resequencer Configuration page, select **FIFO** from the **Resequence** drop-down list. If you are configuring resequencing at the operation level, select **FIFO** from the **Resequence Mode** drop-down list for the operation.

The Resequencer Options or Operation Detail area is populated with fields related to FIFO resequencing.

2. In the **Group** field, enter an XQuery expression pointing to the field in the incoming message on which grouping is performed.

Click the Expression Builder icon on the right to invoke the XQuery Expression Builder

Configuring a Best Effort Resequencer

To configure a best effort resequencer:

1. In the Resequencer Configuration page, select **Best Effort** from the **Resequence** drop-down list. If you are configuring resequencing at the operation level, select **Best Effort** from the **Resequence Mode** drop-down list for the operation.

The Resequencer Options or Operation Detail area is populated with fields related to Best Effort resequencing.

2. Fill in the fields listed in [Table 16-2](#) to configure the best effort resequencer.

Table 16-2 Best Effort Resequencing Options

Field Name	Description	Default Value	Mandatory
Group	An XQuery expression that points to the field in the incoming message on which grouping is performed. If no value is entered here, then all messages are considered to be in one default group. Click the Expression Builder icon on the right to invoke the XQuery Expression Builder.	N/A	N
ID	An XQuery expression that points to the field in the incoming message that contains the ID on which resequencing is performed. Click the Expression Builder icon on the right to invoke the XQuery Expression Builder.	N/A	Y
Datatype	The data type of the sequence ID. The ordering process is based on the data type. Supported values are Date/Time and Numeric.	Numeric	Y
Max Rows	Number of in-sequence messages that the resequencer should pick from the data store at a time. This must be a positive integer value. You must specify Max Rows or Time Window (explained below), but not both.	5	N
Time Window (sec)	The length of time in seconds to wait after a message arrives before selecting messages from the data store for resequencing. The default value of 0 means no wait. You must specify a Time Window or Max Rows (described above), but not both.	0	N

Working with Pipeline Actions in Oracle JDeveloper

This section describes how to add different types of actions to message flows using Oracle JDeveloper.

Actions are the elements of pipeline stages, error handler stages, route nodes, and branch nodes that define how messages are to be defined as they flow through a pipeline. Use the Pipeline Editor, in Oracle JDeveloper, to add actions such as route, publish, service callout, transport headers, conditional actions, error actions, and message transformation actions.

This chapter includes the following sections:

- [Adding and Editing Actions in Pipelines in JDeveloper](#)
- [Adding Publish Actions in JDeveloper](#)
- [Adding Publish Table Actions in JDeveloper](#)
- [Adding Dynamic Publish Actions in JDeveloper](#)
- [Adding Routing Options Actions in JDeveloper](#)
- [Adding Service Callout Actions in JDeveloper](#)
- [Adding Transport Header Actions in JDeveloper](#)
- [Adding Dynamic Routing to Route Nodes in JDeveloper](#)
- [Adding Routing Actions to Route Nodes in JDeveloper](#)
- [Adding Routing Tables to Route Nodes in JDeveloper](#)
- [Adding For Each Actions in JDeveloper](#)
- [Adding If Then Actions in JDeveloper](#)
- [Adding Raise Error Actions in JDeveloper](#)
- [Adding Reply Actions in JDeveloper](#)
- [Adding Resume Actions in JDeveloper](#)
- [Adding Skip Actions in JDeveloper](#)
- [Adding Assign Actions in JDeveloper](#)
- [Adding Delete Actions in JDeveloper](#)
- [Adding Insert Actions in JDeveloper](#)

- [Adding Java Callout Actions in JDeveloper](#)
- [Adding MFL Translate Actions in JDeveloper](#)
- [Adding nXSD Translate Actions in JDeveloper](#)
- [Adding Rename Actions in JDeveloper](#)
- [Adding Replace Actions in JDeveloper](#)
- [Adding Validate Actions in JDeveloper](#)
- [Adding Alert Actions in JDeveloper](#)
- [Adding Log Actions in JDeveloper](#)
- [Adding Report Actions in JDeveloper](#)
- [Adding Error Handlers in JDeveloper](#)
- [Disabling an Action or a Stage in JDeveloper](#)

Adding and Editing Actions in Pipelines in JDeveloper

Actions are the elements of pipeline stages, error handler stages, route nodes, and branch nodes that define how messages are to be defined as they flow through a pipeline.

Before you begin

These instructions assume you are already editing a pipeline in the Pipeline Editor, as explained in [Viewing and Editing Pipelines in JDeveloper](#).

They also assume you have already added a pipeline stage, a route node, or an error handler stage. See:

- [Adding Pipeline Pair Nodes to Pipelines in JDeveloper](#)
- [Adding Stages to Pipelines in JDeveloper](#)
- [Adding Error Handlers in JDeveloper](#)

To add an action to a pipeline:

1. In the Pipeline Editor, right-click the component icon to which you wish to add the action. For example, right-click a Stage to add an action.
2. Select **Insert Into** > *Action Name* from the context menu that appears. The available action names depends on the context.

Note:

You can alternatively choose to drag an **Action** component from the Components window to the appropriate location in the Pipeline Editor.

As you drag a component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the component to one of the yellow circles, the circle changes to green indicating that you can drop the component there.

Table 17-1 through Table 17-4 list the actions you can configure for pipelines.

Table 17-1 Pipeline - Communication Actions

Action	Description	More Information
 Publish	Publish a message to a statically specified service.	Adding Publish Actions in JDeveloper
 Publish Table	Publish a message to zero or more statically specified services. Switch-style condition logic is used to determine at runtime which services will be used for the publish.	Adding Publish Table Actions in JDeveloper
 Dynamic Publish	Publish a message to a service identified by an XQuery expression	Adding Dynamic Publish Actions in JDeveloper
 Routing Options	Modify any or all of the following properties in the outbound request: URI, Quality of Service, Mode, Retry parameters, Message Priority.	Adding Routing Options Actions in JDeveloper
 Service Callout	Configure a synchronous (blocking) callout to a Service Bus-registered proxy or business service.	Adding Service Callout Actions in JDeveloper
 Transport Headers	Set the transport header values in messages	Adding Transport Header Actions in JDeveloper
 Dynamic Routing	Assign a route for a message based on routing information available in an XQuery resource.	Adding Dynamic Routing to Route Nodes in JDeveloper
 Routing	Identify a target service for the message and configure how the message is routed to that service:	Adding Routing Actions to Route Nodes in JDeveloper

Table 17-1 (Cont.) Pipeline - Communication Actions


Action	Description	More Information
 Routing Table	Assign a set of routes wrapped in a switch-style condition table. Different routes are selected based upon the results of a single XQuery expression.	Adding Routing Tables to Route Nodes in JDeveloper

Table 17-2 Pipeline - Flow Control Actions

Action	Description	More Information
 For each	Iterate over a sequence of values and execute a block of actions	Adding For Each Actions in JDeveloper
 If...then...	Perform an action or set of actions conditionally, based on the Boolean result of an XQuery expression.	Adding If Then Actions in JDeveloper
 Raise error	Raise an exception with a specified error code (a string) and description.	Adding Raise Error Actions in JDeveloper
 Reply	Specify that an immediate reply be sent to the invoker.	Adding Reply Actions in JDeveloper
 Resume	Resume message flow after an error is handled by an error handler.	Adding Resume Actions in JDeveloper
 Skip	Specify that at runtime, the execution of the current stage is skipped and the processing proceeds to the next stage in the message flow.	Adding Skip Actions in JDeveloper

Table 17-3 Pipeline - Message Processing Actions


Action	Description	More Information
 Assign	Assign the result of an XQuery expression to a context variable.	Adding Assign Actions in JDeveloper

Table 17-3 (Cont.) Pipeline - Message Processing Actions









Action	Description	More Information
 Delete	Delete a context variable or a set of nodes specified by an XPath expression.	Adding Delete Actions in JDeveloper
 Insert	Insert the result of an XQuery expression at an identified place relative to nodes selected by an XPath expression.	Adding Insert Actions in JDeveloper
 Java callout	Invoke a Java method from the pipeline.	Adding Java Callout Actions in JDeveloper
 MFL transform	Convert non-XML to XML or XML to non-XML in the pipeline.	Adding MFL Translate Actions in JDeveloper
 nXSD translate	Convert native data format (nXSD) to XML or XML to native data format (nXSD) in the pipeline.	Adding nXSD Translate Actions in JDeveloper
 Rename	Rename elements selected by an XPath expression without modifying the contents of the element.	Adding Rename Actions in JDeveloper
 Replace	Replace a node or the contents of a node specified by an XPath expression.	Adding Replace Actions in JDeveloper
 Validate	Validate elements selected by an XPath expression against an XML schema element or a WSDL resource.	Adding Validate Actions in JDeveloper

Table 17-4 Pipeline - Reporting Actions




Action	Description	More Information
 Alert	Send an alert notification based on pipeline message context.	Adding Alert Actions in JDeveloper

Table 17-4 (Cont.) Pipeline - Reporting Actions

Action	Description	More Information
 Log	Construct a message to be logged.	Adding Log Actions in JDeveloper
 Report	Enable message reporting for a pipeline.	Adding Report Actions in JDeveloper

3. Use the following steps to modify the properties for an Action:
 - a. Click the Action to select it. The Properties window displays the properties for the selected component.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. In the Properties window, change the properties, as desired.
 - c. Click **Save** in the Oracle JDeveloper toolbar.
4. When you have finished adding actions, you can further configure the actions in stage or route node, as described in [Table 17-5](#).

Table 17-5 Edit Stage Configuration Tasks

To...	Complete This Step...
Delete an action	Right-click the action in the Pipeline Editor. Select Delete from the context menu that appears.
Move an action	In the Pipeline Editor, click the action icon, and hold the mouse button to drag it. Yellow circles appear to indicate the valid places where you can drop the action. When you drag the action to a yellow circle, it turns green to indicate a valid drop. Release the mouse button to drop the action item on a green circle.
Cut an action	Right-click the action in the Pipeline Editor. Select Cut from the context menu that appears.
Copy an action	Right-click the action in the Pipeline Editor. Select Copy from the context menu that appears.

Table 17-5 (Cont.) Edit Stage Configuration Tasks

To...	Complete This Step...
Paste an action that you have cut or copied	<p>In the Pipeline Editor, right-click the component where you wish to paste the action. Select Paste from the context menu.</p> <p>You can copy and paste actions across stages. However, in the case of Assign, Replace or Insert actions, note the following:</p> <ul style="list-style-type: none"> • All variable-related and user-defined namespaces from the source (copied) stage are added as user-defined namespaces in the target (pasted) stage. • Duplicate namespaces (identical namespaces in both source and target stage) are not copied. • Conflicting namespaces (namespace declarations that use the same prefix but different URIs) are copied. Users will be able to save the configuration, but will not be able activate it until the conflicting namespace declarations in stage B are removed.

5. Click **Save** in the Oracle JDeveloper toolbar.
6. In the Pipeline Editor, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in the Console](#).
7. Click **Save** in the Oracle JDeveloper toolbar.

Adding Publish Actions in JDeveloper

Use a publish action to identify a statically specified target service for a message and to configure how the message is packaged and sent to that service. For more information on publish behavior, see [Performing Transformations in Pipelines](#).

Before you begin

These instructions assume you are already editing a pipeline in the Pipeline Editor, as explained in [Viewing and Editing Pipelines in the Console](#).

They also assume you have already added a pipeline stage, a route node, or an error handler stage. See:

- [Adding Pipeline Pair Nodes to Pipelines in JDeveloper](#)
- [Adding Stages to Pipelines in JDeveloper](#)
- [Adding Error Handlers in JDeveloper](#)

To add a Publish action to a pipeline:

1. Use one of the following methods to add a Publish action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the Publish action. For example, right-click a Stage to add a Publish action to it. Select **Insert Into > Publish** from the context menu that appears.

- Drag the **Publish** component from the Components window to the appropriate location in the Pipeline Editor. To find the Publish component, select the Message Flow category in the Components window. Next, look under the Communication section for the **Publish** icon.

As you drag the Publish component to the editor window, yellow circles appear to indicate the valid places where you can drop the Publish component. When you drag the Publish component to one of the yellow circles, the circle changes to green indicating that you can drop the Publish component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the Publish action:
 - a. Click the **Publish** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. In the Properties window, click the Search icon next to **Service** to specify a target service for the message.
 - c. If the service has operations specified, you can specify an operation to be invoked by selecting it from the **Operation** list.
 - d. Click the **General** tab on the left side of the Properties window.
 - e. Under **Description**, enter an optional description for the action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Publish Table Actions in JDeveloper

Use a publish table action to publish a message to zero or more statically specified services. Switch-style condition logic is used to determine at runtime which services will be used for the publish. For more information on publish behavior, see [Performing Transformations in Pipelines](#).

To add Publish Table actions:

1. Use one of the following methods to add a Publish Table action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the Publish action. For example, right-click a Stage to add a Publish action to it. Select **Insert Into > Publish Table** from the context menu that appears.
 - Drag the **Publish Table** component from the Components window to the appropriate location in the Pipeline Editor. To find the Publish component, select the Message Flow category in the Components window. Next, look under the Communication section for the **Publish Table** icon.

As you drag the Publish Table component to the editor window, yellow circles appear to indicate the valid places where you can drop the Publish Table component. When you drag the Publish Table component to one of the yellow

circles, the circle changes to green indicating that you can drop the Publish Table component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the Publish Table action:
 - a. In the Pipeline Editor, click the **Publish Table** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. In the Properties window, click the **fx** icon next to **Value** to specify an XQuery expression. At runtime, the XQuery expression returns the value upon which the routing decision is made.
 - c. Under **Description**, specify an optional description for the Publish Table action.
3. Click a **Case** construct in the Publish Table node to select it. The Properties window displays the properties for the selected node.
4. In the Properties window, select an operator, and a value for the XQuery expression in the **Value** field.
5. Configure the **Publish** action corresponding to the **Case**. See [Adding Publish Actions in JDeveloper](#) for more information on configuring Publish actions.
6. Optionally add more cases by clicking the **Add Case** icon. Repeat steps 3 to 5 for each new case that you add.
7. Configure the **Publish** action corresponding to the **Default** case. See [Adding Publish Actions in JDeveloper](#) for more information on configuring Publish actions.
8. Click **Save** in the Oracle JDeveloper toolbar.

Adding Dynamic Publish Actions in JDeveloper

Use a dynamic publish action to publish a message to a service specified by an XQuery expression. For more information on publish behavior, see [Performing Transformations in Pipelines](#).

To add Dynamic Publish actions:

1. Use one of the following methods to add a Dynamic Publish action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the Dynamic Publish action. For example, right-click a Stage to add a Dynamic Publish action to it.
Select **Insert Into > Dynamic Publish** from the context menu that appears.
 - Drag the **Dynamic Publish** component from the Components window to the appropriate location in the Pipeline Editor. To find the Dynamic Publish component, select the Message Flow category in the Components window. Next, look under the Communication section for the **Publish** icon.

As you drag the Dynamic Publish component to the editor window, yellow circles appear to indicate the valid places where you can drop the Dynamic Publish component. When you drag the Dynamic Publish component to one of the yellow circles, the circle changes to green indicating that you can drop the Dynamic Publish component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the Dynamic Publish action:

- a. In the Pipeline Editor, click the **Dynamic Publish** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. In the Properties window, click the **fx** icon next to **Service** to specify an XQuery expression. When complete, the XQuery expression should provide a result similar to:

```
<ctx:route>
<ctx:service isProxy="false">project/folder/businessservicename</
ctx:service>
<ctx:operation>foo</ctx:operation>
</ctx:route>
```

Note:

If a proxy service is being invoked, set `isProxy` to true. If a business service is being invoked, set `isProxy` to false.

The element `operation` is optional.

- c. Click the **General** tab on the left side of the Properties window.
 - d. Under **Description**, specify an optional description for the Dynamic Publish action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Routing Options Actions in JDeveloper

Use the Routing Options action to modify any or all of the following properties for the outbound request in `$outbound`: URI, Quality of Service, Mode, Retry parameters. Although these properties can be modified using Assign, Insert, Replace, or Delete actions on `$outbound`, using Routing options provides a simpler way to perform this task, without requiring knowledge of XPath, XQuery, or the structure of the `$outbound` context variable.

The Routing Options action can only be used where the context variable `$outbound` is valid. It can be added to the following actions:

- Publish
- Dynamic Publish

- Publish Table
- Service Callout
- Routing
- Dynamic Routing
- Routing Table

For more information on routing, see [Modeling Message Flow in Oracle Service Bus](#).

To add a Routing Options action:

1. Use one of the following methods to add a Routing Options action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the Routing Options action. For example, right-click **Request Action** in a Publish node to add a Routing Options action to it.
Select **Insert Into > Routing Options** from the context menu that appears.
 - Drag the **Routing Options** component from the Components window to the appropriate location in the Pipeline Editor. To find the Routing Options component, select the Message Flow category in the Components window. Next, look under the Communication section for the **Routing Options** icon.
As you drag the Routing Options component to the editor window, yellow circles appear to indicate the valid places where you can drop the Routing Options component. When you drag the Routing Options component to one of the yellow circles, the circle changes to green indicating that you can drop the Routing Options component there. Release the mouse button to drop the component.
2. Use the following steps to set the Properties for the Routing Options action:
 - a. In the Pipeline Editor, click the **Routing Options** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. In the Properties window, set any or all of the following:
 - URI:** Click the fx icon to invoke the XQuery Expression Builder. Enter an expression that returns a URI. This overrides the URI for the invoked service.
 - QoS:** Select the Quality of Service option from the list. This overrides the default that is auto computed.
 - Mode:** Select between One-way or Request-Response.
 - Retry Interval:** Specify the number of seconds between retries. This overrides the default configured with the invoked service.
 - Retry Count:** Specify the number of retries the system must attempt before discontinuing the action. This overrides the default configured with the invoked service.

Priority: Click the fx icon to launch the XQuery Expression Builder. Enter an expression that returns a positive integer.

Description: Specify an optional description for the Dynamic Publish action.

3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Service Callout Actions in JDeveloper

Use a service callout action to configure a synchronous (blocking) callout to a Service Bus-registered proxy or business service.

For more information on service callout actions, see [Constructing Service Callout Messages](#).

To add a Service Callout action:

1. Use one of the following methods to add a Service Callout action to the pipeline:

- In the Pipeline Editor, right-click the component icon where you wish to add the Service Callout action. For example, right-click a Stage node to add a Service Callout action to it.

Select **Insert Into > Service Callout** from the context menu that appears.

- Drag the **Service Callout** component from the Components window to the appropriate location in the Pipeline Editor. To find the Service Callout component, select the Message Flow category in the Components window. Next, look under the Communication section for the **Service Callout** icon.

As you drag the **Service Callout** component to the editor window, yellow circles appear to indicate the valid places where you can drop the Service Callout component. When you drag the **Service Callout** component to one of the yellow circles, the circle changes to green indicating that you can drop the **Service Callout** component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the Service Callout action:

- a. In the Pipeline Editor, click the **Service Callout** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Click the **Search** icon next to **Service** to specify a business or proxy service.
- c. If the service you chose in the preceding step is a WSDL-based service and has operations that can be invoked on the service, select the operation from the Operation field.
- d. Under **Configuration**, specify how you want to configure the request and response messages by selecting one of the following options:

Select **Configure Body** to configure the SOAP Body. Selecting this option allows you to use \$body directly.

Select **Configure Payload Document** to configure the payload.

- e. Depending on the kind of service you selected, and on the kind of configuration options you chose for that service, select values for the Request and Response variables that appear.
 - f. Specify an optional **Description** for the Service Callout action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Transport Header Actions in JDeveloper

Use a transport header action to set the header values in messages.

To add a Transport Header action:

1. Use one of the following methods to add a Transport Header action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the Transport Header action. For example, right-click a Stage node icon to add a Transport Header action to it.

Select **Insert Into > Transport Header** from the context menu that appears.

- Drag the **Transport Header** component from the Components window to the appropriate location in the Pipeline Editor. To find the Transport Header component, select the Message Flow category in the Components window. Next, look under the Communication section for the **Transport header** icon.

As you drag the **Transport Header** component to the editor window, yellow circles appear to indicate the valid places where you can drop the Transport Header component. When you drag the **Transport Header** component to one of the yellow circles, the circle changes to green indicating that you can drop the **Transport Header** component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the **Transport Header** action:
 - a. In the Pipeline Editor, click the **Transport Header** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. For the Direction list, select one of the following:

Outbound Request: Select this option to set header values for outbound requests (the messages sent out by a pipeline in route, publish, or service callout actions). This header element is located in the message context as follows:

```
$outbound/ctx:transport/ctx:request/tp:headers
```

Inbound Response: Select this option to set header values for inbound responses (the response messages a proxy service sends back to clients). This header element is located in the message context as follows:

```
$inbound/ctx:transport/ctx:response/tp:headers
```

- c. Optionally select **Copy Headers** to copy the header from inbound message to outbound message and vice versa.
- d. Under Headers, select a **Protocol**, and click the **Add Header** icon, identified by the green plus (+) sign, to add a header. Select an **Action**, specify a **Name** for the Header, and any **Value** that may be required.

The **Set** Action enables you to use an XQuery or XSLT expression to set the value of the header. The **Copy** Action copies the header value from the inbound request if setting value for the outbound request. The **Copy** Action copies the header value from the outbound response if setting value for the inbound response. The **Delete** action removes the header from the request or response metadata.

- e. Repeat the preceding step to add any additional headers.
 - f. To delete a header, you can select the row, and click the **Delete Header** icon, identified by the red X sign.
 - g. Specify an optional **Description** for the Transport Callout action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Dynamic Routing to Route Nodes in JDeveloper

Assign a route for a message based on routing information available in an XQuery resource. This is a terminal action, which means you cannot add another action after this one. However, this action can contain request and response actions. For more information on routing, see [Modeling Message Flow in Oracle Service Bus](#).

To add dynamic routing to a route node:

1. Use one of the following methods to add a Dynamic Routing action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the Dynamic Routing action. For example, right-click a Route node icon to add a Dynamic Routing action to it.

Select **Insert Into > Dynamic Routing** from the context menu that appears.

- Drag the **Dynamic Routing** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Dynamic Routing** component, select the Message Flow category in the Components window. Next, look under the Route section for the **Dynamic Routing** icon.

As you drag the **Dynamic Routing** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Dynamic Routing** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the **Dynamic Routing** action:

- a. In the Pipeline Editor, click the **Dynamic Routing** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Under **Service**, click the **fx** icon to specify an XQuery expression. The result of the XQuery expression should be similar to:

```
<ctx:route>
  <ctx:service isProxy='true'>{$service}</ctx:service>
  <ctx:operation>{$operation}</ctx:operation>
</ctx:route>
```

Note:

If a proxy service is being invoked, set `isProxy` to true. If a business service is being invoked, set `isProxy` to false.

- The service name is the fully qualified service name.
 - The operation element is optional.
-
-

- c. Specify an optional **Description** for the Dynamic Routing action.
 - d. Proceed to add actions to the **Request Action** and **Response Action** branches.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Routing Actions to Route Nodes in JDeveloper

Identify a target service for the message and configure how the message is routed to that service. This is a terminal action, which means you cannot add another action after this one. However, this action can contain request and response actions. For more information on routing, see [Modeling Message Flow in Oracle Service Bus](#).

To add a Routing action to a route node:

1. Use one of the following methods to add a Routing action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the Routing action. For example, right-click a Route node icon to add a Routing action to it.

Select **Insert Into > Routing** from the context menu that appears.

- Drag the **Routing** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Routing** component, select the Message Flow category in the Components window. Next, look under the Route section for the **Routing** icon.

As you drag the **Routing** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Routing** component to one of the yellow circles, the circle

changes to green indicating that you can drop the component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the **Routing** action:
 - a. In the Pipeline Editor, click the **Routing** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

 - b. Click the **Search** icon to the right of the **Service** field to specify a service. The Resource Chooser dialog appears.
 - c. Select a service from the resources, and click **OK**.
 - d. If you selected a WSDL-based service, then select an **Operation** corresponding to the service.
 - e. Specify an optional **Description** for the Routing action.
 - f. Proceed to add actions to the **Request Action** and **Response Action** branches.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Routing Tables to Route Nodes in JDeveloper

A routing table is a set of routes wrapped in a switch-style condition table. It is a shorthand construct that allows different routes to be selected based upon the results of a single XQuery expression. You can nest multiple levels in the stage editor. Identify target services for messages and configure how the messages are routed to these services.

This is a terminal action, which means you cannot add another action after this one. However, this action can contain request and response actions. For more information on routing, see [Modeling Message Flow in Oracle Service Bus](#).

To add a routing table to a route node:

1. Use one of the following methods to add a Routing Table action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the Routing Table action. For example, right-click a Route node to add a Routing Table action to it.
Select **Insert Into > Routing Table** from the context menu that appears.
 - Drag the **Routing Table** component from the Components window to the appropriate location in the Pipeline Editor. To find the Routing Table component, select the Message Flow category in the Components window. Next, look under the Route section for the **Routing Table** icon.
As you drag the **Routing Table** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Routing Table** component to one of the yellow circles, the

circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the Routing Table action:
 - a. In the Pipeline Editor, click the **Routing Table** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. In the Properties window, click the **fx** icon next to **Value** to specify an XQuery expression. At runtime, the XQuery expression returns the value upon which the routing decision is made.
 - c. Under **Description**, specify an optional description for the Routing Table action.
3. Click a **Case** construct in the Routing Table node to select it. The Properties window displays the properties for the selected node.
4. In the Properties window, select an operator, and a value for the XQuery expression in the **Value** field.
5. Configure the **Routing** action corresponding to the **Case**. See [Adding Routing Actions to Route Nodes in JDeveloper](#). for more information on configuring Routing actions.
6. Optionally add more cases by clicking the **Add Case** icon. Repeat steps 3 to 5 for each new case that you add.
7. Configure the **Routing** action corresponding to the **Default** case. See [Adding Routing Actions to Route Nodes in JDeveloper](#). for more information on configuring Routing actions.
8. Click **Save** in the Oracle JDeveloper toolbar.

Adding For Each Actions in JDeveloper

Use the for each action to iterate over a sequence of values and execute a block of actions.

To add a For Each action:

1. Use one of the following methods to add a For Each action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the Routing Table action. For example, right-click a Stage node to add a For Each action to it.
Select **Insert Into > For Each** from the context menu that appears.
 - Drag the **For Each** component from the Components window to the appropriate location in the Pipeline Editor. To find the For Each component, select the Message Flow category in the Components window. Next, look under the Flow Control section for the **For Each** icon.

As you drag the **For Each** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **For Each** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the Routing Table action:
 - a. In the Pipeline Editor, click the **For Each** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Enter a value for **For Each Value**. Click **fx** to specify an XPath expression.
 - c. Enter values for **Value Variable**, **Index Variable**, and **Count Variable**.
 - d. Under **Description**, specify an optional description for the For Each action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding If Then Actions in JDeveloper

Use an If Then action to perform an action or set of actions conditionally, based on the Boolean result of an XQuery expression.

To add an If Then action:

1. Use one of the following methods to add an If Then action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the If Then action. For example, right-click a Stage node to add an If Then action to it.

Select **Insert Into > If Then** from the context menu that appears.
 - Drag the **If Then** component from the Components window to the appropriate location in the Pipeline Editor. To find the **If Then** component, select the Message Flow category in the Components window. Next, look under the Flow Control section for the **If Then** icon.

As you drag the **If Then** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **If Then** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.
2. Use the following steps to set the Properties for the If Then action:
 - a. In the Pipeline Editor, click the **If Then** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Specify an optional **Description** for the If Then action.
3. Select the **If: <condition>** branch in the If Then node by clicking on it.
4. In the Properties window, click **fx** to specify an expression for the If condition.
5. Drag actions, corresponding to the condition returning true, from the Components window to the branch.
6. Optionally add more If conditions by clicking on the **Add Condition** icon in the If Then node.
7. Repeat steps 3 to 5 for each additional condition that you add.
8. Click **Save** in the Oracle JDeveloper toolbar.

Adding Raise Error Actions in JDeveloper

Use the raise error action to raise an exception with a specified error code (a string) and description.

To add a Raise Error action:

1. Use one of the following methods to add a raise error action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the raise error action. For example, right-click a Stage node to add a raise error action to it.
Select **Insert Into > Raise Error** from the context menu that appears.
 - Drag the **Raise Error** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Raise Error** component, select the Message Flow category in the Components window. Next, look under the Flow Control section for the **Raise Error** icon.
As you drag the **Raise Error** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Raise Error** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.
2. Use the following steps to set the Properties for the raise error action:
 - a. In the Pipeline Editor, click the **Raise Error** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

To add a Resume action:

1. Use one of the following methods to add a resume action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the resume action. For example, right-click a Stage node, in an error handler, to add a resume action to it.

Select **Insert Into > Resume** from the context menu that appears.

- Drag the **Resume** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Resume** component, select the Message Flow category in the Components window. Next, look under the Flow Control section for the **Resume** icon.

As you drag the **Resume** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Resume** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the reply action:
 - a. In the Pipeline Editor, click the **Resume** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Specify an optional **Description** for the resume action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Skip Actions in JDeveloper

Use the skip action to specify that at runtime, the execution of this stage is skipped and the processing proceeds to the next stage in the message flow. This action has no parameters and can be used in the request, response or error pipelines.

To add a Skip action:

1. Use one of the following methods to add a skip action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the skip action. For example, right-click a Stage node to add a skip action to it.

Select **Insert Into > Skip** from the context menu that appears.

- Drag the **Skip** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Skip** component, select the Message Flow category in the Components window. Next, look under the Flow Control section for the **Skip** icon.

As you drag the **Skip** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Skip** component to one of the yellow circles, the circle changes to

green indicating that you can drop the component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the skip action:
 - a. In the Pipeline Editor, click the **Skip** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Specify an optional **Description** for the skip action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Assign Actions in JDeveloper

Use the assign action to assign the result of an XQuery expression to a context variable.

To add an Assign action:

1. Use one of the following methods to add an assign action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the assign action. For example, right-click a Stage node to add an assign action to it.

Select **Insert Into > Assign** from the context menu that appears.

- Drag the **Assign** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Assign** component, select the Message Flow category in the Components window. Next, look under the Message Processing section for the **Assign** icon.

As you drag the **Assign** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Assign** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the assign action:
 - a. In the Pipeline Editor, click the **Assign** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Click the **fx** icon to the right of the **Value** field to specify an XQuery expression. The XQuery expression specifies the value to be assigned to the named variable.

- c. Enter a context variable in the **Variable** field.
 - d. Specify an optional **Description** for the assign action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Delete Actions in JDeveloper

Use the delete action to delete a context variable or a set of nodes specified by an XPath expression. The delete action is one of a set of update actions.

To add a Delete action:

1. Use one of the following methods to add a delete action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the delete action. For example, right-click a Stage node to add a delete action to it.
Select **Insert Into > Delete** from the context menu that appears.
 - Drag the **Delete** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Delete** component, select the Message Flow category in the Components window. Next, look under the Message Processing section for the **Delete** icon.
As you drag the **Delete** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Delete** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.
2. Use the following steps to set the Properties for the delete action:
 - a. In the Pipeline Editor, click the **Delete** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Under **Location**, specify the context variable.
 - c. Click the **fx** icon to specify an XPath expression. All nodes selected by the XPath expression are deleted.
 - d. Under the General section, specify an optional **Description** for the delete action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Insert Actions in JDeveloper

Use the insert action to insert the result of an XQuery expression at an identified place relative to nodes selected by an XPath expression. The insert action is one of a set of update actions.

To add an Insert action:

1. Use one of the following methods to add an insert action to the pipeline:

- In the Pipeline Editor, right-click the component icon where you wish to add the insert action. For example, right-click a Stage node to add an insert action to it.

Select **Insert Into > Insert** from the context menu that appears.

- Drag the **Insert** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Insert** component, select the Message Flow category in the Components window. Next, look under the Message Processing section for the **Insert** icon.

As you drag the **Insert** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Insert** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the insert action:

- a. In the Pipeline Editor, click the **Insert** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Click the **fx** icon to the right of the **Value** field to specify an XQuery expression. The XQuery expression is used to create the data that is inserted at a specified location in a named variable.
- c. Under **Position**, select the relative location from the list. The relative location is used to control where the insert is performed relative to the result of the XPath expression. Select from the following:

Before: As sibling before each element or attribute selected by the XPath expression.

After: As sibling after each element or attribute selected by the XPath expression.

As first child of: As first child of each element identified by the XPath expression. An error occurs if the result of the XPath returns attributes.

As last child of: As last child of each element identified by the XPath expression. An error occurs if the XPath returns attributes.

- d. Under **Location**, specify the context variable. The XPath evaluates the contents of this variable.
- e. Click the **fx** icon to specify an XPath expression. Valid configurations include those in which:
 - Both the XQuery and XPath expressions return elements.
 - Both the XQuery and XPath expressions return attributes.

3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Java Callout Actions in JDeveloper

Use the Java callout action to invoke a Java method, or EJB business service, from within the pipeline.

To add a Java Callout action:

1. Use one of the following methods to add a Java callout action to the pipeline:

- In the Pipeline Editor, right-click the component icon where you wish to add the Java callout action. For example, right-click a Stage node to add a Java callout action to it.

Select **Insert Into > Java Callout** from the context menu that appears.

- Drag the **Java Callout** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Java Callout** component, select the Message Flow category in the Components window. Next, look under the Message Processing section for the **Java Callout** icon.

As you drag the **Java Callout** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Java Callout** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the Java callout action:

- a. In the Pipeline Editor, click the **Java Callout** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Click the Search icon to the right of the **Method** field to specify a Java archive (jar). Select a Java class and method corresponding to the selected jar file. The method must be a static method.

The Arguments section is populated with the method arguments. An XQuery expression must be specified for each argument.

- c. Click the fx icon under the Value column for an argument to specify an XQuery expression for it.

If the type of the input value you enter does not match the declared input argument type, Service Bus tries to automatically typecast input values to the declared type of the input argument. For example a string value of "123" will be converted to integer 123 if the declared type of the input argument is java primitive int.

- d. Repeat the preceding step for each argument in the selected method.
- e. Under the Return section, specify a **Variable** to which the result is to be assigned.

- f. Optionally, select **As Reference** to return the result as a reference. This option makes the return value of a Java Callout invocation a `<java-content ref="jcid">` reference element regardless of its actual type, where `jcid` is the key to the object in the pipeline object repository.
 - g. Optionally, click the **Search** icon to the right of **Service Account** to specify a service account if there is a security context for the selected Java method.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding MFL Translate Actions in JDeveloper

Use the MFL (Message Format Language) translate action to convert message content from XML to non-XML, or vice versa, in the message pipeline. An MFL is a specialized XML document used to describe the layout of binary data. It is an Oracle proprietary language used to define rules to transform formatted binary data into XML data, or vice versa. See [Defining Data Structures with Message Format Language](#).

To add an MFL Transform action:

1. Use one of the following methods to add an MFL translate action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the MFL translate action. For example, right-click a Stage node to add an MFL translate action to it.
Select **Insert Into > MFL Translate** from the context menu that appears.
 - Drag the **MFL Translate** component from the Components window to the appropriate location in the Pipeline Editor. To find the **MFL Translate** component, select the Message Flow category in the Components window. Next, look under the Message Processing section for the **MFL Translate** icon.
As you drag the **MFL Translate** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **MFL Translate** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.
2. Use the following steps to set the Properties for the MFL translate action:
 - a. In the Pipeline Editor, click the **MFL Translate** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Under **Translate**, select whether you are translating from XML to Native or vice versa.
- c. Click the **fx** icon to the right of the **Input** field to specify an input variable using an XQuery expression.

The input must be text or binary when transforming to XML, and must be XML when transforming to non-XML. Binary content in the message context

is represented by the binary-content XML element. This XML should be the result of the XQuery expression when the input needs to be binary.

- d. Under **MFL**, select **Static** to specify a static MFL resource. Alternatively select **Dynamic** to select a dynamic MFL resource.
 - e. If you selected **Static** in the preceding step, click the **Search** icon to select an MFL resource to perform the MFL translation.
If you selected **Dynamic** in the preceding step, click the **fx** icon to specify an XQuery expression that dynamically selects the MFL resource to perform the translation.
 - f. Specify the **Output** variable to which the result of the translate is assigned.
 - g. Specify an optional **Description** for the MFL translate action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding nXSD Translate Actions in JDeveloper

Use the nXSD translate action to convert message content from XML to native format data, or vice versa, in the message pipeline. See "Native Format Builder Wizard" in *Understanding Technology Adapters* for information on creating native schemas used for translation.

To add an nXSD translate action:

1. Use one of the following methods to add an nXSD translate action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the nXSD translate action. For example, right-click a Stage node to add an nXSD translate action to it.
Select **Insert Into > nXSD Translate** from the context menu that appears.
 - Drag the **nXSD Translate** component from the Components window to the appropriate location in the Pipeline Editor. To find the **nXSD Translate** component, select the Message Flow category in the Components window. Next, look under the Message Processing section for the **nXSD Translate** icon.
As you drag the **nXSD Translate** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **nXSD Translate** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.
2. Use the following steps to set the Properties for the nXSD translate action:
 - a. In the Pipeline Editor, click the **nXSD Translate** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Under **Location**, specify the context variable that holds the element that you wish to rename.
 - c. Click the **fx** icon to specify an XPath expression that is used to specify the data, in the named variable, that is to be renamed.
 - d. Enter a **Local Name** and **Namespace** for the renamed element. At least one of these attributes is required.
 - e. Under the General section, specify an optional **Description** for the rename action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Replace Actions in JDeveloper

Use a replace action to replace a node or the contents of a node specified by an XPath expression. The node or its contents are replaced with the value returned by an XQuery expression. A replace action can be used to replace simple values, elements and even attributes. An XQuery expression that returns nothing is equivalent to deleting the identified nodes or making them empty, depending upon whether the action is replacing entire nodes or just node contents. The replace action is one of a set of update actions.

To add a Replace action:

1. Use one of the following methods to add a replace action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the replace action. For example, right-click a Stage node to add a replace action to it.
Select **Insert Into > Replace** from the context menu that appears.
 - Drag the **Replace** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Replace** component, select the Message Flow category in the Components window. Next, look under the Message Processing section for the **Replace** icon.
As you drag the **Replace** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Replace** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.
2. Use the following steps to set the Properties for the replace action:
 - a. In the Pipeline Editor, click the **Replace** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Under **Location**, specify the context variable that holds the element that you wish to replace.
- c. Click the **fx** icon to specify an XPath expression that is used to specify the data, in the named variable, that is to be replaced.
- d. Click the **fx** icon to the right of the **Value** field to specify an XQuery expression. The XQuery expression is used to create the data that replaces the data specified by the XPath in the named variable.
- e. Under **Replace**, choose one of the following options:

Replace entire node: The nodes selected by the XPath expression are replaced along with all of the contents.

Replace node contents: The node is not replaced; only the node contents are replaced.

Note:

Selecting the **Replace node contents** option and leaving the XPath blank is more efficient than selecting the **Replace entire node** option and setting the XPath to `./*`

- f. Under the General section, specify an optional **Description** for the rename action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Validate Actions in JDeveloper

Use a validate action to validate elements selected by an XPath expression against an XML schema element or a WSDL resource. You can validate global elements only; Service Bus does not support validation against local elements. You can also choose to dynamically select the XML schema element or WSDL resource, at runtime, based on the result of an XQuery expression.

To add a Validate action:

1. Use one of the following methods to add a validate action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the validate action. For example, right-click a Stage node to add a validate action to it.
Select **Insert Into > Validate** from the context menu that appears.
 - Drag the **Validate** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Validate** component, select the Message Flow category in the Components window. Next, look under the Message Processing section for the **Validate** icon.

As you drag the **Validate** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Validate** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.

2. Use the following steps to set the Properties for the validate action:
 - a. In the Pipeline Editor, click the **Validate** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Under **Location**, specify the context variable that holds the element that you wish to validate.
- c. Click the **fx** icon to specify an XPath expression that is used to specify the data, in the named variable, that is to be validated.
- d. Under **Schema**, select **Static** to specify a schema or WSDL file to validate against. Alternatively, select **Dynamic** to dynamically specify the schema or WSDL file at runtime.
- e. If you selected Static in the preceding step, then click the **Search** icon to specify an element contained in an XML schema (.xsd) or WSDL file.

If you selected Dynamic in the preceding step, then click the **fx** icon to specify an XQuery expression. The XQuery expression dynamically specifies a WSDL or schema resource.

Here's an example of dynamically specifying a WSDL resource:

```
<validate xmlns="http://www.bea.com/wli/sb/context">
  <wsdl>default/MyWSDL</wsdl>
  <schemaType>
    <namespaceURI>http://openuri.org</namespaceURI>
    <localname>MyType</localname>
  </schemaType>
</validate>
```

The following is an example of dynamically specifying a schema resource:

```
<validate xmlns="http://www.bea.com/wli/sb/context">
  <schema>default/MySchema</schema>
  <schemaElement>
    <localname>MyElementType</localname>
  </schemaElement>
</validate>
```

- f. Under **Action**, select **Save Variable** to specify a variable that holds the validation result. Alternatively, select **Raise an Error** to raise an error if the element fails validation against the WSDL or XML schema element.
- g. Under the General section, specify an optional **Description** for the validate action.

3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Alert Actions in JDeveloper

Use the alert action to generate alerts based on message context in a pipeline, to send to an alert destination. Unlike SLA alerts, notifications generated by the alert action are primarily intended for business purposes, or to report errors, and not for monitoring system health. Alert destinations should be configured and chosen with this in mind. To learn more about alert destinations, see [Working with Alert Destinations](#).

If pipeline alerting is not enabled for the service or at the domain level, the configured alert action is bypassed during message processing.

To add an Alert action:

1. Use one of the following methods to add an alert action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the alert action. For example, right-click a Stage node to add an alert action to it.
Select **Insert Into > Alert** from the context menu that appears.
 - Drag the **Alert** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Alert** component, select the Message Flow category in the Components window. Next, look under the Reporting section for the **Alert** icon.
As you drag the **Alert** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Alert** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.
2. Use the following steps to set the Properties for the alert action:
 - a. In the Pipeline Editor, click the **Alert** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Click the **fx** icon to the right of the **Content** field to specify an XQuery expression. The XQuery expression is used to specify the message context to be added to the alert message.
- c. Enter a short description of the alert in the **Summary** field.
This becomes the subject line in the case of an Email notification, and can contain no more than 80 characters. If no description is provided, a predefined subject line that reads, "Oracle Service Bus Alert", is used instead.
- d. Select the **Severity** level of the alert.
- e. Click the Search icon to the right of the Destination field to select the alert destination.

By default, the alert goes to the Administration Console.

- f. Under the General section, specify an optional **Description** for the alert action.
3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Log Actions in JDeveloper

Use the log action to construct a message to be logged and to define a set of attributes with which the message is logged.

Before you begin

To see log data in the log file or standard out (server console), WebLogic Server logging must be set to the following severity levels:

- Minimum severity to log: Info
- Log file: Info
- Standard out: Info

For information on setting log severity levels, see "Using Log Severity Levels" in *Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server*.

To add a Log action:

1. Be sure Logging is enabled globally. For more information, see "Configuring Operational Settings at the Global Level" in *Administering Oracle Service Bus*.
2. Use one of the following methods to add a log action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the log action. For example, right-click a Stage node to add a log action to it. Select **Insert Into > Log** from the context menu that appears.
 - Drag the **Log** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Log** component, select the Message Flow category in the Components window. Next, look under the Reporting section for the **Log** icon.

As you drag the **Log** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Log** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.
3. Use the following steps to set the Properties for the log action:
 - a. In the Pipeline Editor, click the **Log** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Click the **fx** icon to the right of the **Content** field to specify an XQuery expression. The XQuery expression is used to specify the message context to be logged.
- c. Enter a short description of the alert in the **Summary** field. This description is logged along with the result of the previously defined expression.
- d. Select the **Severity** level of the log. [Table 17-6](#) describes the options available.

Table 17-6 Log Action Severity Levels

Severity Level	Typical Usage
Info	Used for reporting normal operations; a low-level informational message.
Warning	A suspicious operation or configuration has occurred but it might not affect normal operation.
Error	A user error has occurred. The system or application can handle the error with no interruption and limited degradation of service.
Debug	While your application is under development, you might find it useful to create and use messages that provide verbose descriptions of low-level activity within the application.

- e. Under the General section, specify an optional **Description** for the log action.
4. Click **Save** in the Oracle JDeveloper toolbar.

Adding Report Actions in JDeveloper

Use the report action to enable message reporting for a pipeline.

To add a Report action:

1. Use one of the following methods to add a report action to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the report action. For example, right-click a Stage node to add a report action to it.
Select **Insert Into > Report** from the context menu that appears.
 - Drag the **Report** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Report** component, select the Message Flow category in the Components window. Next, look under the Reporting section for the **Report** icon.
As you drag the **Report** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Report** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.
2. Use the following steps to set the Properties for the report action:

- a. In the Pipeline Editor, click the **Report** node to select it. The Properties window displays the properties for the selected node.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Click the **fx** icon to the right of the **Content** field to specify an XQuery expression. The XQuery expression is used to create the data that is reported to the Service Bus dashboard.
- c. Under Search Keys, click the **Add Key** icon, identified by the green plus (+) sign, to add a search key. The **Key** field is used to specify a key name. The **Variable** field and **XPath** field together specify the key value. The **Variable** field specifies the context variable and the **XPath** field specifies an XPath expression.

You use key value pairs to extract key identifiers from any message context variable or message payload, and ignore the rest of the message. The keys are a convenient way to identify a message. They are displayed as report indexes in the Reporting module. See and "Working with Message Reports" in *Administering Oracle Service Bus*.

For example, consider a report action configured on an error handler in a stage. The action reports the contents of the `fault` context variable in the event of an error. The report action is configured as follows:

- Key name = `errorCode`

- Key value = `./ctx:errorCode` in variable `fault`

Each time this action is executed at runtime, a message is reported through the Reporting Data Stream. The following table shows the results after the report action is executed twice.

Report Index	DB TimeStamp	Inbound Service	Error Code
errorCode=OSB-382505	04/26/07 9:45 AM	MortgageBroker/ProxySvcs/loanGateway3	OSB-382505
errorCode=OSB-382505	04/26/07 9:45 AM		OSB-382505

- d. You can choose to add more keys by repeating the preceding step. To delete a key, click the **Delete Key** icon, identified by the red **X** sign.
- e. Under the General section, specify an optional **Description** for the report action.

3. Click **Save** in the Oracle JDeveloper toolbar.

Adding Error Handlers in JDeveloper

Implementing error handlers in JDeveloper is very similar to using the Oracle Service Bus Console. The primary difference is the drag and drop feature of the pipeline designer, and the properties window that appears for each pipeline action.

How to Add Error Handlers in Pipelines in JDeveloper

Use an error handler to specify what should happen if an error occurs in a specific location in the pipeline. You can configure error handling at the pipeline, pipeline pair, route node, and stage level.

Before you begin

Display the message flow for the desired pipeline.

To add an error handler to a pipeline:

1. Use one of the following methods to add an error handler to the pipeline:
 - In the Pipeline Editor, right-click the component icon where you wish to add the error handler. You can add an error handler for the following:
 - To add an error handler at the pipeline level, right-click the start node. Select **Add Error Handler** from the context menu.
 - To add an error handler for a request or response pipeline, right-click the **Request Pipeline** or the **Response Pipeline**. Select **Add Error Handler** from the context menu.
 - To add an error handler for a stage node, right-click the **Stage** node. Select **Add Error Handler** from the context menu.
 - To add an error handler for a route node, right-click the **Route** node. Select **Add Error Handler** from the context menu.
 - Drag the **Error Handler** component from the Components window to the appropriate location in the Pipeline Editor. To find the **Error Handler** component, select the Message Flow category in the Components window. Next, look under the Nodes section for the **Error Handler** icon.

As you drag the **Error Handler** component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the **Error Handler** component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.
2. To change the default name and add a description for the error handler stage, do the following:
 - a. Click the error handler stage node to select it. The Properties window displays the properties for the selected component.

Note:

If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Edit the **Name** and **Description** fields for the selected stage node.
- c. Click **Save** in the Oracle JDeveloper toolbar.

3. To add actions to the error handler stage, see [Adding and Editing Actions in Pipelines in JDeveloper](#).
4. To add more stages to the error handler, see [Adding Error Handlers in JDeveloper](#).
5. On the Pipeline Editor, continue to construct the pipeline, as described in [Viewing and Editing Pipelines in JDeveloper](#).
6. Click **Save** in the Oracle JDeveloper toolbar.

Disabling an Action or a Stage in JDeveloper

You can choose to disable an action or a stage in a pipeline. A disabled action or stage is skipped from the message flow execution. A disabled stage or action is not validated at design time.

Note:

If a disabled stage has an error handler, then the error handler is also disabled.

When you disable an action or a stage, all the nested actions, if any, are implicitly disabled. This means that the nested actions are also skipped from the message flow execution. If you wish to disable the nested actions from design time validation, you need to individually disable these actions.

You can still edit the configuration of a disabled action or stage. Refactoring also takes place for disabled actions and stages. This means that if there is a call to a service in the disabled action or stage, and the service gets renamed, then the service callout is automatically updated.

Disabling an Action or Stage

To disable an action or stage:

1. In the Pipeline Editor, right-click the stage or action icon that you wish to disable.
2. Select **Disable** from the context menu that appears.

The action or stage is disabled, and the Disabled icon appears next to it.

3. Click **Save** in the Oracle JDeveloper toolbar.

You can re-enable a disabled stage or action at any time, and the action or stage is no longer skipped in the message flow.

Re-Enable an Action or Stage

To re-enable an action or stage:

1. In the Pipeline Editor, right-click the disabled stage or action icon.
2. Select **Enable** from the context menu that appears.

The Disabled icon next to the stage or action disappears, and the stage or action is enabled.

3. Click **Save** in the Oracle JDeveloper toolbar.

Working with Pipeline Templates

This chapter describes designing prototype message flows using pipeline templates. It also describes how to use Oracle JDeveloper to design and configure pipeline templates and concrete pipelines.

Note:

You cannot create a pipeline template in Oracle Service Bus Console. However, you can see existing template resources in the Project Navigator and view template properties like binding type and message handling options. You can also delete a template, rename a template, and move the template into a different folder or project.

Use pipeline templates to design prototype message flows for proxy services. A pipeline template defines the general shape or pattern of the message flow. Concrete pipelines can then be generated out of the pipeline template. All concrete pipelines use the message flow defined by the pipeline template with designated places where custom logic can be inserted.

The following sections describe working with pipeline templates in Oracle JDeveloper:

- [Adding a Pipeline Template](#)
- [Editing a Pipeline Template](#)
- [Adding Placeholder Blocks to a Pipeline Template Message Flow](#)
- [Locking an Action in a Pipeline Template](#)
- [Creating a Concrete Pipeline from a Pipeline Template](#)
- [Editing the Message Flow for a Concrete Pipeline](#)
- [Converting a Concrete Pipeline in to a Regular Pipeline](#)

Adding a Pipeline Template

Use pipeline templates to design prototype message flows for proxy services. A pipeline template defines the general shape or pattern of the message flow.

How to Add a Pipeline Template

You can add a pipeline template to an open Service Bus project in Oracle JDeveloper.

To add a pipeline template:

1. Make sure you have the Service Bus project open in Oracle JDeveloper.

2. Use one of the following methods to add a new pipeline template:
 - In the Application Navigator, right-click the Service Bus project icon and select **New > Pipeline Template** from the context menu.
 - From the **File** menu, select **New > Pipeline Template**.The **Create Pipeline Template** wizard appears.
3. Under **Service Name**, enter a name for the pipeline template.
4. Click the **Browse** icon to the right of the **Location** field to specify a directory for the pipeline template. The default directory is the Service Bus project folder.
5. Optionally, enter a **Description** for the pipeline template.
6. If you wish to use an existing pipeline to define the template, select **From Pipeline**. Click the **Browse** icon to select a pipeline file.
7. Click **Next**. The Service Type page appears.
8. Select a Service Type for the pipeline template. All concrete pipelines created from this template would use the same service type.

Select **Any** if you wish to have the option of selecting the service type when creating a concrete pipeline.
9. Click **Finish** to create the pipeline template. The pipeline template (.ptx) opens in the Template Designer. The pipeline template also appears in the Application Navigator.
10. In the Template Designer, select the **Configuration** tab at the bottom left. The Configuration tab is used to specify message handling configuration settings and other general settings.
11. Click General to specify an optional Description for the pipeline template.

Editing a Pipeline Template

After creating a pipeline template, you can edit it using the Template Designer. You can specify settings like message handling configuration settings for the template, and edit the message flow for the template.

How to Edit a Pipeline Template

Use the Template Designer to configure the settings and message flow for a pipeline template.

To edit a pipeline template:

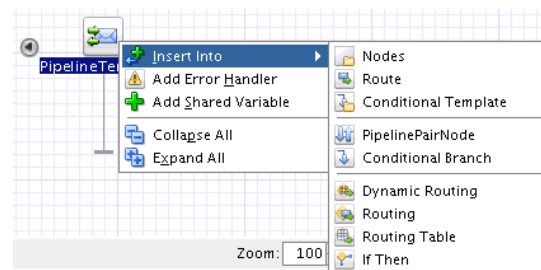
1. In the Application Navigator, double-click the pipeline template to open it in the Template Designer.
2. Click the **Configuration** tab at the bottom left to specify General configuration settings, like Description, and message handling settings, like content streaming and transaction settings. Press **F1** to get help related to the settings.
3. Click the **Design** tab at the bottom left to edit the message flow for the pipeline template.

If no message flow has yet been created for the pipeline template, the Template Designer Design view shows a single icon on the page, the Proxy Service icon. This is the starting node for the message flow.

4. Use one of the following methods to add a message flow component (node) to the message flow:
 - Right-click the start node (proxy service icon) to get options for message flow components that you can add.

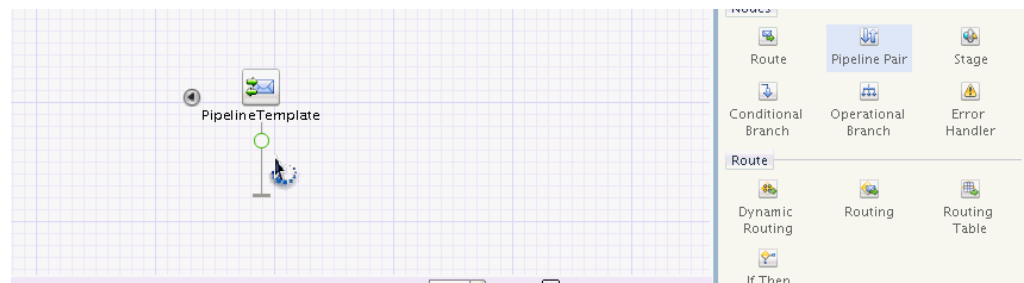
Figure 18-1 shows the options available for the start node. You can add nodes like the PipelinePair node and Conditional Branch. You can also add template placeholders like Nodes, Route, and Conditional Template. Template placeholders are placeholders for actual nodes that you can add when creating concrete pipelines.

Figure 18-1 Right-Clicking a Node to Add a Message Flow Component



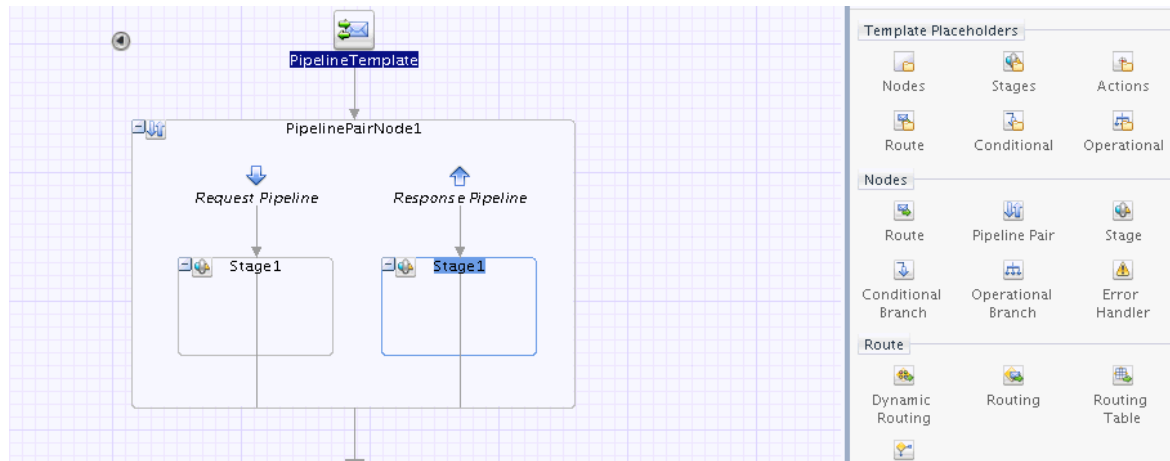
- Alternatively, select the message flow component to add from the Components window, and drag the component to the Template Designer window. Yellow circles appear indicating valid places to drop the component in the message flow. Drag the component to a yellow circle. The yellow circle turns green. Release the component to add it. Figure 18-2 shows a Pipeline Pair node being added to a start node.

Figure 18-2 Adding a Pipeline Pair to the Start Node



When you add components to the Template Designer, corresponding icons are displayed on the Template Designer to represent the components. The relationships among the components are shown with lines and bounding boxes.

Figure 18-3 shows a message flow where a Pipeline Pair node has been added to the flow. The Pipeline Pair comprises of a Request Pipeline and a Response Pipeline. The Request and Response pipelines have stages that can contain action nodes.

Figure 18-3 Message Flow with a Pipeline Pair Node

5. Continue to build the message flow by adding more components. For example, to add an Actions placeholder to the stage node, you can drag the Actions component from the Components window to the stage node in the editor. Alternatively, you can right-click the Stage node to get options for message flow components that you can add to the Stage node. The options available for each component may differ, depending on context.
6. Click **Save** in the Oracle JDeveloper toolbar.

How to View External Services

The external services listed in the Start Node are those invoked outside the context of the pipeline. They are specified in an Invoke Service node but are listed here for convenience.

To view external services:

To view external services, click the left-arrow button on the Start Node. The External Services box appears to the left of the Start Node. Hover your mouse over an external service to view the complete path of the service resource.

How to View Shared Variables

The shared variables listed in the Start Node are the variables that the pipeline can share with other pipelines in the same call chain.

Note:

If two pipelines in a single call chain declare the same shared variable, then they read and modify the same variable in the scope of the invocation call chain. In other words, if pipeline P1 declares a shared variable var, and pipeline P1 invokes pipeline P2, which also declares a shared variable var, then any changes to var in P1 are visible in P2, and vice versa. A shared variable must be of the String, Boolean, or XML data type.

To view external services:

To view shared variables, click the left-arrow button on the Start Node. The Shared Variables box appears to the left of the Start Node. You can right-click the Shared

Variables box to get a context menu. The context menu enables you to perform tasks like add or delete shared variables.

Adding Placeholder Blocks to a Pipeline Template Message Flow

When a pipeline template defines a template placeholder of a particular type, a concrete pipeline can add zero or more nodes of the same type to the template placeholder. For example, the actions template placeholder can contain zero or more action nodes in the concrete pipeline.

Template placeholders can be of the following types:

- **Nodes:** A nodes template placeholder can contain the following nodes:
 - Concrete pipeline pair nodes that contain stages/actions
 - Conditional branch nodes
 - Operational branch nodes
 - Route node

For example, a node placeholder block may contain a pipeline pair node followed by a route node in the concrete pipeline.

- **Stages:** A stages template placeholder can contain zero or more stages in the concrete pipeline. Each stage can in turn contain actions.
- **Actions:** An actions template placeholder can contain zero or more actions in the concrete pipeline.
- **Route:** A route template placeholder is a placeholder for routing actions in the concrete pipeline.
- **Conditional:** A conditional template placeholder enables you to specify a conditional branch node for the concrete pipeline.
- **Operational:** An operational template placeholder enables you to specify an operational branch node for the concrete pipeline when using WSDL SOAP binding.

Locking an Action in a Pipeline Template

If you have specified all required properties for an action added to a pipeline template message flow, and you wish all concrete pipelines to use the same values for the properties, then you can lock the action in the pipeline template message flow.

A locked action cannot be edited in the concrete pipeline. You must make sure that you have specified all the required properties, and any optional properties, in the pipeline template.

How to Lock an Action in a Pipeline Template

You can lock an action from the Properties window in Template Designer.

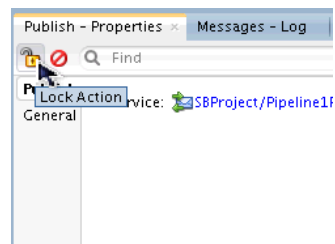
Before you begin:

Make sure you are editing the message flow in the Template Designer, as described in [Editing a Pipeline Template](#)

To lock an action:

1. Select the action in the Template Designer. The properties for the action appear in the Properties window.
2. Make sure you have specified all the required properties specific to the selected action.
3. In the Properties window, click the **Lock** icon to lock the action.

Figure 18-4 Locking an Action in a Pipeline Template



Creating a Concrete Pipeline from a Pipeline Template

Concrete pipelines implement the message flow pattern defined by the pipeline template. You can customize the message flow for the concrete pipeline at designated places.

How to Create a Concrete Pipeline

You can use the pipeline creation wizard to create a concrete pipeline based on a pipeline template.

To create a concrete pipeline:

1. Make sure that the service bus project is open in Oracle JDeveloper.
2. Use one of the following methods to create a new pipeline:
 - Under the **File** menu, click **New > Pipeline**. The Create Pipeline Service wizard appears.
 - In Application Navigator, right-click the service bus project icon. Select **New > Pipeline**. The Create Pipeline Service wizard appears.
 - In Application Navigator, right-click the pipeline template from which you wish to create the new pipeline. Select **Service Bus > Generate Pipeline**.
The Create Pipeline Service wizard appears. The name of the pipeline template is already populated in the **From Template** field.
3. Under **Service Name**, enter a name for the new concrete pipeline.
4. Click the **Browse** icon to the right of the **Location** field to select the location for the concrete pipeline resource. The default location is the service bus project folder.
5. Specify an optional **Description** for the concrete pipeline.
6. If not already selected, select **From Template** to specify the pipeline template. Click the **Browse** icon to the right of the From Template field to select the pipeline template. Select the pipeline template file and click **OK**.

7. Click **Next**.
8. Select the **Service Type** for the concrete pipeline. The options available depend on the service type specified for the underlying template.
9. Click **Finish** to create the concrete pipeline.

Editing the Message Flow for a Concrete Pipeline

The concrete pipeline inherits its message flow from the pipeline template. You can complete the message flow in the Pipeline Editor. You can add nodes and actions to template placeholders in the message flow. In addition, you can edit or complete the properties for other unlocked actions defined in the pipeline template. You cannot edit locked actions in the concrete pipeline.

How to Edit the Message Flow for a Concrete Pipeline

Use the Pipeline Editor to edit the message flow for a concrete pipeline. You can add nodes and actions to template placeholders, and edit other unlocked actions defined in the pipeline template.

To edit the message flow for a concrete pipeline:

1. Make sure you have the Service Bus project open in Oracle JDeveloper.
2. Use one of the following methods to edit the message flow for a pipeline:
 - In Application Navigator, locate the pipeline node. Right-click the pipeline node and select **Open**. You can alternatively double-click the pipeline node to open it.
 - In Application Navigator, click the project node (or overview.xml) to open the Overview Editor.

In the Overview Editor, double-click the pipeline component to open the Pipeline Editor.

The Pipeline Editor appears. Ensure that the **Design** tab is selected at the bottom left corner of the editor.

The Pipeline Editor shows the message flow designed in the pipeline template.

3. To edit the property for an unlocked action:
 - a. Select the action in the Pipeline Editor message flow by clicking on it. The Properties window displays the properties for the selected action.

Note:

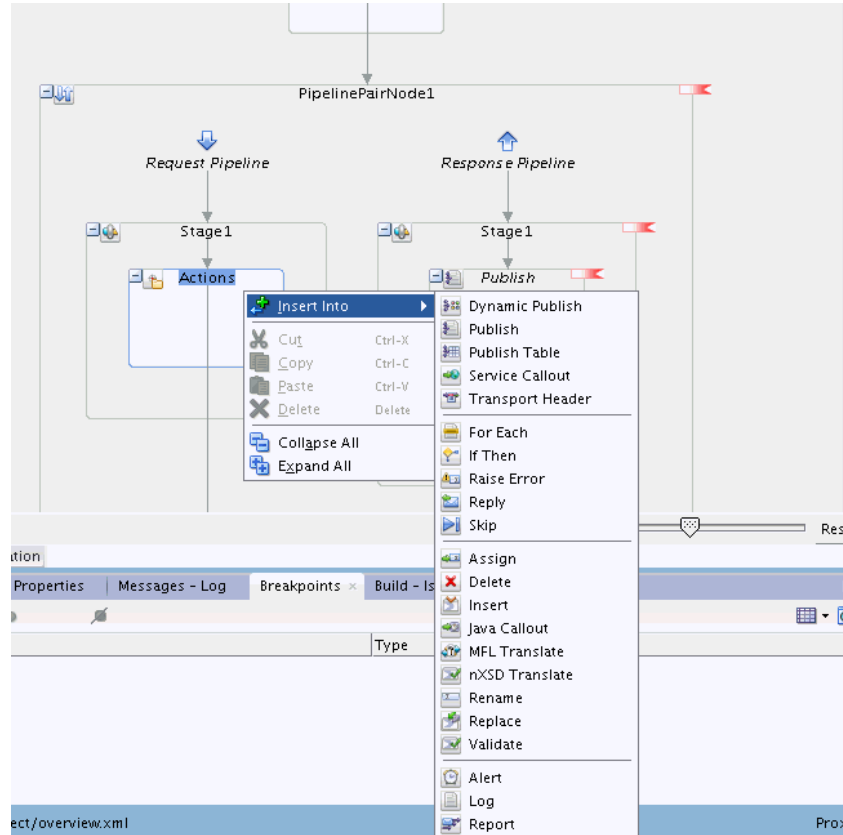
If the Properties window is not visible, select **Properties** from the **Window** main menu.

- b. Edit the properties in the Properties window.
4. Use one of the following methods to edit a template placeholder:

- Right-click the template placeholder icon to get options for message flow components that you can add. Click **Insert Into** from the context menu that appears.

Figure 18-5 shows the options available for the actions template placeholder.

Figure 18-5 Adding a Node to a Template Placeholder



- Drag the desired component from the Components window to the template placeholder in the Pipeline Editor.
As you drag the component to the editor window, yellow circles appear to indicate the valid places where you can drop the component. When you drag the component to one of the yellow circles, the circle changes to green indicating that you can drop the component there. Release the mouse button to drop the component.
5. Continue to build the message flow by editing more actions, or dropping more nodes into template placeholders.
 6. Click **Save** in the Oracle JDeveloper toolbar.

Converting a Concrete Pipeline in to a Regular Pipeline

If you no longer need a concrete pipeline to be associated with a pipeline template, you can break the template link to convert the pipeline in to a regular pipeline. The regular pipeline can be edited without the restrictions applicable to concrete pipelines.

How to Break a Template Link for a Concrete Pipeline

You can use the Configuration tab of the Pipeline Editor to break its link with the associated template.

To break the template link for a concrete pipeline:

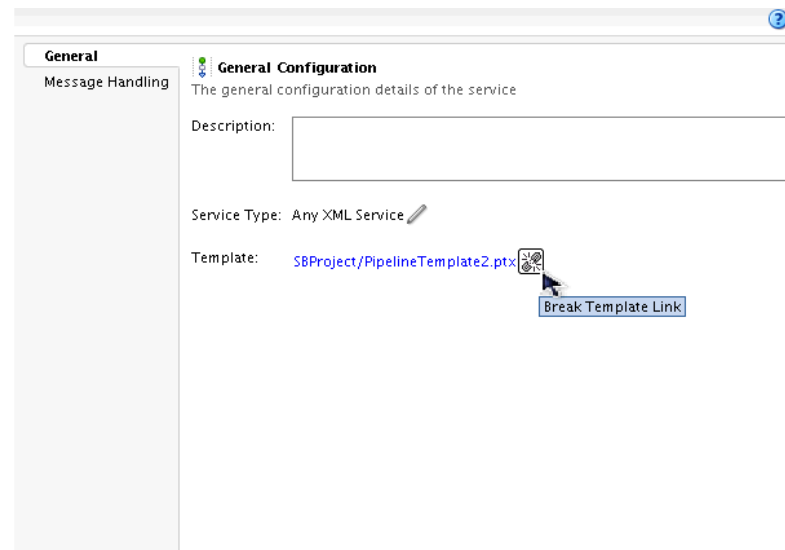
1. Make sure you have the Service Bus project open in Oracle JDeveloper.
2. Use one of the following methods to open the Pipeline Editor:
 - In Application Navigator, locate the concrete pipeline node. Right-click the pipeline node and select **Open**. You can alternatively double-click the pipeline node to open it.
 - In Application Navigator, click the project node (or overview.xml) to open the Overview Editor.

In the Overview Editor, double-click the concrete pipeline component to open the Pipeline Editor.

The Pipeline Editor appears. Ensure that the **Configuration** tab is selected at the bottom left corner of the editor.

3. Click **General** to display the General Configuration page.
4. Under **Template**, click the **Break Template Link** icon. [Figure 18-6](#) shows the Break Template Link icon.

Figure 18-6 *Breaking a Template Link*



Part IV

Transforming Data

This part describes tools you can use to map, transform, and translate the data in messages processed by Service Bus.

This part contains the following chapters:

- [Transforming Data with XQuery](#)
- [Transforming Data with XSLT](#)
- [Mapping Data with Cross-References](#)
- [Mapping Data with Domain Value Maps](#)
- [Defining Data Structures with Message Format Language](#)
- [Using Java Callouts and POJOs](#)

Transforming Data with XQuery

This chapter describes how to create, locate, edit, and delete XQuery Transformation resources using the Oracle Service Bus Console.

XQuery transformation maps can describe XML-to-XML, XML to non-XML, and non-XML to XML mappings.

This chapter includes the following topics:

- [Introduction to XQuery Transformations](#)
- [XQuery Editors and Mappers](#)
- [Creating XQuery Maps in JDeveloper](#)
- [Testing Service Bus Projects Converted from XQuery 2004 to XQuery 1.0 in JDeveloper](#)
- [Working with XQuery Resources in the Oracle Service Bus Console](#)
- [Service Bus XQuery Functions](#)

Introduction to XQuery Transformations

XQuery helps in querying XML data from XML documents. XQuery uses and extends XPath to help navigate and extract elements and attributes from an XML document.

Service Bus uses XQuery to implement its business logic. Service Bus makes use of XQuery resources for various activities, like transformations, data selection, condition evaluation, and data manipulation. Service Bus fully supports XQuery 1.0. This includes optional features such as modules. The older XQuery 2004 is also supported.

XQuery transformation maps describe the mapping between two data types. XQuery maps describe mappings between XML documents with different schemas. Using XQuery, Service Bus can process XML documents and transform document data from one XML schema to another, enabling data interchange among applications that use different schemas. You can perform complex data manipulation and transformation using XQuery. For example, you can map an incoming purchase order schema to an outgoing invoice schema.

You use XQuery expressions to create the data content for the message context variables (or part of a message context variable) during the execution of the message flow. You can use the Test Console directly in the XQuery Expression Editor to test the definition of the expression. Similarly, you use XQuery conditions to evaluate Boolean conditions in the message flow. You can use the Test Console directly in the XQuery Condition Editor to test the definition of the condition.

XQuery Editors and Mappers

JDeveloper provides both an Expression Builder, where you can script transformations using XQuery, and an XQuery Mapper, where you can create complex mappings.

The Oracle Service Bus Console provides an editor for scripting transformations using XQuery. The editor provides options to define an XQuery expression or to define an expression that evaluates at runtime to the name of an existing XQuery resource.

For both JDeveloper and the console, you access the editors from an action in either a pipeline or split-join.

JDeveloper Editors and Mappers

The XQuery mapper in JDeveloper is a graphical tool that lets you define mappings between schema root elements, WSDL message parts, or WSDL messages. Schema root elements can come from XSD schema files or WSDL files, but only those WSDL messages that contain a single message part can be mapped directly. Once you create an XSLT mapping in JDeveloper, you can upload the `.xsl` file generated by the mapper to an XSLT resource in the Oracle Service Bus Console.

JDeveloper also includes a variety of Expression Builders, where you can create expressions that specify an existing XSLT resource to use. For more information about the mapper and editors in JDeveloper, see the following topics:

- "Creating Transformations with the XQuery Mapper" in *Developing SOA Applications with Oracle SOA Suite*
- "Building XPath Expressions in the Expression Builder in Oracle JDeveloper" in *Developing SOA Applications with Oracle SOA Suite*

Oracle Service Bus Console Editors

In the Oracle Service Bus Console, the XQuery/XSLT Expression Editor lets you create expressions that specify an existing XQuery resource to use.

Before you can reference an XQuery resource, you need to create the resource in the console and upload an existing XQuery transformation file (`.xqy`) to the resource. This feature allows you to create complex mappings in JDeveloper that you can then import and use in the console. You can reuse an XQuery transformation in multiple pipelines and split-joins.

For information about the XQuery/XSLT Editor in the Oracle Service Bus Console, see [Working With Expression Editors in Oracle Service Bus Console](#).

Creating XQuery Maps in JDeveloper

You can create XQuery maps in a Service Bus project in JDeveloper, and then use them in XQuery expressions in pipelines and split-joins to map objects between external systems.

How to Create XQuery Mappings in JDeveloper

When you create an XQuery mapping, you need to select the source XML schema elements or XML files to use for the source and target mappings.

See "Creating an XQuery Map File" in *Developing SOA Applications with Oracle SOA Suite* for details on creating an XQuery map.

See "Using the XQuery Mapper" in *Developing SOA Applications with Oracle SOA Suite* for details on using the XQuery mapper to build your XQuery.

Testing Service Bus Projects Converted from XQuery 2004 to XQuery 1.0 in JDeveloper

When converting a Service Bus project from XQuery 2004 to XQuery 1.0, all 2004 XQueries will be switched to run against the XQuery 1.0 engine. After converting from XQuery 2004 to XQuery 1.0, the XQuery Mapper tab in JDeveloper displays, but doesn't display actual mapping.

To test converted XQueries in JDeveloper:

1. Restart JDeveloper.
2. Ensure that the XQuery file that you want to test is open.
3. Click the **XQuery Source** tab to enter Source view.
4. Right click the source, and then select **Run XQuery**.

When testing converted XQueries:

- Ensure that you make the namespace declaration correctly. This can be done in two ways:

- Using an import statement from the XQuery specification:

```
import schema namespace ns0="http://www.example.com/custele"
at "../TestInputSchemas/customerEle.xsd";
```

- Using Oracle's annotation mechanism:

```
xquery version "1.0"; (:: OracleAnnotationVersion "1.0" ::)
declare namespace ns0="http://www.example.com/custele";
(:: import schema at "../TestInputSchemas/customerEle.xsd"::)
```

- Ensure that you declare variables as schema-elements so that they are recognized by the JDeveloper mapper mechanism. For example:

```
declare function local:AttributeToElement($customerOut as element() (::schema-
element(ns0:customerOut)::))
as element() (::schema-element (ns1:customer)::)
```

Working with XQuery Resources in the Oracle Service Bus Console

You can add XQuery resources to your Service Bus project. XQuery files, created using JDeveloper or other editors, can be imported into your project as resources.

- [How to Create an XQuery Resource in the Console](#)
- [How to Edit an XQuery Resource in the Console](#)
- [How to Delete an XQuery Resource in the Console](#)
- [How to Upgrade Your XQuery Resources to use XQuery 1.0](#)

How to Create an XQuery Resource in the Console

Use the Oracle Service Bus Console to add XQuery resources to your Service Bus project. You can either import an XQuery file created in an editor like JDeveloper, or create a resource and edit the code inline.

To create an XQuery Resource in the console:

1. In the Project Navigator, right-click the project or folder to contain the XQuery resource, point to **Create**, and select **XQuery**.

The Create XQuery dialog appears.

2. Do one of the following:

- To create the resource from an existing XQuery file, click **Choose File** next to the **File Upload** field and then navigate to and select the file to use.

The **Resource Name** field is automatically populated with the file name minus the file extension. You can change this name.

- To create an XQuery from scratch, enter a unique name for the XQuery resource.

3. Optionally, enter a brief **Description** of the resource.

4. Click **Create**.

The XQuery resource opens in the XQuery Definition Editor.

5. To modify the XQuery, do the following:

- a. Click **Edit XQuery Contents** in the toolbar.

The View/Edit Source dialog appears.

- b. To browse to and select a new XQuery file to upload, click **Choose File**.

- c. To modify the contents of the file, update the code directly in the **Contents** section of the dialog.

- d. Click **Save**. The XQuery is validated upon save.

6. In the XQuery Definition Editor toolbar, click **Save**.

7. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Edit an XQuery Resource in the Console

Use the Oracle Service Bus Console to edit XQuery resources in your Service Bus project. You can either import an updated XQuery file created in an editor like JDeveloper, or edit the code inline.

To edit an XQuery Resource in the console:

1. In the Project Navigator, expand the project and folders containing the XQuery resource to edit.

2. Right-click the XQuery resource name, and select **Open**.

3. To edit the XQuery source, click **Edit XQuery contents** in the toolbar.

The View/Edit Source dialog appears.

4. To browse to and select a new XQuery file to upload, click **Choose File**.
5. To modify the contents of the file, update the code directly in the **Contents** section of the dialog.
6. Click **Save**.
7. In the XQuery Definition Editor toolbar, click **Save**.
8. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Delete an XQuery Resource in the Console

You can use the Oracle Service Bus Console to delete an XQuery resource from your Service Bus project. If the resource has any references, remove them before deleting it. Open the XQuery resource in the XQuery Definition Editor and click the **References** icon in the upper right to find out whether there are any references.

To delete an XQuery resource in the console:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the XQuery resource to delete.
2. Right-click the name of the XQuery resource, and select **Delete**. A confirmation dialog appears.
3. Click **Yes** to delete the resource.
4. Click **Activate** to end the session and deploy the configuration to the runtime.

How to Upgrade Your XQuery Resources to use XQuery 1.0

Service Bus supports XQuery 1.0. The older XQuery 2004 is also supported. Any new XQuery resource created in Service Bus uses the XQuery 1.0 version, by default.

If you have upgraded from a pre-12g Service Bus project, all XQuery resources in the project are configured to use the XQuery 2004 version. The following line is present as the first line in all XQuery files:

```
xquery version "2004-draft";
```

You can choose to upgrade all XQuery 2004 resources in your project to use XQuery 1.0. The XQuery converter performs basic translation of XQuery 2004 files to XQuery 1.0. You need to manually verify and correct syntax errors that cannot be handled by the converter.

To upgrade the XQuery resources in a project:

1. In the Application Navigator or Project Navigator, right-click the project to upgrade.
2. Select **Convert to XQ 1.0** from the context menu that appears. A confirmation dialog appears.
3. Select **Yes** to convert all resources to use the XQuery 1.0 engine.

You can use the Test Console to test your XQuery resources. To use the test console, open the project by clicking the project name in the Project Navigator.

Click the **Launch Test Console** icon, under the Actions column, corresponding to the XQuery resource that you wish to test.

Service Bus XQuery Functions

Service Bus supports the following XQuery functions:

- The standard XQuery functions described in the W3C specification:
<http://www.w3.org/TR/xpath-functions/>
- Oracle function extensions and language keywords provided as part of the Oracle XQuery engine—with a small number of exceptions, as described in [Supported Function Extensions from Oracle](#).
- Service Bus-specific function extensions. See [Function Extensions from Service Bus](#).

Note:

All of the Oracle function extensions use the following function prefix `fn-bea`: In other words, the full XQuery notation for an extended function is of this format:

```
fn-bea: function_name.
```

Supported Function Extensions from Oracle

For descriptions of all Oracle function extensions, see [Service Bus XQuery Functions](#).

Service Bus supports all Oracle function extensions to XQuery except for the following:

- `fn-bea:is-access-allowed`
- `fn-bea:is-user-in-group`
- `fn-bea:is-user-in-role`
- `fn-bea:userid`
- `fn-bea:async`
- `fn-bea:timeout`
- `fn-bea:get-property`
- `fn-bea:execute-sql()`

Oracle recommends that you do not use the following functions in Service Bus. They are better covered by other language features:

- `fn-bea:if-then-else`
- `fn-bea:QName-from-string`
- `fn-bea:sql-like`

Function Extensions from Service Bus

Service Bus provides the following XQuery functions:

- [fn-bea:lookupBasicCredentials](#)
- [fn-bea:isUserInGroup](#)
- [fn-bea:isUserInRole](#)
- [fn-bea:uuid](#)
- [fn-bea:execute-sql\(\)](#)
- [fn-bea:serialize\(\)](#)

fn-bea:lookupBasicCredentials

The `fn-bea:lookupBasicCredentials` function returns the user name and unencrypted password from a specified service account. You can specify any type of service account (static, pass-through, or user-mapping). See [Working with Service Accounts](#).

Use the `fn-bea:lookupBasicCredentials` function as part of a larger set of XQuery functions that you use to encode a user name and password in a custom transport header or in an application-specific location within the SOAP envelope. You do not need to use this function if you only need user names and passwords to be located in HTTP Authentication headers or as WS-Security user name tokens. Service Bus already retrieves user names and passwords from service accounts and encodes them in HTTP Authentication headers or as WS-Security user name tokens when required.

The function has the following signature:

```
fn-bea:lookupBasicCredentials( $service-account as xs:string ) as
UsernamePasswordCredential
```

where `$service-account` is the path and name of a service account in the following form:

```
project-name[/folder[...]]/service-account-name
```

The return value is an XML element of this form:

```
<UsernamePasswordCredential
  xmlns="http://www.bea.com/wli/sb/services/security/config">
  <username>name</username>
  <password>unencrypted-password</password>
</UsernamePasswordCredential>
```

You can store the returned element in a user-defined variable and retrieve the user name and password values from this variable when you need them.

For example, your Service Bus project is named `myProject`. You create a static service account named `myServiceAccount` in a folder named `myFolder1/myFolder2`. In the service account, you save the user name of `pat` with a password of `patpassword`.

To get the user name and password from your service account, invoke the following function:

```
fn-bea:lookupBasicCredentials( myProject/myFolder1/myFolder2/  
myServiceAccount )
```

The function returns the following element:

```
<UsernamePasswordCredential  
  xmlns="http://www.bea.com/wli/sb/services/security/config">  
  <username>pat</username>  
  <password>patpassword</password>  
</UsernamePasswordCredential>
```

fn-bea:isUserInGroup

Returns whether or not a given user belongs to a given group (true or false). For example:

```
fn-bea:isUserInGroup($user-name as xs:string, $group-name as xs:string)
```

fn-bea:isUserInRole

Returns whether or not a given user belongs to a given role (true or false). For example:

```
fn-bea:isUserInRole($user-name as xs:string, $role-name as xs:string)
```

fn-bea:uuid

The function `fn-bea:uuid` returns a universally unique identifier. The function has the following signature:

```
fn-bea:uuid() as xs:string
```

You can use this function in the proxy pipeline to generate a unique identifier. You can insert the generated unique identifier into an XML document as an element. You cannot generate a unique identifier to the system variable. You can use this to modify a message payload.

For example, suppose you want to generate a unique identifier to add to a message for tracking purposes. You could use this function to generate a unique identifier. The function returns a string that you can add it to the SOAP header.

fn-bea:execute-sql()

The `fn-bea:execute-sql()` function provides low-level database access from XQuery within Service Bus message flows--see [Accessing Databases Using XQuery](#). The query returns a sequence of flat row elements with typed data.

The function has the following signature:

```
fn-bea:execute-sql( $datasource as xs:string, $rowElemName as xs:QName,  
$sql as xs:string, $param1, ..., $paramk) as element()*
```

where

- `$datasource` is the JNDI name of the datasource
- `$rowElemName` is the name of the row element—specify `$rowElemName` as whatever QName you want each element of the resulting element sequence to have
- `$sql` is the SQL statement
- `$param1, ..., $paramk` are 1 to k parameters

- `element()` * represents the sequence of elements returned

The return value is a sequence of flat row elements with typed data and automatically translates values between SQL/JDBC and XQuery data models. Data Type mappings that the XQuery engine generates or supports for the supported databases can be found in the [XQuery-SQL Mapping Reference](#).

When you execute the `fn-bea:execute-sql()` function from a Service Bus message flow, you can store the returned element in a user-defined variable.

Use the following examples to understand the use of the `fn-bea:execute-sql()` function in Service Bus:

- [Example 1: Retrieving the URI from a Database for Dynamic Routing](#)
- [Example 2: Getting XMLType Data from a Database](#)

Example 1: Retrieving the URI from a Database for Dynamic Routing

Service Bus proxy services support specification of the URI to which messages are to be routed at runtime (dynamically)—see [Using Dynamic Routing](#). The following is an example use of the `fn-bea:execute-sql()` function to retrieve the URI from a database in a dynamic routing scenario.

Example - Get the URI for a Business Service from a Database

```
<ctx:route><ctx:service>
{
  fn-bea:execute-sql(
    'ds.myJDBCDataSource',
    xs:QName('customer'),
    'SELECT targetService FROM DISPATCH_MAPPING WHERE customer_priority=?',
    xs:string($body/m:Request/m:customer_pri/text())
  )/TARGETSERVICE/text()
}
</ctx:service></ctx:route>
```

In the example:

- `ds.myJDBCDataSource` is the JNDI name to the data source
- `xs:string($body/m:Request/m:customer_pri/text())` interrogates the request message and populates `customer_priority=?` with the value of `customer_pri` in the message
- `/TARGETSERVICE/text()` is the path applied to the result of the SQL statement, which results in the string (CDATA) contents of that element being returned
- `<ctx:route><ctx:service> ... </ctx:service></ctx:route>` are required elements of the XQuery statement for a dynamic routing scenario
- The following is the table definition for `DISPATCH_MAPPING`:

```
create table DISPATCH_MAPPING
(
  customer_priority varchar2(256),
  targetService varchar2(256),
  soapPayload varchar2(1024)
);
```

The `DISPATCH_MAPPING` table is populated as shown in the following example:

Example - DISPATCH_MAPPING Table

```
INSERT INTO DISPATCH_MAPPING (customer_priority, targetService, soapPayload)
VALUES ('0001', 'system/UCGetURI4DynamicRouting_proxy1', '<something/>');
INSERT INTO DISPATCH_MAPPING (customer_priority, targetService, soapPayload)
VALUES ('0002', 'system/UCGetURI4DynamicRouting_proxy2', '<something/>');
```

Note:

The third column in the table (`soapPayload`) is not used in this scenario.

Executing the `fn-bea:execute-sql` for Example 3

If the XQuery in the Get the URI for a Business Service from a Database example is executed as a result of a proxy service receiving the request message in the following example (note that the value of `<customer_pri>` in the request message is 0001), the URI returned for the dynamic route scenario is

```
system/UCGetURI4DynamicRouting_proxy1
```

Example Request Message \$body

```
<m:Request xmlns:m="http://www.bea.com/alsb/example">
<m:customer_pri>0001</m:customer_pri>
</m:Request>
```

Example 2: Getting XMLType Data from a Database

Data Type mappings that the XQuery engine generates or supports for the supported databases can be found in the [XQuery-SQL Mapping Reference](#). Note that the XMLType column type in SQL is not supported. However, you can access the data in an XMLType column by using the `getStringVal()` method of the XMLType object to convert it to a String value.

The following scenario outlines a procedure you can use to select data from an XMLType column in an Oracle database.

1. Use an assign action in a proxy service message flow to assign the results of the following XQuery to a variable (`$result`).

Example - Get XMLType Data from a Database

```
fn-bea:execute-sql(
  'ds.myJDBCDataSource',
  'Rec',
  'SELECT a.purchase_order.getStringVal() purchase_order from datatypes a'
)
```

where:

- `ds.myJDBCDataSource` is the JNDI name to the data source
- `Rec` is the `$rowElemName`—therefore, `Rec` is the QName given to each element of the resulting element sequence
- `select a.purchase_order.getStringVal() ...` is the SQL statement that uses the `getStringVal()` method of the XMLType object to convert it to a String value
- `datatypes` is the table from which the value of the XML is read (the `datatypes` table in this case contains one row)

Note:

The following is the table definition for the `datatypes` table:

```
create table datatypes
(
  purchase_order xmltype
);
```

2. Use a replace action to replace the node contents of `$body` with the results of the `fn-bea:execute-sql()` query (assigned to `$result` in the preceding step):

```
Replace [ node contents ] of [ undefined XPath ] in [ body ] with
[ $result/purchase_order/text() ]
```

The following listing shows `$body` after the replacement.

Note:

The `datatypes` table contains one row (with the purchase order data); the row contains the XML represented in the following example.

Example - \$body After XML Content is Replaced with Result of fn-bea:execute-sql()

```
<soap-env:Body>
  <openuri:orders xmlns:openuri="http://openuri.com/">
    <openuri:order>
      <openuri:customerID>123</openuri:customerID>
      <openuri:orderID>123A</openuri:orderID>
    </openuri:order>
    <openuri:order>
      <openuri:customerID>345</openuri:customerID>
      <openuri:orderID>345B</openuri:orderID>
    </openuri:order>
    <openuri:order>
      <openuri:customerID>789</openuri:customerID>
      <openuri:orderID>789C</openuri:orderID>
    </openuri:order>
  </openuri:orders>
</soap-env:Body>
```

fn-bea:serialize()

You can use the `fn-bea:serialize()` function if you need to represent an XML document as a string instead of as an XML element. For example, you may want to exchange an XML document through an EJB interface and the EJB method takes `String` as argument. The function has the following signature:

```
fn-bea:serialize($input as item()) as xs:string
```

Creating and Using Custom XPath Functions

You can create and use your own custom XPath functions in both inline XQuery expressions and in XQuery resources. For more information, see [Creating Custom XPath Functions](#).

Transforming Data with XSLT

This chapter provides an overview of eXtensible Stylesheet Language Transformation (XSLT) and how it is used in Service Bus services to map XML input to XML output. It also describes how to create XSLT maps in Service Bus projects.

This chapter includes the following sections:

- [Introduction to XSLT](#)
- [XSLT Editors and Mappers](#)
- [Creating XSLT Mappings in JDeveloper](#)
- [Working with XSLT Resources in the Oracle Service Bus Console](#)
- [How to Delete an XSLT Resource](#)

For more in-depth information about the XSLT mapper in JDeveloper, see "Creating Transformations with the XSLT Map Editor" in *Developing SOA Applications with Oracle SOA Suite*.

Introduction to XSLT

Transformation maps describe the mapping between two data types. eXtensible Stylesheet Language Transformation (XSLT) maps describe mappings between XML documents with different schemas. Using XSLT, Service Bus can process XML documents and transform document data from one XML schema to another, enabling data interchange among applications that use different schemas. You can perform complex data manipulation and transformation using XSLT. For example, you can map an incoming purchase order schema to an outgoing invoice schema.

XSLT Editors and Mappers

JDeveloper provides both an Expression Builder, where you can script transformations using XQuery, and an XSLT Mapper, where you can create complex mappings. The Oracle Service Bus Console provides an editor for scripting transformations using XQuery. These editors provide options to define an XQuery expression, to specify an XSLT resource to execute, or to define an expression that evaluates at runtime to the name of an existing XSLT resource. For both JDeveloper and the console, you access the editors from an action in either a pipeline or split-join.

JDeveloper Editors and Mappers

The XSLT mapper in JDeveloper is a graphical tool that lets you define mappings between schema root elements, WSDL message parts, or WSDL messages. Schema root elements can come from XSD schema files or WSDL files, but only those WSDL messages that contain a single message part can be mapped directly. Once you create

an XSLT mapping in JDeveloper, you can upload the `.xsl` file generated by the mapper to an XSLT resource in the Oracle Service Bus Console.

JDeveloper also includes a variety of Expression Builders, where you can create expressions that specify an existing XSLT resource to use. For more information about the mapper and editors in JDeveloper, see the following topics:

- "Creating Transformation with the XSLT Map Editor" in *Developing SOA Applications with Oracle SOA Suite*
- "Building XPath Expressions in the Expression Builder in Oracle JDeveloper" in *Developing SOA Applications with Oracle SOA Suite*

Oracle Service Bus Console Editors

In the Oracle Service Bus Console, the XQuery/XSLT Expression Editor lets you create expressions that specify an existing XSLT resource to use. Before you can reference an XSLT resource, you need to create the resource in the console and upload an existing XSL transformation to the resource. This feature allows you to create complex mappings in JDeveloper that you can then import and use in the console. You can reuse an XSL transformation in multiple pipelines and split-joins.

For information about the XQuery/XSLT Editor in the Oracle Service Bus Console, see [Working With Expression Editors in Oracle Service Bus Console](#).

Creating XSLT Mappings in JDeveloper

You can create XSLT mappings in a Service Bus project in JDeveloper, and then use them in XQuery expressions in pipelines and split-joins to map objects between external systems.

How to Create XSLT Mappings in JDeveloper

When you create an XSLT mapping, you need to select the source XML schema elements or XML files to use for the source and target mappings.

To create an XSLT mapping in JDeveloper:

1. In the Application Navigator in JDeveloper, right-click the Service Bus project or folder in which you want to create the mapping.
2. In the menu that appears, point to **New** and select **XSL Map**.
The Create XSL Map File dialog appears.
3. In the **File Name** field, enter a unique name for the XSLT map file.
4. Optionally enter a new directory location and a brief description for the cross reference.
The location must be within the directory structure of the current application.
5. To define the source schema, do the following:
 - a. Select **Use Source Schema**.
 - b. Click **Browse** next to the Primary Source field.
 - c. On the Select Schema dialog, select whether to use an XML schema or to generate the source directly from an XML file.

- d. Click **Browse** in the section you chose above to navigate to and select the XML schema element or file to use.
You can select an XML schema element from the current application. You can select an XML file from the file system.
 - e. When you have selected the schema element or file, click **OK** on the Select Schema dialog.
 - f. To select additional sources to use as parameters, click **Add Schema** above the Additional Sources table.
6. To define the target schema, select **Use target schema**, and repeat the above steps on the Select Schema dialog to select the XML components to use.
 7. Click **OK** on the Create XSL Map File dialog.
The XSLT Mapper appears with the source and target structures displayed.
 8. To define the mapping logic, see "Editing an XSLT Map in Map Mode" in *Developing SOA Applications with Oracle SOA Suite*.

Working with XSLT Resources in the Oracle Service Bus Console

In the Oracle Service Bus Console, XSLT maps are stored in XSLT resources, which can be reused in any pipelines and split-joins in the current session. You can upload existing XSLT maps into the XSLT resources, and use a text editor to edit them.

- [How to Create XSLT Resources in the Console](#)
- [How to Edit XSLT Resources in the Console](#)

How to Create XSLT Resources in the Console

If you are using the Oracle Service Bus Console, you can add XSL transformations that you first create in JDeveloper and that you then import into a Service Bus project or upload into an XSL transformation resource. For information on importing, see [Importing and Exporting Resources and Configurations](#). Use the procedure below to upload a mapping file into an XSL transformation resource.

Before you Begin

Create the XSL transformation mapping file as described in [Creating XSLT Mappings in JDeveloper](#).

To create an XSLT resource in the console:

1. In the Project Navigator, right-click the project or folder to contain the new XSL transformation, point to **Create**, and select **XSLT**.

The Create Schema dialog appears.

2. Do one of the following:

- To create the resource from an existing XSL mapping file, click **Browse** next to the **File Upload** field and then navigate to and select the file you created in JDeveloper.

The **Resource Name** field is automatically populated with the file name minus the file extension. You can change this name.

- To create a new XSL transformation, enter a unique name for the transformation resource.
3. Optionally, enter a brief description of the resource.
 4. Click **Create**.
The XSLT parameters, if defined, appear in the XSLT Definition Editor.
 5. To modify the schema, do the following:
 - a. Click **Edit Source** in the toolbar.
The Edit Source dialog appears.
 - b. To browse to and select a new mapping file to upload, click **Browse**.
 - c. To modify the contents of the file, update the code directly in the Contents section of the dialog.
 - d. Click **Save**.
 6. In the XSLT Definition Editor toolbar, click **Save**.
 7. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Edit XSLT Resources in the Console

The Oracle Service Bus Console lets you edit an XSLT resource directly or update the contents by uploading a new or updated file.

To edit an XSL transformation in the console:

1. In the Project Navigator, expand the project and folders containing the XSL transformation to edit.
2. Right-click the XSL transformation name, and select **Open**.
3. Click **Edit Source** in the toolbar.
The Edit Source dialog appears.
4. To browse to and select a new XSL transformation file to upload, click **Browse**.
5. To modify the contents of the file, update the code directly in the Contents section of the dialog.
6. Click **Save**.
7. In the XSLT Definition Editor toolbar, click **Save**.
8. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Delete an XSLT Resource

If any resources reference the XSLT resource you want to delete, remove those references before deleting the resource. In the Oracle Service Bus Console, open the XSLT resource in the XSLT Definition Editor and click the **References** icon in the upper right to find out whether it has any references. In JDeveloper, right-click the XSLT resource and select **Explore Dependencies**.

You can delete the transformation even if it is referenced by other resources, though this might result in conflicts due to unresolved references to the deleted resource.

To delete an XSLT resource:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the XSL transformation to delete.
2. Right-click the name of the transformation, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the transformation. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Mapping Data with Cross-References

This chapter provides an overview of cross references and how they are used in Service Bus services to map identifiers for like objects between external systems. It also describes how to create cross references in Service Bus projects.

This chapter includes the following sections:

- [Introduction to Cross References](#)
- [Creating Cross Reference Tables in JDeveloper](#)
- [Working with Cross Reference Resources in the Oracle Service Bus Console](#)
- [Deleting a Cross Reference Resource](#)
- [Populating Cross Reference Tables in Oracle Service Bus](#)

For more in-depth information about cross references, see "Working with Cross References" in *Developing SOA Applications with Oracle SOA Suite*.

Introduction to Cross References

Cross reference tables map identifiers that represent equivalent objects across multiple applications, associating like objects created in different external applications. For example, you can use cross references to map customer identifiers for records that were created in multiple customer management systems. Cross references are similar to domain value maps (DVMs), but cross references can be updated during runtime, allowing you to dynamically integrate values between systems. Cross reference data updated at runtime is persisted in the database.

Cross references can be used across Oracle SOA Suite components. In Service Bus, you can create cross reference tables in both JDeveloper and the Oracle Service Bus Console.

Cross Reference Database Tables

All cross reference mappings are stored in the form of tables. When you create a cross reference table in a Service Bus project in JDeveloper, you only define the metadata for the table (that is, the keys and not the values). All cross reference tables are stored in the database, by default in the `XREF_DATA` table. Cross reference tables can either be custom or generic (the default). Generic tables are all stored in a single database table that is created during installation (`XREF_DATA`); custom tables are each stored in individual database tables that you create. Service Bus uses the standard SOA Suite data source, `jdbc/SOADataSource`, to access the database tables. You can create your own data source with the following requirements:

- The data source must be named `jdbc/xref`, otherwise the runtime will use the default schema.

- The data source must be XA enabled.

For instructions on creating a custom database table, see "How to Create Custom Database Tables" in *Developing SOA Applications with Oracle SOA Suite*.

Cross Reference Functions

In the Service Bus message flow, you can reference cross reference tables from XQuery expressions and XSLT transformations using a set of XRef functions to lookup, populate, and update cross reference entries at runtime based on information in incoming messages. Use these functions to populate the values for the keys you defined when you first created the cross reference table.

You can access the XRef functions in JDeveloper from the XSLT mapper, the XQuery mapper, and the expression editors. In the Oracle Service Bus Console, the functions are available from the expression and condition editors. For information about the Service Bus XRef functions, see [Cross-Reference Functions](#).

Managing Cross Reference Data at Runtime

Fusion Middleware Control provides features to help you manage cross reference data in the runtime. For information and instructions, see "Managing Cross References" in *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Creating Cross Reference Tables in JDeveloper

You can create cross reference tables in a Service Bus project in JDeveloper, and then use them in XQuery expressions in pipelines and split-joins to map objects between external systems.

How to Create Cross Reference Tables in JDeveloper

When you create a cross reference table, you only need to specify a name for the table and the names of the end systems that are sharing the data. You do not need to specify the values for each system in the design time.

To create cross reference tables in JDeveloper:

1. In the Application Navigator in JDeveloper, right-click the Service Bus project or folder in which you want to create the cross reference.

2. In the menu that appears, point to **New** and select **Cross Reference(XREF)**.

The Create Cross Reference(XREF) File dialog appears.

3. In the **File Name** field, enter a unique name for the cross reference file.

Two cross reference tables cannot have same name in the cross reference repository. The file name is the name of the cross reference table with an extension of `.xref`.

4. Optionally enter a new directory location and a brief description for the cross reference.

The location must be within the directory structure of the current application.

5. In the **End System** fields, enter the names for up to two end systems.

The end systems map to the cross reference columns in a cross reference table. You can add more system names once you create the cross reference table. Each name must be unique within a table.

6. Click **OK**.

The Cross Reference Editor appears with the new cross reference table displayed.

7. To add new end systems to the cross reference table, click **Add** above the End Systems table, double-click in the newly added row, and enter the end system name.

8. To create a custom database table for this cross referencing mapping, see "How to Create Custom Database Tables" in *Developing SOA Applications with Oracle SOA Suite*.

Working with Cross Reference Resources in the Oracle Service Bus Console

If you are using the Oracle Service Bus Console, you can create new cross reference tables or you can upload cross reference tables that you first created in JDeveloper into a cross reference (XRef) resource.

- [How to Create Cross Reference \(XRef\) Resources in the Console](#)
- [How to Edit Cross Reference Resources in the Console](#)
- [How to Create a Custom Database Table in the Console](#)

How to Create Cross Reference (XRef) Resources in the Console

When you create a cross reference table, you only need to specify a name for the table and the names of the end systems that are sharing the data. You do not need to specify the values for each system in the design time.

Before you Begin

If you are uploading a cross reference file already created in JDeveloper, as described in [Creating Cross Reference Tables in JDeveloper](#), make sure the file is available on your system.

To create a cross reference resource in the console:

1. In the Project Navigator, right-click the project or folder to contain the new cross reference mapping, point to **Create**, and select **Cross Reference(XREF)**.

The Create Cross Reference (XRef) dialog appears.

2. Do one of the following:

- To upload an existing cross reference table, click **Browse** next to the **File Upload** field and then navigate to and select the file you created in JDeveloper.

The **Resource Name** field is automatically populated with the file name minus the file extension. You can change this name.

- To create a new cross reference table, enter a unique name for the cross reference resource.

3. Optionally, enter a brief description of the resource.
4. Click **Create**.

The new cross reference table appears on the Cross Reference (XREF) Definition Editor.

5. In the Cross Reference (XREF) Definition Editor toolbar, click **Save**.
6. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Edit Cross Reference Resources in the Console

If you are using the Oracle Service Bus Console, use the following procedure to edit cross reference resources.

To edit a cross reference resource in the console:

1. In the Project Navigator, expand the project and folders containing the cross reference to edit.
2. Right-click the cross reference name, and select **Open**.
3. To modify the source file, do the following:
 - a. Click **Edit Source** in the toolbar.
The Edit Source dialog appears.
 - b. To browse to and select a new cross reference file to upload, click **Browse**.
 - c. To modify the contents of the file, update the code directly in the Contents section of the dialog.
 - d. Click **Save**.
4. To add end systems to the cross reference table, click **Add** above the End System table and enter the name of the system in the new row that appears.
5. In the Cross Reference (XREF) Definition Editor toolbar, click **Save**.
6. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Create a Custom Database Table in the Console

As mentioned previously, all the runtime data is stored in the `XREF_DATA` table by default. If you want to create custom database tables, then perform the following steps.

Note:

When you create a cross reference table in JDeveloper, you can run the custom database creation script directly from the Cross Reference Editor. If you use the Oracle Service Bus Console, you create the custom database table manually. For more information, see "How to Create Custom Database Tables" in *Developing SOA Applications with Oracle SOA Suite*.

To create a custom database table in the console:

1. On the Cross Reference (XRef) Definition Editor, select **Enable Optimization**.
2. In the **Table Name** field, enter a custom name for the database table.

This custom database table name must be prefixed with `xref_`, and cannot be `XREF_DATA` or `XREF_DELETED_DATA`.
3. Click **Save**.
4. Create the custom table in the `soainfra` schema of the Oracle Fusion Middleware database using the following syntax:

```
CREATE TABLE TABLE_NAME (
    ROW_ID VARCHAR2(48) NOT NULL,
    SYSTEM1 VARCHAR2(100),
    SYSTEM2 VARCHAR2(100),
    SYSTEM3 VARCHAR2(100),
    LAST_MODIFIED TIMESTAMP NOT NULL
);
```

Where `TABLE_NAME` is the name you specified in [step 2](#) for the custom table, and `SYSTEM1`, `SYSTEM2`, and `SYSTEM3` are the names of the end systems being cross referenced.

Deleting a Cross Reference Resource

If any resources reference the resource you want to delete, remove those references before deleting the resource. You can delete the resource even if it is referenced by other resources, though this might result in conflicts due to unresolved references to the deleted resource.

How to Delete a Cross Reference Resource

Before deleting a cross reference, check for references and dependencies. In the Oracle Service Bus Console, open the cross reference resource in the Cross Reference (XREF) Definition Editor and click the **References** icon in the upper right to find out whether there are any references. In JDeveloper, right-click the XQuery resource and select **Explore Dependencies**.

To delete a cross reference resource:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the resource to delete.
2. Right-click the name of the resource, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the transformation. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Populating Cross Reference Tables in Oracle Service Bus

Before using a cross reference to look up a particular value, you must populate it at runtime. Use the cross reference XPath functions provided with Service Bus to populate the cross-reference tables. The XPath functions let you populate a cross reference column, perform lookups, and delete a column value. These XPath functions

can be used in the Expression Builder to create an expression or in the XSLT Mapper to create transformations. You can access the Expression Builder dialog through several pipeline activities and split-join operations.

For information about the XRef functions provided with Service Bus, see [Cross-Reference Functions](#). For general information and instructions on using functions in the Oracle Service Bus Console expression and condition editors, see [Building Expressions in the Editor Workspace Text Fields](#).

For information, examples, and instructions on managing cross reference data in JDeveloper, see the following topics in *Developing SOA Applications with Oracle SOA Suite*. The information in these topics is also helpful when working in the Oracle Service Bus Console.

- [Populating Cross Reference Tables](#)
- [Looking Up Cross Reference Tables](#)
- [Deleting a Cross Reference Table Value](#)

Mapping Data with Domain Value Maps

This chapter provides an overview of domain value maps and how they are used in Service Bus services to associate terms used by different domains to describe like objects. It also describes how to create domain value maps in Service Bus projects.

This chapter includes the following sections:

- [Introduction to Domain Value Maps](#)
- [Creating Domain Value Maps in JDeveloper](#)
- [Working with DVM Resources in the Oracle Service Bus Console](#)
- [Deleting a Domain Value Map](#)
- [Using Domain Value Maps in Expressions and Conditions](#)

For more in-depth information about domain value maps, see "Working with Domain Value Maps" in *Developing SOA Applications with Oracle SOA Suite*.

Introduction to Domain Value Maps

A domain value map associates values used by one domain for a specific field to the values used by other domains for the same field, providing the capability to map values across vocabularies or systems. For example, you can map country codes, city codes, currency codes, and so on. You might have several domain value maps for one Service Bus project, depending on the number of fields that require mapping. Domain value maps are similar to cross references, but they are based on a static definition. You create and populate domain value maps in the design time, and deploy them to the runtime. Domain value map data is not changed by runtime activities as it is for cross references, but rather the domain value maps are used for lookups only.

Domain value maps can be used across Oracle SOA Suite components. In Service Bus, you can create domain value maps in both JDeveloper and the Oracle Service Bus Console.

Domain Value Map Functions

In the Service Bus message flow, you can reference domain value maps from XQuery expressions and XSLT transformations using a set of DVM functions to lookup values at runtime based on information in incoming messages. Use these functions so Service Bus knows how to map data coming in from one system to data being sent to another system.

You can access the DVM functions in JDeveloper from the XSLT mapper, the XQuery mapper, and the expression editors. In the Oracle Service Bus Console, the functions are available from the expression and condition editors. For information about the Service Bus DVM functions, see [Domain Value Map Functions](#).

Creating Domain Value Maps in JDeveloper

You can create domain value maps in a Service Bus project in JDeveloper, and then use them in XQuery expressions in pipelines and split-joins to map objects between external systems. Since a domain value map typically defines the mapping for only one field, a pipeline or split-join can reference multiple domain value maps.

How to Create a Domain Value Map in JDeveloper

Create and configure domain value maps using the Create Domain Value Map(DVM) File dialog in JDeveloper. This dialog lets you define two domains, each with one value. Upon completion, the Domain Value Map Editor appears so you can define additional domains and corresponding values.

When you create a domain value map, you can specify a name for up to two domains whose values are being mapped, along with the value for each domain. Once you create the map, you can add more domains and values.

To create a domain value map in JDeveloper:

1. In the Application Navigator, right-click the project in which you want to create a domain value map, point to **New**, and select **Domain Value Map(DVM)**.

The Create Domain Value Map(DVM) File dialog appears.

2. In the **File Name** field, enter a unique and descriptive name for the domain value map file. The file name must have an extension of `.dvm`.
3. Optionally enter a new directory location and a brief description for the cross reference.

The location must be within the directory structure of the current application.

4. In the **Domain Name** fields, enter a name for up to two domains.

These are the column names for the domain value map, and each represents the same field in different domains.

Note:

Domain names must be of the type NCName (non-colonized name), which is a valid XML element name with no colons. Each domain name must be unique in a domain value map. You can add more domains later.

5. In the **Domain Value** field, enter the value corresponding to each domain.
6. Click **OK**.

The Domain Value Map Editor appears with the new domain value map displayed.

7. To add new domains to the domain value map, see "How to Add Domains to a Domain Value Map" in *Developing SOA Applications with Oracle SOA Suite*.
8. To add new values to the domain value map, see "How to Add Domain Values to a Domain Value Map" in *Developing SOA Applications with Oracle SOA Suite*.

Working with DVM Resources in the Oracle Service Bus Console

If you are using the Oracle Service Bus Console, you can create new domain value maps or you can upload domain value maps that you first created in JDeveloper into a DVM resource.

- [How to Create DVM Resources in the Console](#)
- [How to Add Domains to a Domain Value Map](#)
- [How to Add Domain Values to a Domain Value Map](#)
- [How to Edit a Domain Value Map in the Console](#)

How to Create DVM Resources in the Console

When you create a DVM, you define the names of up to two end systems that are sharing the data along with the corresponding values for the field covered by the domain value map. After you create the map, you can define additional domains and values in the DVM Definition Editor.

Before you Begin

If you are uploading a domain value map file already created in JDeveloper, as described in [Creating Domain Value Maps in JDeveloper](#), make sure the file is available on your system.

To create a DVM resource in the console:

1. In the Project Navigator, right-click the project or folder to contain the new domain value map, point to **Create**, and select **DVM**.

The Create DVM dialog appears.

2. Do one of the following:

- To upload an existing domain value map file, click **Browse** next to the **File Upload** field and then navigate to and select the file you created in JDeveloper.

The **Resource Name** field is automatically populated with the file name minus the file extension. You can change this name.

- To create a new domain value map, enter a unique name for the DVM resource.

3. Optionally, enter a brief description of the resource.
4. Click **Create**.

The new domain value map appears on the DVM Definition Editor.

5. In the DVM Definition Editor toolbar, click **Save**.
6. Configure domains and values by performing the following steps:
 - [How to Add Domains to a Domain Value Map](#).
 - [How to Add Domain Values to a Domain Value Map](#).

How to Add Domains to a Domain Value Map

You can define additional domains to map, which are represented as columns in the domain value map. Each new domain can contain values that are either to be included in the lookups at runtime or to be used only to qualify the mapping. Domain names must be of the type NCName (non-colonized name), which is a valid XML element name with no colons.

To add a domain to a domain value map:

1. If the DVM resource is not open, locate the DVM resource in the Project Navigator and click its name.
2. In the Map Table, click **Add** and then select **Add Domain**.

The Add Domain dialog appears.

3. In the **Name** field, enter a name for the domain.
4. To set this column as a qualifier column, select **Enable a Qualifier for lookup**.

Tip:

For more information about qualifier domains and qualifier order, see "Qualifier Domains" and "Qualifier Hierarchies" in *Developing SOA Applications with Oracle SOA Suite*.

5. In the **Lookup Order** field, enter a number indicating the priority of the qualifier domain.

This field is enabled only if you selected **Enable a Qualifier for lookup**.

6. Click **Add**.
7. To define a value for the new domain, continue to [How to Add Domain Values to a Domain Value Map](#).

How to Add Domain Values to a Domain Value Map

Domain values are displayed in rows in the domain value map, with each row containing the value to be mapped for each domain. You can add as many domain values as required to fully define the mapping between domains.

To add domain values to a domain value map:

1. If the DVM resource is not open, locate the DVM resource in the Project Navigator and click its name.
2. In the Map Table, click **Add** and then select **Add Domain Values**.

A new row appears beneath the existing rows in the Map Table.

3. In the new row, enter the values for each domain.
4. Repeat the above steps to create any additional values.
5. In the DVM Definition Editor toolbar, click **Save**.
6. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Edit a Domain Value Map in the Console

If you are using the Oracle Service Bus Console, use the following procedure to modify an existing domain value map.

To edit a domain value map in the console:

1. In the Project Navigator, expand the project and folders containing the domain value map to edit.
2. Click the DVM resource name.
3. To modify the source file, do the following:
 - a. Click **Edit Source** in the toolbar.
The Edit Source dialog appears.
 - b. To browse to and select a new cross reference file to upload, click **Browse**.
 - c. To modify the contents of the file, update the code directly in the Contents section of the dialog.
 - d. Click **Save**.
4. To add domains to the map, see [How to Add Domains to a Domain Value Map](#).
5. To add domain values, see [How to Add Domain Values to a Domain Value Map](#).
6. To edit a domain, select the domain in the Map Table and click **Edit**. In the Update Domain dialog, make any of the changes describe in [How to Add Domains to a Domain Value Map](#).
7. To change a value, double-click the value name in the Map Table and enter the new value.
8. In the DVM Definition Editor toolbar, click **Save**.
9. To end the session and deploy the configuration to the runtime, click **Activate**.

Deleting a Domain Value Map

If any resources reference the resource you want to delete, remove those references before deleting the resource. You can delete the resource even if it is referenced by other resources, though this might result in conflicts due to unresolved references to the deleted resource.

How to Delete a Domain Value Map

Before deleting a domain value map, check for resources or dependencies. In the Oracle Service Bus Console, open the DVM in the DVM Definition Editor and click the **References** icon in the upper right to find out whether any services are using it. In JDeveloper, right-click the DVM and select **Explore Dependencies**.

To delete a DVM resource:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the resource to delete.

2. Right-click the name of the resource, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the transformation. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Using Domain Value Maps in Expressions and Conditions

Use the DVM XPath functions provided with Service Bus to define the domain value lookups that will occur during runtime. The XPath functions provide a variety of ways to perform a lookup for either a single value or multiple values. These XPath functions can be used in the Expression Builder to create an expression or in the XSLT Mapper to create transformations. You can access the Expression Builder dialog through several pipeline activities and split-join operations.

For information about the DVM functions provided with Service Bus, see [Domain Value Map Functions](#). For general information and instructions on using functions in the Oracle Service Bus Console expression and condition editors, see [Building Expressions in the Editor Workspace Text Fields](#).

For information, examples, and instructions on using DVM functions in JDeveloper, see "Using Domain Value Map Functions" in *Developing SOA Applications with Oracle SOA Suite*. This information is also helpful when working in the Oracle Service Bus Console.

Defining Data Structures with Message Format Language

This chapter provides instructions for defining MFL structures in JDeveloper, which can then be used in XQuery Mapper tools to automatically transform data between XML and non-XML formats. This chapter also describes how to add MFL resources to projects in the Oracle Service Bus Console.

This chapter includes the following topics:

- [Introduction to the Format Builder](#)
- [Working with MFL Resources in the Oracle Service Bus Console](#)
- [Creating the MFL Message Structure](#)
- [Configuring the MFL Message Structure](#)
- [Importing and Converting Metadata](#)
- [Deleting MFL Resources](#)
- [Testing Format Definitions](#)
- [Using the Palette](#)
- [Format Builder Supported Data Types](#)
- [Format Builder Field Reference](#)

Introduction to the Format Builder

The Format Builder tool helps you create descriptions of non-XML data records, letting you to describe the layout and hierarchy of the non-XML data so that it can be transformed to or from XML. With Format Builder, you can describe sequences of bytes as fields. Each field description includes the type of data (floating point, string, and so on), the size of the data, and the name of the field. Format Builder allows you to further define groupings of fields (Groups), repetition of fields and groups, and aggregation.

About MFL Files

The descriptions you create in Format Builder are saved in an XML grammar called Message Format Language (MFL). MFL documents are used at runtime to transform an instance of a non-XML data record to an instance of an XML document (or vice-versa). A Message Format Language (MFL) document is a specialized XML document used to describe the layout of binary data. It is an Oracle proprietary language you can use to define rules that transform formatted binary data into XML data. An MFL document conforms to the `mfl.dtd`, which includes elements and attributes that

describe each field of data, as well as groupings of fields (groups), repetition, and aggregation.

When you create business services or proxy services of Messaging Service type, you can select MFL types as the request message type or the response message type of the service.

Valid Names for Formats, Fields, and Groups

Message formats, fields, and groups are identified by a name. This name is used as the XML tag when non-XML data is transformed to XML. Therefore the name must conform to the XML rules for a name.

The format guidelines for a name are as follows:

- Must start with a letter or underscore.
- Can contain letters, digits, colon, the period character, the hyphen character, or the underscore character.

The following are valid name examples:

```
MyField
MyField1
MyField_again
MyField-again
```

The following are invalid name examples:

```
1MyField - may not start with a digit
My>Field - the greater-than sign (>) is an illegal character
My Field - a space is not permitted
My/Field - the back slash (/), which is an illegal character
My\Field - the forward slash (\), which is an illegal character
My:Field - a semi-colon (;), which is an illegal character
```

Supported Character Delimiters

You can specify delimiters in Format Builder by entering the correct syntax. For example, to specify a tab character as the delimiter ('`\u009`'), enter the construct `\t` to match it.

Table 23-1 Character Delimiters

Construct	Matches
x	The character x
\\	The backlash
\\0n	The character with octal value 0n (<= n <= 7)

Table 23-1 (Cont.) Character Delimiters

Construct	Matches
<code>\0nn</code>	The character with octal value 0nn (0 <= n <= 7)
<code>\0mnn</code>	The character with octal value 0mnn (0 <= m <= 3, 0 <= n <= 7)
<code>\xhh</code>	The character with hexadecimal value 0xhh
<code>\uhhhh</code>	The character with hexadecimal value 0xhhhh
<code>\xff</code>	The end-of-file (EOF) character
<code>\t</code>	The tab character ('\u0009')
<code>\n</code>	The newline (line feed) character ('\u000A')
<code>\r</code>	The carriage-return character ('\u000D')
<code>\f</code>	The form-feed character ('\u000C')
<code>\a</code>	The alert (bell) character ('\u0007')
<code>\e</code>	The escape character ('\u001B')
<code>\cx</code>	The control character corresponding to x

For more information, see <http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html>.

Working with MFL Resources in the Oracle Service Bus Console

When you use the Format Builder in JDeveloper to define the hierarchy of a binary record, the layout of fields, and the grouping of fields and groups, the information is saved as an MFL document that can then be used to perform runtime translations. An MFL document can also be used in Format Builder to generate the corresponding DTD that describes its content model.

How to Create MFL Resources in the Console

You define the message format using the Format Builder in JDeveloper. If you are using the Oracle Service Bus Console, you can create MFL resources to upload the MFL files that you already created in JDeveloper. You can also import the MFL resources into the console. For more information, see [Importing and Exporting Resources and Configurations](#). Use the procedure below to upload an MFL file into an MFL resource.

Before you Begin

Create the MFL file in the Format Builder in JDeveloper, as described in [Creating the MFL Message Structure](#).

To add an MFL resource in the console:

1. In the Project Navigator, right-click the project or folder to contain the new MFL file, point to **Create**, and select **MFL**.
The Create MFL dialog appears.
2. Click **Browse** next to the **File Upload** field and then navigate to and select the MFL file you created in JDeveloper.
The **Resource Name** field is automatically populated with the file name minus the file extension. You can change this name.
3. Optionally, enter a brief description of the resource.
4. Click **Create**.
The text of the MFL file appear in the MFL Definition Editor.
5. To modify the schema, do the following:
 - a. Click **Edit Source** in the toolbar.
The Edit Source dialog appears.
 - b. To browse to and select a new MFL file to upload, click **Browse**.
 - c. To modify the contents of the file, update the code directly in the Contents section of the dialog.
 - d. Click **Save**.
6. In the MFL Definition Editor toolbar, click **Save**.
7. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Edit MFL Resources in the Console

The recommended way to modify an MFL file is to use the Format Builder in JDeveloper. For more information about working with the Format Builder, see [Creating the MFL Message Structure](#).

If you are using the Oracle Service Bus Console, you can upload the updated file from JDeveloper, or you can modify the code directly in the console (not recommended).

To update an MFL resource in the console:

1. In the Project Navigator, expand the project and folders containing the MFL resource to edit.
2. Right-click the resource name, and select **Open**.
3. Click **Edit Source** in the toolbar.
The Edit Source dialog appears.
4. To browse to and select a new or updated MFL file to upload, click **Browse**.
5. To modify the contents of the file, update the code directly in the Contents section of the dialog.
6. Click **Save**.
7. In the MFL Definition Editor toolbar, click **Save**.
8. To end the session and deploy the configuration to the runtime, click **Activate**.

Creating the MFL Message Structure

An MFL message structure can contain one or more fields, groups of fields, references and comments.

- [Using Drag and Drop in the Format Builder](#)
- [How to Create an MFL File in JDeveloper](#)
- [How to Create a Group](#)
- [How to Create a Field](#)
- [How to Reference Groups or Fields](#)
- [How to Add a Comment](#)

Using Drag and Drop in the Format Builder

You can use the drag and drop feature of the Format Builder to copy or move the items in the tree view. To move an item, simply drag and drop the item to its new location. To copy an item, press and hold the **CTRL** key while you drag and drop the item.

How to Create an MFL File in JDeveloper

Creating an MFL file automatically generates the root node of the message format file with the same name that you give to the MFL file. The root node name must comply with XML element naming conventions because it becomes the root element in the transformed XML document.

To create an MFL file in JDeveloper:

1. In the Application Navigator, right-click the project or folder to contain the new MFL resource, point to **New**, and select **MFL**.

The Create MFL dialog appears.

2. Enter a name and, optionally, a brief description of the resource.

3. Click **Finish**.

The Format Builder appears.

4. Define the message format using any of the following instructions.

How to Create a Group

Groups define fields that are related in some way (for example, the fields `PAYDATE`, `HOURS`, and `RATE` could be part of the `PAYINFO` group). You can create a group as a child of the message format item, as a child of another group, or as a sibling of a group or field.

To create a group:

1. Select a node in the tree view in the left pane.
2. Choose **Insert > Group > As Child** if you want to create the group as the child of the message format or another group. Choose **Insert > Group > As Sibling** if you want to create the group as the sibling of another group or a field. The Group Details window displays in the right pane.

3. Enter data in the fields as appropriate.

For additional information and instructions, see [Configuring the MFL Message Structure](#) and [Group Configuration Window](#).

4. Click **Apply** to save your changes to the message format file, or click **Reset** to discard your changes to the detail window and reset all fields to the last saved value.

How to Create a Field

Fields are a sequence of bytes that have some meaning to an application. (For example, the field `EMPNAME` contains an employee name.) You can create a field as a child of the message format item, as a child of a group, or as a sibling of a group or another field.

To create a field:

1. Select a node in the tree view in the left pane.
2. Choose **Insert > Field > As Child** if you want to create the field as the child of the message format or group. Choose **Insert > Field > As Sibling** if you want to create the group as the sibling of another group or a field. The Field Details window displays in the right pane.

3. Enter data in the fields as appropriate.

For additional information and instructions, see [Configuring the MFL Message Structure](#) and [Field Configuration Window](#).

4. Click **Apply** to save your changes to the message format file, or click **Reset** to discard your changes to the detail window and reset all fields to the last saved value.

How to Reference Groups or Fields

References indicate that the description of the field or group format has been previously defined and you want to reuse this description without re-entering the data. Reference fields or groups have the same format as the original field or group,

but you can change only the optional setting and the occurrence setting for the reference field or group. For example, if you have a "bill to" address and a "ship to" address in your data and the format for the address is the same, you only need to define the address format once. You can create the "bill to" address definition and create a reference for the "ship to" address.

Note:

References are named exactly the same as the original item. For example, the "bill to" address definition and the "ship to" address definition would be named the same. If you want to reuse a group definition, create a generic group and embed it within a specific group. For example, in the previous example, you can create an *address* group within a *bill_to* group and reference *address* within a *ship_to* group.

To reference a group or field:

1. Select a field or group in the tree pane.
2. Choose **Edit > Copy**.
3. Choose the proper sibling in the tree.
4. Choose **Edit > Paste > As Reference**.
5. Enter data in the fields as appropriate.

For additional information and instructions, see [Configuring the MFL Message Structure](#) and [Format Builder Reference Configuration Window](#).

6. Click **Apply** to save your changes to the message format file, or click **Reset** to discard your changes to the detail window and reset all fields to the last saved value.

How to Add a Comment

Comments contain notes about the message format or the data transformed by the message format. Comments are included in the message format definition for informational purposes only. You can create a comment as a child or sibling of any message format, group, or field.

To add a comment:

1. Select an item in the tree view in the left pane.
2. Choose **Insert > Comment > As Child** if you want to create the comment as the child of the selected item. Choose **Insert > Comment > As Sibling** if you want to create the comment as the sibling of the selected item. The Comment Details window displays in the right pane.
3. Enter the comment text.
4. Click **Apply** to save your changes to the message format file, or click **Reset** to discard your changes to the detail window and reset all fields to the last saved value.

Configuring the MFL Message Structure

Once you create a field, group, or reference, you can configure certain attributes, such as whether a node is repeating and how often, or whether a node is fixed length or delimited.

- [How to Make a Node Recurring](#)
- [How to Define Delimiters](#)

How to Make a Node Recurring

A node can recur a fixed number of times, a number of times specified in the message, or an unlimited number of times.

To make a node recurring:

1. Double-click the group, field, or reference you want to make recurring in the tree view in the left pane.
2. In the occurrence section, do any of the following:
 - To specify a delimiter that indicates recurrence, select **Repeat Delimiter** and enter the delimiter value in the associated field.
 - To specify a field in the message that will indicate the number of times to repeat, select **Repeat Field** and then select the name of the field from the list of available options.
 - To configure a specific number of times the node repeats, select **Repeat Number** and enter the number in the associated field.
 - If the node can repeat an unlimited number of times, select **Unlimited**.
3. Click **Apply** to save your changes to the message format file, or click **Reset** to discard your changes to the detail window and reset all fields to the last saved value.

How to Define Delimiters

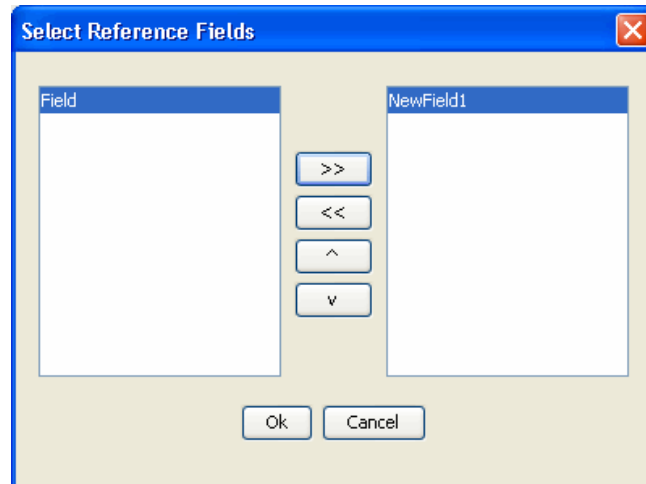
Variable-sized data types can have their termination point specified by a delimiter. A delimiter is a character that marks the end of the field. The field data continues until the delimiter character is encountered. You can specify a delimiter either by reference or by value.

Specifying a Delimiter by Reference

To specify a delimiter by reference:

1. Select the group or field in the tree view in the left pane.
2. In the Termination section, select **Delimiter**.
The Attributes section appears.
3. Click **Ref Fields**.

The Select Reference Fields dialog appears.

Figure 23-1 Select Reference Fields

4. Select any reference fields in the left pane, and click the right-arrow button to move them into the selected fields pane on the right.
5. Click **OK**.
6. In the **Values** field, enter a default delimiter in case the reference field does not exist in the message.
7. Click **Apply** to save your changes to the message format file, or click **Reset** to discard your changes to the detail window and reset all fields to the last saved value.

Specifying a Delimiter by Value

To specify a delimiter by value:

1. Select the group or field in the tree view in the left pane.
2. In the Termination section, select **Delimiter**.

The Attributes section appears.

3. In the **Values** field, enter the delimiter or delimiters separated by the specified separator character.

For example, in the following list, the delimiter can be a comma, tilde, or semi-colon. The separator is a pipe (|).

```
,|~|;
```

4. If the field is optional, select the **Optional** check box. To ensure that the binary data contain the delimiter even if the field is not present, clear the check box.
5. Click **Apply** to save your changes to the message format file, or click **Reset** to discard your changes to the detail window and reset all fields to the last saved value.

Importing and Converting Metadata

Format Builder can import COBOL copybooks and gXML guideline files, and convert a C structure definition into MFL Message Definition.

- [How to Convert a Guideline XML File](#)
- [How to Convert an XML Schema](#)
- [How to Convert a COBOL Copybook](#)
- [How to Convert C Structures](#)
- [How to Convert an FML Field Table Class](#)

How to Convert a Guideline XML File

Format Builder lets you import a guideline XML (gXML) file and convert it into a message definition, which you can modify and customize to suit your needs. gXML is an open specification designed to facilitate exchange of e-commerce guidelines for business documents (like purchase orders, invoices and so on) using XML. gXML version 0.71 is supported in this release.

To convert a gXML file:

1. Choose **Tools > Import > EDI Importer**.

The EDI Importer dialog appears.

2. Enter the gXML file path and name, or click **Browse** to navigate to and select the file to use.
3. Click **OK**.

How to Convert an XML Schema

Format Builder lets you import an XML Schema representing the desired XML representation of your non-XML document. This can provide you with a jump-start on specifying the format of your non-XML document.

To convert an XML schema:

1. Choose **Tools > Import > XML Schema Importer**.

The XML Schema Importer dialog appears.

2. Enter the XML schema file path and name, or click **Browse** to navigate to and select the file to use.
3. In the **Root Element** field, select the element from the XML schema to use as the root node for the MFL file.
4. In the **MFL Field Delimiter Default** field, enter the default value for the delimiter.
5. Click **OK**.

How to Convert a COBOL Copybook

Format Builder lets you import a COBOL copybook into Format Builder and create a message definition to transform the COBOL data. When importing a copybook, comments are used to document the imported copybook and the Groups and Fields it contains.

To convert a COBOL Copybook:

1. Choose **Tools > Import > COBOL Copybook Importer**.

The COBOL Copybook Importer dialog appears.

2. Enter the COBOL Copybook file path and name, or click **Browse** to navigate to and select the file to use.

3. Select one of the following:

- **Big Endian:** Sets the byte order to Big Endian.

Note: This option is used for IBM 370, Motorola, and most RISC designs (IBM mainframes and most Unix platforms).

- **Little Endian:** Sets the byte order to Little Endian.

Note: This option is used for Intel, VAX, and Unisys processors (Windows, VMS, Digital, Unix, and Unisys).

4. Select the character set from the following options:

- EBCDIC
- US-ASCII

Note: The above two values are attributes of the originating host machine.

- **Other** (if you select this option, you must select the character set from the list of available options).

Once you have imported a copybook, you can work with it as you would any message format definition. If an error or unsupported data type is encountered in the copybook, a warning message informs you of the error. You can choose to display the error or save the error to a log file for future reference.

How to Convert C Structures

Format Builder includes a C structure importer utility that converts a C structure definition into an MFL Message Definition by generating MFL or C Code output.

When defining a conversion to MFL, you must provide some profile configuration data to generate the MFL directly. You can do this by creating a new hardware profile, or specifying an existing profile. If the generation is successful, the main Format Builder window appears with the MFL object listed in the navigation tree. The MFL object has the same name as the input file used in the parse operation. If errors are detected during the generation process, the MFL Generation Errors dialog displays providing you the opportunity to view or file the error log. Once you have determined what errors were generated, you can return to the C Structure Importer and repeat the prior steps.

To convert a C Structure:

1. From the Format Builder main window, choose **Tools > Import > C Struct Importer**.

The C Structure Importer dialog appears.

2. Enter the path and name of the file to import, or click **Browse** to navigate to and select a file.

3. Click **Parse**.

The **Structure** field is populated with the list of structures found in the file you selected.

4. Select the structure you want to convert.
5. To generate MFL data, do the following:
 - a. Under Output, select the **MFL** option.
 - b. In the **Structure** field, select the desired structure from the list of available options.
 - c. In the **Name** field, select an existing hardware profile, or click **New** to create a new profile. On the dialog that appears, specify a name and description, modify the primitive data types and byte order, and click **OK**.
Click **Edit** to open the hardware profile editor if you need to view or edit the profile parameters.
 - d. Click **OK** to generate the MFL file.
 - e. Click **Display Error Log** to view any errors encountered, click **Save Error Log** to save the error log to the location of your choice, or click **Cancel** to dismiss the MFL Generation Errors dialog box.
6. To generate C code, do the following:
 - a. Under Output, select the **C Code** option.
 - b. Enter a file name in either the **MFL Gen** or **Data Gen** field, or click **Browse** to select a file.
 - c. Click **OK**.
You will be warned about overwriting existing files and notified about the success or failure of the code generation.
 - d. Copy the generated source code to the platform in question and compile and execute it.

Note:

You must copy the input file containing the structure declarations as well. Both programs, when compiled, take an argument of the output file name.

- e. Copy the generated MFL or data back to the platform running Format Builder.

How to Convert an FML Field Table Class

The FML Field Table Class Importer facilitates the integration of WebLogic Tuxedo Connector and business process management (BPM) functionality. Tuxedo application buffers are translated to and from XML by the FML to XML Translator that is a feature of WebLogic Tuxedo Connector. The integration of Tuxedo with BPM functionality requires the creation of the XML that is passed between the WebLogic Tuxedo Connector Translator and the process engine. To create the necessary XML, use the FML Field Table Class Importer and the XML generation feature of Format Tester.

Before You Begin

1. Move the field tables associated with the FML buffer from the Tuxedo system to the WebLogic Server/WebLogic Tuxedo Connector environment.
2. Use the `weblogic/wtc/jatmi/mkfldclass` utility to build Java source code representing the field tables. For information about FML field table administration, see the WebLogic Server documentation.
3. Compile the source code. The resulting class files are called `fldtbl` classes because they implement the `FldTbl` interface. These class must be packaged in a JAR file that can be selected from the FML Field Table Class Importer dialog.

Note:

Because most users perform these steps when configuring WebLogic Tuxedo Connector, these class files may already exist. If you create Java classes using WebLogic Tuxedo Connector, you can place the `.class` files in the `\ext` directory. You can then populate the Available Fields list automatically from the FML Field Table Class Importer dialog box.

To create an XML document with the FML Field Table Class Importer:

1. Choose **Tools > Import > EDI Importer**.

The FML Field Table Class Importer dialog appears.

2. Click **Select** to select the JAR file containing the `fldtbl` classes.

The `fldtbl` classes are displayed in the Classes list. If the selected JAR file contains no `fldtbl` classes, an error message appears and the Fld Table Jar File and Classes fields are cleared.

3. In the Classes section, select one or more `fldtbl` class names from the list of available classes.
4. In the Available Fields list, select the fields from the list of options and then click **Add**.

This list does not allow duplicate names. Even if the name of a field appears in different field tables, it is included only once in the list.

5. To remove any fields added in error to the Selected Fields list, select the fields and click **Remove**.
6. Click **OK**.

7. Enter data in the fields as described in the following table:

8. Edit the created MFL document to specify the order and number of occurrences of the fields in the XML document to be passed to the WebLogic Tuxedo Connector FML/XML Translator from business process management (BMP).

9. Choose **Tools > Test** to display the Format Tester tool.

10. From the Format Tester menu bar, choose **Generate > XML** to create an XML document that conforms to the MFL document in Format Builder.

11. Edit the data content of the fields in the XML document as desired.
12. From the Format Tester menu bar, choose **File > Save XML** to save the XML document in a file with a specified name and location.

The created XML can be imported and used in business process management functions by using the XML instance editor. For information about importing XML, see the BPM documentation.

Deleting MFL Resources

If any resources reference the MFL file you want to delete, remove those references before deleting the file. You can delete the MFL resource even if it is referenced by other resources, though this might result in conflicts due to unresolved references to the deleted resource.

How to Delete an MFL Resource

Before deleting an MFL resource, check for any references or dependencies. In the Oracle Service Bus Console, open the MFL file in the MFL Definition Editor and click the **References** icon in the upper right to find out whether any services are using it. In JDeveloper, right-click the MFL file and select **Explore Dependencies**.

To delete an MFL resource:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the XSL transformation to delete.
2. Right-click the name of the MFL file, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the MFL resource. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Testing Format Definitions

Once you have built a format definition, you can test it using Format Tester. Format Tester parses and reformats data as a validation test and generates sample non-XML or XML data. This sample data can be edited, searched, and debugged to product the expected results.

How to Start Format Tester

Format Tester is launched from the Format Builder and opens in a separate window.

To start Format Tester:

1. In Format Builder, open a message format document (MFL file).

Note:

To run Format Tester, you must have a message format document open in Format Builder.

2. From the **Format Builder** menu bar, choose **Tools > Test**.
3. The Format Tester dialog box appears.

Format Tester uses the currently loaded message definition document.

How to Test Using the Non-XML Window

The Non-XML data display panel acts as a hexadecimal editor or a text editor, depending on which tab is selected.

The hexadecimal editor panel displays data offsets, the hex value of individual bytes, and the corresponding text. The corresponding text can be optionally displayed as ASCII or EBCDIC characters. The editor allows for editing of the hex byte or the text value. If a hex data value is modified, the corresponding text value is updated, and vice versa.

Using the Data Offset Feature

The data offset feature of the hexadecimal editor allows you to display your data offsets as Hexadecimal or Decimal.

To change your data offsets:

Choose **Display > Hex**. The following two data offset options display.

- Offsets as Hexadecimal
- Offsets as Decimal

Click the display option that best suits your needs. The data offset panel of the Non-XML window dynamically changes to reflect your choice.

Using the Text Feature

To use the Text feature, select the Text tab from within the Non-XML window to view all printable characters, such as carriage returns. The Text window shows these as text with line breaks.

How to Test Using the XML Window

The XML data panel displays XML data that has been converted or transformed from the contents of the Non-XML panel. The contents of the XML panel can be cleared or edited to suit your needs.

You can also use this window to enter or generate the XML data to be transformed into non-XML format.

How to Test Using the Debug Window

The Debug window displays the actions that take place during the transformation operation, any errors that are encountered, and field and group values and delimiters. To determine the location of the error, determine the last field that parsed successfully and examine the specification of the next field on the tree pane of Format Builder.

When you open the Format Tester, only the Non-XML and XML windows are visible. To open the Debug window, choose **Display > Debug** to toggle the Debug window on and off. The Debug window opens below the Non-XML and XML windows.

Note:

Debug output is restricted to the most recent 64 KB of messages. Full debug information can be captured to a file. See [Using the Debug Log](#) for more information.

How to Debug Format Definitions

The following topics discuss the various Format Tester utilities you can use to debug and correct your data.

- [Searching for Values](#)
- [Searching for Offsets](#)
- [Using the Debug Log](#)

Searching for Values

The Find feature allows you to search for hex or text values in the Non-XML data. The following fields are available from the Find dialog.

To search for values:

1. From within the Format Tester, choose **File > Open Non-XML** to open the non-XML data file you want to search.
2. Choose **Edit > Find**.

The Find dialog appears.

3. Enter data in the fields as appropriate. See [Table 23-2](#).

Table 23-2 Find Options

Field	Description
Value	Enter the value you want to find.
Text	Select this option if you want to find a text value.
Hex	Select this option if you want to find a hex value.
Forwards	Select this option if you want to search from the selected location to the end of the document.
Backwards	Select this option if you want to search from the selected location to the beginning of the document.
Beginning of File	Select this option if you want to start the search at the beginning of the file.
Current Position	Select this option if you want to start the search at the current cursor location.
End of File	Select the option if you want to start the search at the end of the file.
OK Button	Begins the search operation.

Table 23-2 (Cont.) Find Options

Field	Description
Cancel Button	Closes the Find dialog without performing a search.

4. Click **OK** to begin the Search operation.

Searching for Offsets

The Goto feature allows you to move the cursor in the Non-XML editor to a byte offset you specify. The following fields are available from the Goto dialog.

To move to a specified offset:

1. From within the Format Tester, choose **File > Open Non-XML** to open the non-XML data file you want to search.

2. Choose **Edit > Go To**.

The Goto dialog appears.

3. Enter data in the fields as appropriate. See [Table 23-2](#).

Table 23-3 Goto Options

Field	Description
Offset	Enter the offset value you want to find.
Dec	Select this option if you want to go to a decimal value.
Hex	Select this option if you want to go to a hex value.
Forwards	Select this option if you want to search from the selected location to the end of the document.
Backwards	Select this option if you want to search from the selected location to the beginning of the document.
Beginning of File	Select this option if you want to start the search at the beginning of the file.
Current Position	Select this option if you want to start the search at the current cursor location.
End of File	Select the option if you want to start the search at the end of the file.
OK Button	Begins the search operation.
Cancel Button	Closes the Goto dialog without performing a search.

4. Click **OK** to begin the Search operation.

Using the Debug Log

The debug log allows you to save your debug information to a text file.

To use the debug log:

- Select **File > Debug Log**.

A dialog appears, where you can enter a new path and file name or choose an existing file in which to save the debug information.

Note:

If you select an existing file, the new debug information is appended to the end of the file.

Format Tester Command Reference

This section describes the commands available from the Format Tester main menu.

- [File Menu](#)
- [Edit Menu](#)
- [Display Menu](#)
- [Generate Menu](#)
- [Transform Menu](#)

File Menu

The following commands are available from the File menu.

Element	Description
Open Non-XML	Lets you select a non-XML file to be displayed in the Non-XML window. Note: The default file extension for non-XML files is.DATA.
Open XML	Lets you select a file to be displayed in the XML section of the Format Tester window. Note: The default file extension for XML files is.XML
Save Non-XML	Saves the contents of the Non-XML window.
Save XML	Saves the contents of the XML window.
Debug Log	Saves the debug information in a text file.
Close	Closes the Format Tester window.

Edit Menu

The following commands are available from the Edit menu.

Element	Description
Cut	Removes the currently selected text and places it on the clipboard for pasting into another location.
Copy	Copies the currently selected text and places it on the clipboard for pasting into another location.

Element	Description
Paste	Inserts the cut or copied text at the cursor location.
Find	Lets you to search for a hex or text value in the non-XML data.
Find Next	Continues your search to the next instance of the specified value.
Go To	Lets you move the cursor in the Non-XML editor to a specified byte offset.

Display Menu

The following commands are available from the Display menu.

Element	Description
XML	Lets the XML data panel be hidden or shown. If hidden, the non-XML data window expands to fill the width of the tester. The <i>To XML</i> button remains, but the splitter disappears.
Debug	Lets the Debug output window be hidden or shown.
Clear > Non-XML	Resets the contents of the Non-XML data window to be empty.
Clear > XML	Resets the contents of the XML window to be empty.
Hex > Offsets as Hexadecimal	Displays the offset values as hexadecimal. Selecting this option turns off the <i>Offsets as Decimal</i> display.
Hex > Offsets as Decimal	Displays the offset values as decimal. Selecting this option turns off the <i>Offset as Hexadecimal</i> display.

Generate Menu

The following commands are available from the Generate menu.

Element	Description
Non-XML	Generates non-XML data to match the current format specification.
XML	Generates XML data to match the current format specification.
Prompt while generating data	Select this option if you want to be prompted during the generation process to determine if optional fields or groups should be generated, determine which choice of children should be generated, and determine how many times a repeating group should repeat.

Transform Menu

The following commands are available from the Transform menu.

Element	Description
Non-XML to XML	Converts the contents of the Non-XML window to XML.

Element	Description
XML to Non-XML	Converts the contents of the XML window to non-XML data.

Using the Palette

The Format Builder palette allows you to store commonly used message format items and insert them into your message format definitions. These items are stored in an XML document, and you can use the standard Windows drag and drop feature to copy items from the palette into your message format definition.

The palette contains some common date formats, literals, and strings. You can use these items in the message formats you create, as well as adding your own items to the palette.

How to Display the Palette Window

To turn the palette display on or off, choose **View > Show palette**. If the palette is not currently displayed, it opens in a separate window next to the Format Builder window. If the palette is currently displayed, its window closes.

How to Add Items to the Palette

You cannot add nodes to the palette that depend on the existence of another node to the palette. For example, you cannot add Field or Group References, and you cannot add items that have a Repeat Field specified. You can add comments, but this is not recommended since comments do not have unique names and therefore are indistinguishable on the palette.

To add items to the palette:

1. From the navigation tree, choose the item you want to add to the palette.
2. Click and hold the left mouse button and drag the item into the palette window.
3. When the item is placed in the position you want it (as a sibling of the selected item), release the mouse button. The item is copied from the navigation tree to the palette window.

How to Add Palette Items to a Message Format

To copy items from the palette to a message format:

1. From the palette window, choose the item you want to add to your message format.
2. Click and hold the left mouse button and drag the item into the left pane of the Format Builder window.
3. When the item is placed in the position you want it (as the child or sibling of the desired item), release the mouse button. The item is copied from the palette to the message format.

Format Builder Supported Data Types

This section provides information about MFL data types, COBOL Copybook Importer data types, and unsupported C language features.

- [MFL Data Types](#)
- [COBOL Copybook Importer Data Types](#)
- [Unsupported C Language Features](#)

MFL Data Types

[Table 23-4](#) lists the MFL data types that data transformer supports. These are metadata data types used in non-XML to XML or XML to non-XML conversions, and are specified in the "type" attribute of a Field Format element.

Table 23-4 Supported MFL Data Types

Data Type	Description
Binary (Base64 encoding)	Any character value accepted. Requires a length, length field, delimiter, or a delimiter field. Resulting XML data for this field is encoded using base-64.
Binary (Hex encoding)	Any character value accepted. Requires a length, length field, delimiter, or a delimiter field. Resulting XML data for this field is encoded using base-16.
Date: DD-MMM-YY	A string defining a date; for example, 22-JAN-00.
Date: DD-MMM-YYYY	A string defining a date; for example, 22-JAN-2000.
Date: DD/MM/YY	A string defining a date; for example, 22/01/00.
Date: DD/MM/YYYY	A string defining a date; for example, 22/01/2000.
Date: DDMMMYY	A string defining a date; for example, 22JAN00.
Date: DDMMMYYYY	A string defining a date; for example, 22JAN2000.
Date: MM/DD/YY	A string defining a date; for example, 01/22/00.
Date: MM/DD/YYYY	A string defining a date; for example, 01/22/2000.
Date: MMDDYY	A six digit numeric string defining a date; for example, 012200.
Date: MMDDYYYY	An eight digit numeric string defining a date; for example, 01222000.
Date: MMM-YY	A string defining a date; for example, JAN-00.
Date: MMM-YYYY	A string defining a date; for example, JAN-2000.
Date: MMMDDYYYY	A string defining a date; for example, JAN222000.
Date: MMMYY	A string defining a date; for example, JAN00.
Date: MMMYYYY	A string defining a date; for example, JAN2000.

Table 23-4 (Cont.) Supported MFL Data Types

Data Type	Description
Date: Wed Nov 15 10:55:37 CST 2000	The default date format of the Java platform; for example, 'WED NOV 15 10:55:37 CST 2000'
Date: YY-MM-DD	A string defining a date; for example, 00-01-22. (The string: 00-01-22 defines the date January 22, 2000.)
Date: YY/MM/DD	A string defining a date; for example, 00/01/22. (The string: 00/01/22 defines the date January 22, 2000.)
Date: YYMMDD	A string defining a date; for example, 000122. (The string: 000122 defines the date January 22, 2000.)
Date: YYYY-MM-DD	A string defining a date; for example, 2000-01-22. (The string: 2000-01-22 defines the date January 22, 2000.)
Date: YYYY/MM/DD	A string defining a date; for example, 2000/01/22. (The string: 2000/01/22 defines the date January 22, 2000.)
Date: YYYYMMDD	An eight byte numeric string of the format YYYYMMDD. A base data of String or EBCDIC may be specified to indicate the character encoding.
DateTime: DD/MM/YY hh:mm	A string defining a date and time; for example, 22/01/00 12:24.
DateTime: DD/MM/YY hh:mm AM	A string defining a date and time; for example, 22/01/00 12:24 AM.
DateTime: DD/MM/YY hh:mm:ss	A string defining a date and time; for example, 22/01/00 12:24:00.
DateTime: DD/MM/YY hh:mm:ss AM	A string defining a date and time; for example, 22/01/00 12:24:00 AM.
DateTime: MM/DD/YY hh:mm	A string defining a date and time; for example, 01/22/00 12:24.
DateTime: MM/DD/YY hh:mi AM	A string defining a date and time; for example, 01/22/00 12:24 AM.
DateTime: MM/DD/YY hh:mm:ss	A string defining a date and time; for example, 01/22/00 12:24:00.
DateTime: MM/DD/YY hh:mm:ss AM	A string defining a date and time; for example, 01/22/00 12:24:00 AM.
DateTime: MMDDYYhhmm	A string of numeric digits defining a date and time; for example, 0122001224.

Table 23-4 (Cont.) Supported MFL Data Types

Data Type	Description
DateTime: YYYYMMDDhhmmss	A fourteen byte numeric string of the format YYYYMMDDHHMISS. A Base data type may be specified.
DateTime: MMDDYYhhmmss	A string of numeric digits defining a date and time; for example, 012200122400.
EBCDIC	A string of characters in IBM Extended Binary Coded Decimal Interchange Code. Requires a length, length field, delimiter, or a delimiter field.
Filler	A sequence of bytes that is not transformed to XML. This field of data is skipped over when transforming non-XML data to XML. When transforming XML to non-XML data, this field is written to the binary output stream as a sequence of spaces.
FloatingPoint: 4 bytes, Big-Endian	A four byte big endian floating point number that conforms to the IEEE Standard 754.
FloatingPoint, 4 bytes, Little-Endian	A four byte little endian floating point number that conforms to the IEEE Standard 754.
FloatingPoint: 8 bytes, Big-Endian	A eight byte big endian floating point number that conforms to the IEEE Standard 754.
FloatingPoint: 8 bytes, Little-Endian	A eight byte little endian floating point number that conforms to the IEEE Standard 754.
Integer: Signed, 1 byte	A one byte signed integer; for example, '56' is 0x38.
Integer: Unsigned, 1 byte	A one byte unsigned integer; for example, '128' is 0x80.
Integer: Signed, 2 byte, Big-Endian	A signed two-byte integer in big endian format; for example, '4660' is 0x1234.
Integer: Signed, 4 byte, Big-Endian	A signed four-byte integer in big endian format; for example, '4660' is 0x00001234.
Integer: Signed, 8 bytes, Big-Endian	A signed eight-byte integer in big endian format; for example, '4660' is 0x0000000000001234.
Integer: Unsigned, 2 byte, Big-Endian	An unsigned two-byte integer in big endian format; for example, '65000' is 0xFDE8.
Integer: Unsigned, 4 byte, Big-Endian	An unsigned four-byte integer in big endian format; for example, '65000' is 0x0000FDE8.
Integer: Unsigned, 8 bytes, Big-Endian	An unsigned eight-byte integer in big endian format; for example, '65000' is 0x000000000000FDE8.
Integer: Signed, 2 bytes, Little-Endian	A signed two-byte integer in little endian format; for example, '4660' is 0x3412.
Integer: Signed, 4 bytes, Little-Endian	A signed four-byte integer in little endian format; for example, '4660' is 0x34120000.

Table 23-4 (Cont.) Supported MFL Data Types

Data Type	Description
Integer: Signed, 8 bytes, Little-Endian	A signed eight-byte integer in little endian format; for example, '4660' is 0x3412000000000000.
Integer: Unsigned, 2 bytes, Little-Endian	An unsigned two-byte integer in little endian format; for example, '65000' is 0xE8FD.
Integer: Unsigned, 4 bytes, Little-Endian	An unsigned four-byte integer in little endian format; for example, '65000' is 0xE8FD0000.
Integer: Unsigned, 8 bytes, Little-Endian	A unsigned eight-byte integer in little endian format; for example, '65000' is 0xE8FD000000000000.
Literal	A literal value determined by the contents of the value attribute. When non-XML data is transformed to XML, the presence of the specified literal in the non-XML data is verified by WLXT. The literal is read, but is not transformed to the XML data. When XML data is transformed to a non-XML format, and a literal is defined as part of the non-XML format, WLXT writes the literal in the resulting Non-XML byte stream.
Numeric	A string of characters containing only digits; for example, '0' through '9'. Requires a length, length field, delimiter, or a delimiter field.
Packed Decimal: Signed	IBM signed packed format. Requires a length, length field, delimiter, or a delimiter field to be specified. The length or length field should specify the size of this field in bytes.
Packed Decimal: Unsigned	IBM unsigned packed format. Requires a length, length field, delimiter, or a delimiter field to be specified. The length or length field should specify the size of this field in bytes.
String	A string of characters. Requires a length, a length field, a delimiter, or a delimiter field. If no length, length field, or delimiter is defined for a data type String, a delimiter of "\x00" (a NUL character) will be assumed.
String: NUL terminated	A string of characters, optionally NUL (\x00) terminated, residing within a fixed length field. This field type requires a length attribute or length field which determines the amount of data read for the field. This data is then examined for a NUL delimiter. If a delimiter is found, data following the delimiter is discarded. If a NUL delimiter does not exist, the fixed length data is used as the value of the field.
Time: h:mm:ss	A string defining a time; for example, 122400.
Time: hh:mm AM	A string defining a time; for example, 12:24 AM.
Time: hh:mm	A string defining a time; for example, 12:24.
Time: hh:mm:ss AM	A string defining a time; for example, 12:24:00 AM.
Time: hh:mm:ss	A string defining a time; for example, 12:24:00.

Table 23-4 (Cont.) Supported MFL Data Types

Data Type	Description
Zoned Decimal: Leading sign	Signed zoned decimal format (US-ASCII or EBCDIC) where the sign indicator is in the first nibble. Requires a length, length field, delimiter, or a delimiter field to be specified. The length or length field should specify the size of this field in bytes. Note: This data type is supported with US-ASCII data only with Message Format Language Version 2.02
Zoned Decimal: Leading separate sign	Signed zoned decimal format (US-ASCII or EBCDIC) where the sign indicator is in the first byte. The first byte only contains the sign indicator and is separated from the numeric value. Requires a length, length field, delimiter, or a delimiter field to be specified. The length or length field should specify the size of this field in bytes. Note: This data type is supported with US-ASCII data only with Message Format Language Version 2.02.
Zoned Decimal: Signed	Signed zoned decimal format (US-ASCII or EBCDIC). Requires a length, length field, delimiter, or a delimiter field to be specified. The length or length field should specify the size of this field in bytes. Note: This data type is supported with US-ASCII data only with Message Format Language Version 2.02.
Zoned Decimal: Trailing separate sign	Signed zoned decimal format (US-ASCII or EBCDIC) where the sign indicator is in the last byte. The last byte only contains the sign indicator and is separated from the numeric value. Requires a length, length field, delimiter, or a delimiter field to be specified. The length or length field should specify the size of this field in bytes. Note: This data type is supported with US-ASCII data only with Message Format Language Version 2.02.
Zoned Decimal: Unsigned	Unsigned zoned decimal format (US-ASCII or EBCDIC). Requires a length, length field, delimiter, or a delimiter field to be specified. The length or length field should specify the size of this field in bytes. Note: This data type is supported with US-ASCII data only with Message Format Language Version 2.02.

COBOL Copybook Importer Data Types

The Format Builder tool provides a utility for the conversion of COBOL copybooks into MFL files. [Table 23-5](#) lists the COBOL data types that can be converted to metadata data types and the support provided by the Importer. Support for these data types is limited. For example, the following formats are converted to an unsigned 4-byte integer type:

```
05 pic 9(5) comp-5
05 pic 9(5) comp-x
```

Additionally, the following generate errors:

```
05 pic X(5) comp-5
05 pic X(5) comp-x
```

In these samples, `pic9(5)` could be substituted for `pic x(5)`.

The following values are defined as follows:

- **Supported:** The data type will be correctly parsed by the importer and converted to a message format field or group.
- **Unsupported:** The data type is not supported and the importer reports an error when the copybook is imported.
- **Ignored:** The data type is parsed and a comment is added to the message format. No corresponding field or group is created.

Table 23-5 COBOL Data Types

COBOL Type	Support
BLANK WHEN ZERO (zoned)	supported
COMP-1, COMP-2 (float)	supported
COMP-3, PACKED-DECIMAL	supported
COMP, COMP-4, BINARY (integer)	supported
COMP, COMP-4, BINARY (fixed)	supported
COMP-5, COMP-X	supported
DISPLAY (alphanumeric)	supported
DISPLAY numeric (zoned)	supported
edited alphanumeric	supported
edited float numeric	supported
edited numeric	supported
group record	supported
INDEX	supported
JUSTIFIED RIGHT	ignored
OCCURS (fixed array)	supported
OCCURS DEPENDING (variable-length)	supported
OCCURS INDEXED BY	ignored
OCCURS KEY IS	ignored
POINTER	supported
PROCEDURE-POINTER	supported
REDEFINES	supported
SIGN IS LEADING SEPARATE (zoned)	supported

Table 23-5 (Cont.) COBOL Data Types

COBOL Type	Support
SIGN IS TRAILING (zoned)	supported
SIGN IS TRAILING SEPARATE (zoned)	supported
SIGN IS LEADING (zoned)	supported
SYNCHRONIZED	ignored
66 RENAMES	not supported
66 RENAMES THRU	not supported
77 level	supported
88 level (condition)	ignored

Some vendor-specific extensions are not recognized by the importer, however, any copybook statement that conforms to ANSI standard COBOL will be parsed correctly by the Importer. The Importer's default data model, which is based on the IBM mainframe model, can be changed in Format Builder to compensate for character set and data "endianness."

When importing copybooks, the importer may identify fields generically that, upon visual inspection, could easily be identified by a more specific data type. For this reason, the copybook importer creates comments for each field found in the copybook. This information is useful in assisting you in editing the MFL data to better represent the original copybook.

For example, this original copybook entry:

```
05 birth-date    picxx/xx/xx
```

results in a field of type EBCDIC with a length of 8. Closer inspection indicates that this is intended to be a date format and could be defined as a field of type `Date`: `MM/DD/YY` or a field of type `Data`: `DD/MM/YY`.

Unsupported C Language Features

The Format Builder provides a utility for the conversion of C structures into MFL files. This section lists the C Language constructs that *cannot* be converted to metadata data types. (This conversion occurs at design time.) The C struct Importer utility does not parse files containing anonymous unions, bit fields, or in-line assembler code. The following samples of unsupported features are taken from the preprocessor output of a `hello.c` file that contained a `#include <windows.h>` statement:

- Anonymous unions

```
#line 353 "e:\\program files\\microsoft visual studio\\vc98\\include\\winnt.h"
typedef union_LARGE_INTEGER{
    struct {
        DWORD LowPart;
        LONG HighPart;
    };
    struct {
        DWORD LowPart;
```

```
        LONG HighPart;
    } u;
#line 363 "e:\\program files\\microsoft visual studio\\vc98\\include\\winnt.h"
        LONGLONG QuadPart;
    } LARGE_INTEGER
```

- **Bit fields**

```
typedef struct_LDT_ENTRY {
    WORD LimitLow;
    WORD BaseLow;
    union {
        struct {
            BYTE BaseMid;
            BYTE Flags1;
            BYTE Flags2;
            BYTE BaseHi;
        } Bytes;
        struct
            DWORD BaseMid : 8;
            DWORD Type : 5;
            DWORD Dpl : 2;
            DWORD Pres : 1;
            DWORD LimitHi : 4;
            DWORD Sys : 1;
            DWORD Reserved_0 : 1;
            DWORD Default_Big : 1;
            DWORD Granularity : 1;
            DWORD BaseHi : 8;
        } Bits;
    } HighWord;
} LDT_ENTRY, *PLDT_ENTRY;
```

- **Inline assembler code**

```
_inline ULONGLONG
_stdcall
Int64Shr1Mod32(
    ULONGLONG Value,
    DWORD ShiftCount
)
{
    _asm {
        mov ecx, ShiftCount
        mov eax, dword ptr [Value]
        mov edx, dword ptr [Value+4]
        shrd eax, edx, cl
        shr edx, cl
    }
}
```

Format Builder Field Reference

This section describes the Format Builder windows and each of the properties displayed on the Format Builder windows.

- [Format Builder Window](#)
- [Format Builder Tool Bar](#)
- [Format Builder Tree Pane](#)

- [Field Configuration Window](#)
- [Group Configuration Window](#)
- [Format Builder Reference Configuration Window](#)

Format Builder Window

The main window of the Format Builder is split into two panes. The left pane shows the structural information for the data format. The right pane shows the detail for the item selected in the left pane.

Format Builder Tool Bar

Use the following Menu bar options to create and configure the structure of the MFL file. The menus that are available depend on what is selected in the left pane.

Element	Description
New	Creates a new message format
Open	Opens an existing message format.
Save	Saves the current message format
Cut	Removes the item currently selected in the left-hand pane, and its child objects, from the tree. Note: This action is not available if the message format (root) item is selected.
Copy	Makes a copy of the item currently selected in the left-hand pane for insertion elsewhere in the tree. Note: This action is not available if the message format (root) item is selected.
Paste as Sibling	Inserts the cut or copied item as a sibling object of the selected item.
Paste as Reference	Inserts a reference to the cut or copied item as a sibling object of the selected item.
Undo	Reverses the previous action. The tool tip changes to indicate the action that can be undone. For example, if you change the name of the field to Field1 and click Apply , the tool tip reads "Redo Apply Field Field1". Note: Format Builder supports multi-level undoing and redoing.
Redo	Reverses the effects of an Undo command. The tool tip changes to indicate the action that can be redone. For example, if you change the name of a field to Field1 and click Undo , the tool tip to read "Redo Apply Field Field1". Note: Format Builder support multi-level undoing and redoing.
Insert Field	Inserts a field as a sibling of the item selected in the tree pane.
Insert Group	Inserts a group as a sibling of the item selected in the tree pane.
Insert Comment	Inserts a comment as a sibling of the item selected in the tree pane.
Move Up	Moves the selected item up one position under its parent.

Element	Description
Move Down	Moves the selected item down one position under its parent.
Promote item	Promotes the selected item to the next highest level in the tree. For example, Field1 is the child object of Group1. Select Field1 and click Promote to make it a sibling of Group1.
Demote item	Demotes the selected item to the next lower level in the tree. For example, Group1 is the sibling of Field1. Field1 immediately follows Group1 in the tree. Select Field1 and click Demote to make it a child of Group1.
Expand All	Expands all items in the tree pane to show child items.
Collapse All	Collapses the tree pane to show first level items only.
Format Tester	Opens the Format Tester window.

Format Builder Tree Pane

The Tree Pane represents hierarchical and structural information about the format of the non-XML data in a tree. The root node of the tree corresponds to the MFL document being created or edited, and you can create groups and fields under the root node. Following are the different types of elements you can create in the tree pane.

Message formats, fields, and groups are identified by a name, which is used as the XML tag when non-XML data is transformed to XML. Therefore the name must conform to the XML rules for a name.

Follow these guidelines when naming the nodes in your format tree:

- Names must start with a letter or underscore.
- Names can contain letters, digits, colon, the period character, the hyphen character, or the underscore character.

Element	Description
Message Format	The top level, or root, element.
Group	A collection of fields, comments, and other groups or references that are related in some way. For example, the fields PAYDATE, HOURS, and RATE could be part of the PAYINFO group. This defines the formatting for all items contained in the group.
Optional Group	A group that may or may not be included in the message format. This type of group is represented by a dotted line around its icon.
Repeating Group	A group that has one or more occurrences. This type of group is represented by a cascading icon.
Optional Repeating Group	A group that may or may not be included, but if included, occurs more than once.
Group Reference	An indicator that another instance of the group exists in the data. Reference groups have the same format as the original group, but you can change the optional setting and the occurrence setting for the reference group.

Element	Description
Group Choice	An indicator that only one of the items in the group will be included in the message format. This type of group is represented by a dot inside the icon.
Field	A sequence of bytes that have some meaning to an application. For example, the field EMPNAME might contain an employee name. Defines the formatting for the field.
Optional Field	A field that may or may not be included in the message format.
Repeating Field	A field that has one or more occurrences.
Field Reference	An indicator that another instance of the field exists in the data. Reference fields have the same format as the original field, but you can change the optional setting and the occurrence setting for the reference field.
Optional Repeating Field	A field that may or may not be included, but, if included, occurs more than once in the message format.
Comment	An indicator that contains notes about the message format or the data transformed by the message format.
Collapse	A minus sign next to an object indicates that it can be collapsed.
Expand	A plus sign next to an object indicates that it can be expanded to show more objects.

Field Configuration Window

The Field Configuration window defines the fields contained in the message format. These fields are a sequence of bytes that have a meaning in terms of an application. For example, the field EMPNAME means employee name.

Table 23-6 *Format Builder Field Description Properties*

Element	Description
Name	The name of the field. This name must comply with XML element naming conventions.
Optional	Select this check box to make the field optional. Selecting this means that the data for the field may be present in the input. If this option is selected for a file, then you can set the Field is Tagged option from the Field Attributes pane. In addition to it, enter a unique value for each optional field in a group in the Field Is Tagged text box. Multiple groups can use the same tag value, but the tag value for each optional field in a group must be unique
Type	Select the data type of the field from the list of available options. The default type is String. Note: The Field Type that is selected dictates the Field Data Options that appear on the rest of the dialog

Select one of the options in [Table 23-7](#) to indicate how often this field appears in the message format.

Table 23-7 Format Builder Field Occurrence Properties

Element	Description
Once	Select this option if the field appears only once. Note: Unless a field is defined as optional, the field occurs at least once.
Repeat Delimiter	Select this option if the field repeats until the specified delimiter is encountered. Enter the delimiter in the associated field.
Repeat Field	Select this option if the value of the repeat field at runtime is the number of times the field repeats. Select the repeat field name from the list of available options.
Repeat Number	Select this option if the field repeats a specified number of times. Enter the number of recurrences in the associated field.
Unlimited	Select this option if the field repeats an unlimited number of times.

Table 23-8 Format Builder Field Attributes

Element	Description
Field is Tagged	Select this option if the field is a tagged field. Being tagged means that a literal precedes the data, indicating that the data is present. For example: SUP:ACME INC, SUP: is a tag. ACME INC is the field data. If you have selected the Field is Tagged option, enter the tag in the text box to the right of the check box.
Field Default Value	Select this option if the field has a default value. Then, enter the default value in the text box to the right of the check box.
Data Base Type	An indicator that determines the type of characters that make up the data if the field is a date or time field. Whether this field appears is based on the data type selected in the Type field in the Field Description section.
Year Cutoff	If the field is a date field that has a 2-digit year, the year cutoff allows the 2-digit year to be converted to a 4-digit year. If the 2-digit year is greater than or equal to the year cutoff value, a '19' prefix will be added to the year value. Otherwise a '20' prefix will be used. Whether this field appears is based on the data type selected in the Type field in the Field Description section.
Code Page	The code page encodes the String field data. Whether this field appears is based on the data type selected in the Type field in the Field Description section.
Value	The value that appears in a literal field. Whether this field appears is based on the data type selected in the Type field in the Field Description section.

Table 23-9 Format Builder Field Termination Properties Defined by Length

Element	Description
Length Tab	
Value	Enter the number of bytes in the length field if the length field is a variable length. Variable-sized data types can be assigned a fixed length, eliminating the need to use a delimiter to specify the termination point of the field.
String Length in Characters	Select this check box if string is multi-byte encoded to calculate the length in number of characters instead of bytes. By default, the string length is in bytes.
Trim Tab	
Trim Trailing	Select this option to trim data from the trailing edge of the field data. Enter the data to be trimmed in the field next to this option.
Trim Leading	Select this option to trim data from the leading edge of the field data. Enter the data to be trimmed in the field next to this option.
Pad Tab	
Pad Value	Select this option if the data is shorter than the specified length, and enter the necessary value add to the data until it is of correct length. Select Trailing to append padding at the end of a field. Select Leading to append padding at the beginning of a field.

Table 23-10 Format Builder Field Termination Properties Defined by Imbedded Length

Element	Description
Description Tab	
Type	Select the type of the imbedded length from the list of options.
Length	Select the this option to specify the number of bytes, and then enter the number in the corresponding field. Variable-sized data types can have their termination point specified by an imbedded length. An imbedded length precedes the data field and indicates how many bytes the data contains. This option is only available for certain data types selected for the imbedded length.
Delimiter	Select the this option to specify a delimiter, and then enter the delimiter value in the corresponding field. This option is only available for certain data types selected for the imbedded length.
Tag/Length Order Tab	

Table 23-10 (Cont.) Format Builder Field Termination Properties Defined by Imbedded Length

Element	Description
Length Occurs Before Tag Field	Select this option to specify that the length field occurs before the tag field when both are present. The default is tag before length.
Trim Tab	
Trim Trailing	Select this option to trim data from the trailing edge of the field data. Enter the data to be trimmed in the field next to this option.
Trim Leading	Select this option to trim data from the leading edge of the field data. Enter the data to be trimmed in the field next to this option.

Table 23-11 Format Builder Field Termination Properties Defined by Delimiter

Element	Description
Delimiter Tab	Variable-sized data types can have their termination point specified by a delimiter, which you can specify or which can be specified by a field containing the character. A delimiter is a character that marks the end of the field. The field data continues until the delimiter character is encountered.
Ref Fields	Click this button to specify a field that contains the delimiter character that indicates the termination point. On the dialog that appears, select the reference fields and click the right-arrow button. Click OK .
Values	Enter a default delimiter character to use when the delimiter field is not present. You must supply a default value.
Trim Tab	
Trim Trailing	Select this option to trim data from the trailing edge of the field data. Enter the data to be trimmed in the field next to this option.
Trim Leading	Select this option to trim data from the leading edge of the field data. Enter the data to be trimmed in the field next to this option.

The following fields only appear if you select **Literal** in the Type field in the Field Description section.

Table 23-12 *Format Builder Field Properties for Literal Data Types*

Element	Description
Value	An indicator that specifies the literal value. A literal value can be defined as a single value or it can be defined a list of values separated by the literal separator. When the Value is a list of values, the data for the literal field in the binary data will be one of values in the list.
Literal Separator	<p>Supports enumeration of literal values. For literal type Field in MFL definition, a literal separator can be specified when multiple choices of value is needed for the Field.</p> <p>For example, segment terminators that are supported by both EDIFACT and X12 EDI standards are: \r\n, \r, \n, ' , and ~. However, you can use Format Builder to support any other custom terminator. You can append the custom terminator to the existing list of literal values and use comma (,) as literal separator to separate multiple custom values.</p> <p>In the MFL file, you should see the following structure,</p> <pre><FieldFormat name='ISA_Terminator' type='Literal' value='\r\n,\r,\n,~, ' literalSeparator=',' /></pre>

Group Configuration Window

The Group Configuration window defines the groups contained in the message format. Groups are collections of fields, comments, and other groups or references that are related in some way. For example, the fields PAYDATE, HOURS, and RATE could be part of the PAYINFO group.

Table 23-13 *Format Builder Group Description Properties*

Element	Description
Name	The name of the group. This name must comply with XML element naming conventions.
Optional	Select Optional if the group is optional
Choice of Children	Select Choice of Children if only one of the items in the group will be included in the message format.

Select one of the options in [Table 23-14](#) to indicate how often this group appears in the message format.

Table 23-14 *Format Builder Group Occurrence Properties*

Element	Description
Once	<p>Select this option if the group appears only once.</p> <p>Note: Unless a group is defined as optional, it occurs at least once.</p>

Table 23-14 (Cont.) Format Builder Group Occurrence Properties

Element	Description
Repeat Delimiter	Select this option if the group repeats until the specified delimiter is encountered. Enter the delimiter in the associated field.
Repeat Field	Select this option if the value of the repeat field at runtime is the number of times the group repeats. Select the repeat field name from the list of available options.
Repeat Number	Select this option if the group repeats a specified number of times. Enter the number of recurrences in the associated field.
Unlimited	Select this option if the group repeats an unlimited number of times.

Table 23-15 Format Builder Group Attributes

Element	Description
Group is Tagged	Select this option if this is a tagged group. If tagged, a literal precedes the data, indicating that the data is present. If you selected the Group is Tagged option, enter the tag in the text box to the right of the check box.

Table 23-16 Format Builder Group Delimiter Properties

Element	Description
None	Select this option if the group has no delimiter
Delimited	Select this option if the end of the group is marked with a delimiter, which specifies the termination point for the group. A delimiter is a string of characters that marks the end of the group of fields. The group continues until the delimiter characters are encountered. If you select this option, enter the delimiter in the Value field. Note: Normally, groups are not delimited. They are usually parsed by content (the group ends when all child objects have been parsed).
Delimiter Field	Select this option if the group's termination point is specified by a field that contains a delimiter character string. If you select this option, select the Field that contains the delimiter character string, and enter a default delimiter character to use if the above field is not present in the data. This value is required.
Delimiter is Shared	Select this option if the delimiter marks both the end of the group of data, and the end of the last field of the group. The delimiter is shared among the group, and by the last field of the group, to delimit the end of the data.
Delimiter Is Not Optional	Select this option to indicate that the binary data contains the delimiter even if the group is not present.

Format Builder Reference Configuration Window

Use the Reference Configuration window to indicate the existence of another field or group format within the data. Reference fields or groups have the same format as the original field or group, but you can change the optional setting and the occurrence setting for the reference field or group. For example, if you have a "bill to" address and a "ship to" address in your data, you only need to define the address format once. You can create the "bill to" address definition and create a reference for the "ship to" address.

References are given the same name as the original item. For example, the "bill to" address definition and the "ship to" address definition would be named the same.

Table 23-17 *Format Builder Reference Description Properties*

Element	Description
Name	The name signifies the name for the original field or group for which you created this reference. This name must comply with XML element naming conventions.
Optional	Select Optional if the reference field or group is optional.

Table 23-18 *Format Builder Reference Occurrence Properties*

Element	Description
Once	Select this option if the reference appears only once. Note: Unless a reference is defined as optional, it occurs at least once.
Repeat Delimiter	Select this option if the reference repeats until the specified delimiter is encountered. Enter the delimiter in the associated field.
Repeat Field	Select this option if the value of the repeat field at runtime is the number of times the reference repeats. Select the repeat field name from the list of available options.
Repeat Number	Select this option if the reference repeats a specified number of times. Enter the number of recurrences in the associated field.
Unlimited	Select this option if the reference repeats an unlimited number of times.

Using Java Callouts and POJOs

This chapter describes how to extend the capabilities of Service Bus by invoking custom Java code from within pipelines and split-joins. Service Bus pipelines and split-joins each have a Java callout action that allows you to call a Plain Old Java Object (POJO) external to the pipeline or split-join.

This chapter includes the following sections:

- [Introduction to Java Callouts](#)
- [Working with Streaming Content](#)
- [Best Practices for Java Callouts and POJOs](#)

For information about configuring a Java callout to a POJO, see [Adding Java Callout Actions in the Console](#).

Introduction to Java Callouts

The Java callout action lets you access the methods in a Java archive (JAR) file to add processing logic to your pipelines and split-joins. When you configure the callout, you can specify arguments for the method and you can optionally specify a service account for security. The parameters can be mapped to message context variables. Static methods can be accessed from any POJO.

You can also use Java callouts to create Java objects to store in the pipeline and to pass Java objects as parameters to other Java callouts.

Java Callout Usage Guidelines

The scenarios in which you can use Java callouts in Service Bus include the following:

- Performing custom validations, such as validating against a DTD, or doing cross-field semantic validation in Java.
- Performing custom transformations, such as converting a binary document to base64Binary (or vice versa) or using a custom Java transformation class.
- Performing custom authentication and authorizations. Examples include scenarios in which a custom token in a message needs to be authenticated and authorized. However, the authenticated user's identity cannot be propagated by Service Bus to the services or POJOs subsequently invoked by the pipeline or split-join.
- Performing lookups for message enrichment. For example, a file or Java table can be used to look up any piece of data that can enrich a message.
- Accessing binary data. You can use a Java callout to a POJO to sniff the first few bytes of a binary document to deduce the MFL type. The MFL type returned is

used for a subsequent NonXML-to-XML transformation using the MFL Transform action.

- Implementing custom routing rules or rules engines.
- Creating a Java object and storing it in the pipeline.
- Passing a Java object as a parameter to another Java callout.
- Invoking a remote EJB operation or service with a POJO using a JEJB proxy service.

The input and return types for Java callouts are not restricted. For more information about storing and passing Java objects in the pipeline, see [Java Content in the Body Variable](#).

Java Callouts or EJBs

Enterprise JavaBeans (EJBs) also provide a Java exit mechanism. The use of EJBs is recommended over Java callouts in the following cases:

- When you already have an EJB implementation. The JEJB transport lets you invoke EJBs with EJB calls through Service Bus, letting you leverage Service Bus functionality such as message routing, UDDI integration, alerts, per-operation monitoring, reporting, and result caching.
- When you require read access to a JDBC database. Although POJOs can be used for this purpose, EJBs were specifically designed for this and provide better support for managing and connecting to JDBC resources.
- When you require write access to a JDBC database or other J2EE transactional resource. EJBs were specifically designed for transactional business logic and they provide better support for proper handling of failures. However, transaction and security context propagation is supported with POJOs, and the JEJB transport provides error handling in transaction contexts.

For outbound messaging, Oracle recommends that you write a custom transport instead of using POJOs or EJBs.

Working with Streaming Content

You can work with streaming content using Java callouts, both to pass binary-content as an input argument to callout methods and to accept streaming content results from Java callout methods.

Passing Streaming Content to a Java Callout

You can pass binary-content as an input argument to a Java callout method in a streaming fashion. Service Bus handles this by checking the Java type of the input argument. If the argument is of type `javax.activation.DataSource`, the system creates a wrapper `DataSource` object and gets the `InputStream` from the corresponding source by invoking the `Source.getInputStream()` method. You can call this method as many times as you need in your Java callout code.

In addition, the `getContentTypes()` method returns the `application/octet-stream` unless the binary content is a paged MIME attachment, in which case the value of the `Content-Type` header of the corresponding MIME part is used, if present.

Similarly, the `getName()` method returns the string value of the binary-content reference attribute unless the binary content is a paged MIME attachment, in which

case the value of the Content-ID header of the corresponding MIME part is used, if available. The `getOutputStream()` method throws the `UnsupportedOperationException`, as required.

After completing, the result is passed to the Java callout method argument. Note that to properly interpret the binary octets in the input stream, the Java callout method might also require the value of the Content-Transfer-Encoding header (for example, to determine whether the encoding is binary, 7bit, 8bit, and so on). You can pass this parameter as a separate argument, as shown in the following:

```
$attachments/*:attachment[1]/*:Content-Transfer-Encoding/text()
```

Note that if the input argument is not a `DataSource`, Service Bus converts the argument to a `byte[]` array.

Streaming Content Results from a Java Callout

You can get streaming content results from a Java callout method. Service Bus handles this by checking the Java type of the result and then adding the new source to the source repository, setting the appropriate context variable value to the corresponding `ctx:binary-content` XML element.

Note:

To return the contents of a file from a Java callout method, you can use an instance of `javax.activation.FileDataSource`.

Whenever the pipeline or split-joins needs the binary contents of the source, it looks up the `DataSource` object corresponding to the `ctx:binary-content` element in the repository and invokes the `DataSource.getInputStream()` method to retrieve the binary octets.

Note that the `getInputStream()` method might be called multiple times during message processing, for example to accommodate outbound message retries in the transport layer.

Best Practices for Java Callouts and POJOs

POJOs are registered as JAR resources in Service Bus. For information about JAR resources, see [Working with JAR Files](#).

In general, Oracle recommends that the JARs are small and simple; any large bodies of code that a JAR invokes or large frameworks that are used are best included in the system classpath. If you make a change to the system classpath, you must reboot the server.

Oracle recommends that you put dependent and overlapping classes in the same JAR resource. If they are naturally distinct, put them in different JAR resources. Any change to a JAR resource causes all the services that reference it to be redeployed. This can be time consuming for your Service Bus system. The same class can be located in multiple JAR resources without causing conflicts. The JAR files are dynamically class loaded when they are first referenced.

A single POJO can be invoked by one or more services. All the threads in the service invoke the same POJO, so the POJO must be thread safe. A class or method on a POJO can be synchronized, in which case it serializes access across all threads in all of the invoking services. Any finer-grained concurrency (for example, to control access to a

database read results cache and implement stale cache entry handling) must be implemented by the POJO code.

It is generally a bad practice for POJOs to create threads.

Part V

Working with JCA Adapters, Transports, and Bindings

This part provides information and configuration details for all transports provided by Service Bus.

This part contains the following chapters:

- [Using the JCA Transport and JCA Adapters](#)
- [Creating REST Services with Oracle Service Bus](#)
- [Using the DSP Transport](#)
- [Using the EJB Transport](#)
- [Using HTTP and Poller Transports](#)
- [Using the JEJB Transport](#)
- [Using the JMS Transport](#)
- [Using the Local Transport](#)
- [Using the MQ Transport](#)
- [Using the Oracle BPEL Process Manager Transport](#)
- [Using the SB Transport](#)
- [Using the SOA-DIRECT Transport](#)
- [Using the Tuxedo Transport](#)
- [Using the WS Transport](#)

Using the JCA Transport and JCA Adapters

This chapter describes the Oracle JCA adapter framework and provides guidance on using specific adapters with Service Bus. It also describes how to use and configure the Oracle JCA adapter in Service Bus services.

This chapter includes the following sections:

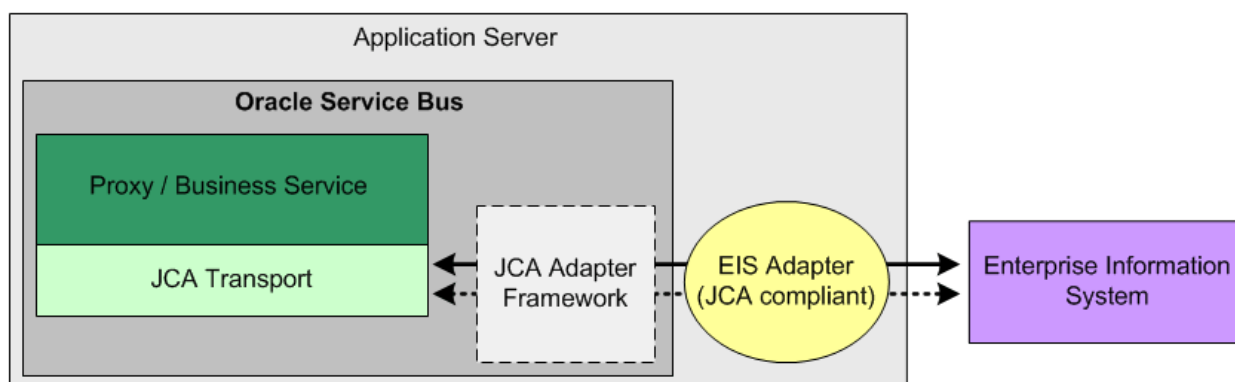
- [Introduction to the JCA Transport](#)
- [JCA Adapter Configuration Recommendations for Service Bus](#)
- [Working with JCA Binding Resources](#)
- [Working with JavaScript Resources](#)
- [JCA Transport Configuration Reference](#)

Introduction to the JCA Transport

Service Bus provides a J2EE Connector Architecture (JCA) transport that interacts with back-end Enterprise Information Systems (EIS), letting these systems participate in the Service Bus integration environment. The JCA transport provides native connectivity between Service Bus and external systems, letting those systems interact in the service bus layer and leverage the capabilities and features of Service Bus. For a list of adapters the JCA transport supports, see [Supported JCA Adapters](#).

In JCA proxy or business services, the JCA transport works in conjunction with a built-in JCA adapter framework and JCA-compliant adapters to interact with EIS systems, as shown in [Figure 25-1](#). Solid arrows signify request, dotted arrows signify response.

Figure 25-1 Service Bus Services Interacting With an EIS



JCA proxy services listen for inbound requests from supported JCA adapters, and JCA business services invoke EIS endpoints through supported adapters.

Supported JCA Adapters

With JCA adapters, you can integrate proxy and business services with technologies like databases, file systems, FTP servers, messaging systems, email, LDAP, Coherence cache, cloud services, and Oracle E-Business Suite. Dragging a JCA adapter into a swimlane of the Service Bus Overview Editor invokes the Adapter Configuration Wizard, where you can define the configuration properties for the adapter. The following sections give an overview of the JCA adapters supported by Service Bus.

AQ Adapter

The AQ adapter lets Service Bus business and proxy services interact with a single consumer or multi-consumer queue. Oracle Streams AQ provides a flexible mechanism for bidirectional, asynchronous communication between participating applications. Advanced queues are an Oracle database feature, and are therefore scalable and reliable. Multiple queues can also service a single application, partitioning messages in a variety of ways and providing another level of scalability through load balancing.

For more information, see "Oracle JCA Adapter for AQ" in *Understanding Technology Adapters*.

Oracle BAM 11g Adapter

The Oracle BAM 11g adapter integrates Service Bus business and proxy services with Oracle BAM Server 11g. Dragging an Oracle BAM adapter into a swimlane of the Service Bus Overview Editor invokes the Adapter Configuration Wizard, where you can configure the adapter properties.

For more information, see Oracle Fusion Middleware Integration with Adapters in *Understanding Technology Adapters*.

Coherence Adapter

The Oracle Coherence adapter integrates Service Bus business services with Oracle Coherence. A Coherence cache is a collection of data objects that serves as an intermediary between the database and client applications. Database data can be loaded into a cache and made available to different applications. A Coherence cache reduces load on the database and provides faster access to database data. Objects in the cache can be either XML or Plain Old Java Objects (POJOs). The Coherence adapter enables you to perform the following operations against a Coherence cache.

- Add an item
- Obtain an item
- Remove an item
- Query for an item

For more information, see "Oracle JCA Adapter for Coherence" in *Understanding Technology Adapters*.

Database Adapter

The database adapter lets Service Bus business and proxy services communicate with Oracle databases or third-party databases through JDBC.

For more information, see "Oracle JCA Adapter for Database" in *Understanding Technology Adapters*.

File Adapter

The file adapter lets Service Bus business and proxy services exchange (read and write) files on local file systems. The file contents can be in both XML and non-XML data formats.

For more information, see "Oracle JCA Adapter for Files/FTP" in *Understanding Technology Adapters*.

FTP Adapter

The FTP adapter lets Service Bus business and proxy services exchange (read and write) files on remote file systems through use of the file transfer protocol (FTP). The file contents can be in both XML and non-XML data formats.

For more information, see "Oracle JCA Adapter for Files/FTP" in *Understanding Technology Adapters*.

JDE World Adapter

The JDE World adapter integrates Service Bus business services with the JDE World System, an ERP (Enterprise Resource Planning) product running on IBM minicomputers. The JDE World Adapter uses a new JDBC-driver based connection to the IBM system and lets Service Bus interact with JDE World system as a typical database.

For more information, see "Oracle JCA Adapter for JDE Edwards World" in *Understanding Technology Adapters* and the [Integration Adapters Documentation page](#) on Oracle Technology Network.

JMS Adapter

The JMS adapter lets Service Bus business and proxy services interact with a Java Messaging System (JMS). The JMS architecture uses one client interface to many messaging servers. The JMS model has two messaging domains:

- Point-to-point: Messages are exchanged through a queue and each message is delivered to only one receiver.
- Publish-subscribe: Messages are sent to a topic and can be read by many subscribed clients.

For more information, see "Oracle JCA Adapter for JMS" in *Understanding Technology Adapters*.

LDAP Adapter

The LDAP adapter lets Service Bus business and proxy services interact with an LDAP directory. The LDAP adapter defines both asynchronous and synchronous interfaces to send requests to and receive responses from LDAP directory servers. The LDAP adapter enables processes to search, compare, and modify LDAP directories using the LDAP protocol.

For more information, see "Oracle JCA Adapter for LDAP" in *Understanding Technology Adapters*.

MQ Series Adapter

The MQ Series adapter provides message exchange capabilities between Service Bus business and proxy services and the WebSphere MQ queuing systems. The Messaging and Queuing Series (MQ Series) is a set of products and standards developed by IBM. The MQ Series provides a queuing infrastructure that provides guaranteed message delivery, security, and priority-based messaging.

For more information, see "Oracle JCA Adapter for MQ Series" in *Understanding Technology Adapters*.

MSMQ Adapter

The MSMQ adapter provides message exchange capabilities between Service Bus business and proxy services and Microsoft Message Queueing (MSMQ). MSMQ is a message infrastructure and a development platform for creating distributed, loosely-coupled messaging applications for the Microsoft Windows operating system.

For more information, see "Oracle JCA Adapter for Microsoft Messaging Queue" in *Understanding Technology Adapters*.

Oracle E-Business Suite Adapter

The Oracle E-Business Suite adapter lets Service Bus business and proxy services interact with Oracle E-Business Suite. The adapter supports all modules of Oracle E-Business Suite in Release 12 and Release 11i, including selecting custom integration interface types based on the version of Oracle E-Business Suite.

For more information, see *Oracle Fusion Middleware Oracle E-Business Suite Adapter User's Guide*.

Salesforce Cloud Adapter

The Salesforce Cloud Adapter lets Service Bus business services interact with Salesforce services. It enables integration with Enterprise, Unlimited, or Developer Editions of Salesforce, and lets you connect a variety of systems to Salesforce, leveraging the SOAP API of Salesforce.

For more information, see the [Integration Adapters Documentation page](#) on Oracle Technology Network.

SAP Adapter

The SAP adapter is a packaged-application adapter that lets Service Bus business and proxy services integrate with SAP applications.

For more information, see "Packaged Application Adapters" in *Understanding Technology Adapters* and the [Integration Adapters Documentation page](#) on Oracle Technology Network.

Socket Adapter

The socket adapter lets you create a client or a server socket and establish a connection. With this adapter, you can model standard or nonstandard protocols for communication over TCP/IP sockets. The transported data can be text or binary in format.

For more information, see "Oracle JCA Adapter for Sockets" in *Understanding Technology Adapters*.

Third Party Adapter

The third party adapter lets Service Bus business and proxy services integrate third-party adapters such as PeopleSoft, SAP, and others. These third-party adapters produce artifacts (WSDL and JCA files) that can configure a JCA adapter.

For more information, see Packaged-Application Adapters in *Understanding Technology Adapters*.

User Messaging Service Adapter

The Oracle User Messaging Service supports messaging channels such as email, secure messaging service (SMS), instant messaging, and voice. The Oracle User Messaging Service provides a messaging proxy between Service Bus business and proxy services and the external world. The Oracle User Messaging Service provides two-way messaging (inbound and outbound).

For more information, see "Oracle JCA Adapter for UMS" in *Understanding Technology Adapters*.

Oracle JCA Adapter Limitations

Following are limitations when using some JCA adapters with Service Bus.

- [Limitations that Apply to All JCA Adapters](#)
- [Oracle JCA Adapters for Files/FTP Limitations](#)

Limitations that Apply to All JCA Adapters

The JCA transport does not support the "singleton" endpoint property for an active/passive topology.

Oracle JCA Adapters for Files/FTP Limitations

For Oracle JCA Adapter for Files/FTP, the JCA transport does not support the following: pre- and post-processing of files, using a re-entrant valve for processing ZIP files, and file chunked read.

JCA Adapter Framework

The JCA transport uses the Service Bus JCA adapter framework to interact with JCA-compliant adapters that in turn provide connectivity to external EIS services. The JCA adapter framework abstracts the complexity of interacting with those adapters, letting you focus on proxy and business service development.

For inbound interactions, the JCA proxy service registers a listener with the associated JCA adapter endpoint. When an event occurs in the EIS system where a message is sent to the JCA proxy, the JCA adapter framework invokes the proxy service with a request-only or request-response pattern. On outbound interactions, when a client invokes an EIS service through Service Bus, the JCA business service invokes the JCA adapter endpoint through the JCA adapter framework.

No configuration of the JCA adapter framework is necessary. It is deployed and functions automatically as you create JCA proxy and business services and deploy your adapters.

JCA Transport Messaging

The JCA transport supports request-only and synchronous request/response messaging patterns using SOAP 1.1. The JCA transport is transactional. If a JCA proxy service is invoked in an EIS transaction, or if a JCA business service is invoked in a transaction, the transport propagates the transaction.

Security for JCA Transports

When a service uses the JCA transport, a JNDI service account is used during both endpoint validation and runtime. During endpoint validation, if a static service account is associated with the endpoint, JNDI lookup against the adapter connection factory is performed using the subject in the static service account. If the service does not use a service account, an anonymous subject is used.

At runtime, a JCA proxy service supports only static service accounts. If a static service account is configured on the service, the subject in the service account is used to perform the JNDI lookup against the adapter connection factory. Otherwise, an anonymous subject is used. JCA business services support static, pass through, and mapping service accounts at runtime. If a service account is configured on a JCA business service, the JNDI lookup against the adapter connection factory and the subsequent outbound invocation is performed using the subject in the service account. If the service does not use a service account, an anonymous subject is used.

Oracle adapters that connect to an EIS database (such as Oracle Adapters for Database, AQ, and Oracle E-Business Suite) connect using the JDBC datasource associated with the adapter-managed connection factory. Other Oracle adapters, such as the Oracle JCA Adapter for Files, support Oracle WebLogic Server container-managed and application-managed sign-on for outbound connections. For more information, see "Securing Enterprise Information System Credentials" in *Understanding Technology Adapters*.

The JCA transport can also be used with Oracle Web Services Manager (OWSM) policies. For more information, see "Which OWSM Policies Are Supported for JCA Adapters?" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Proxy Services

On inbound requests the JCA adapter framework, which invokes a JCA proxy service, always invokes the proxy service as anonymous.

Business Services

Depending on which type of JNDI service account is used in a JCA business service, the outbound service endpoint is invoked with a subject in the following ways:

- No service account is used. Instead, an anonymous subject is used to invoke outbound JCA endpoint.
- A pass-through service account is used. The subject associated with the inbound request is used to invoke the outbound JCA endpoint.
- A static service account is used. The subject associated with the static user name and password in the service account is used to invoke the outbound JCA endpoint.

- A mapping service account is used. The subject associated with the mapped user name and password in the service account is used to invoke the outbound JCA endpoint.

Logging

The JCA transport logs messages using the Oracle WebLogic Server NonCatalog logger. JCA adapter framework logs are redirected to the Oracle WebLogic Server log.

Debug log records generated by the JCA transport, JCA adapter framework, and JCA adapters (except the Oracle BAM Adapter) are redirected to the Oracle WebLogic Server log only if the `oracle.osb.debug.jca-framework-adapter` logger is configured for debugging. JCA framework and adapter messages are logged with the tag `JCA_FRAMEWORK_AND_ADAPTER`. For more information, see "Configuring and Monitoring Log Files" in *Administering Oracle Service Bus*.

Oracle BAM Adapter Logging

To enable debug-level logging for Oracle BAM Adapter batch processing using RMI- and SOAP-based connection pools, add the following logger entries to `domain_home/config/fmwconfig/servers/server_name/logging.xml`:

- For RMI/EJB – `<logger name="oracle.bam.adc.api.batching.BatchProcessor" level="FINER"/>`
- For SOAP – `<logger name="oracle.bam.adapter.adc.soap.SOAPBatchProcessor" level="FINER"/>`

JCA Transport Error Handling

When the JCA framework and adapters throw an exception from an EIS, a JCA business service propagates that exception to the SOAP fault inside a `jca-runtime-fault-detail` element. This fault structure is defined in the JCA transport schema.

You can access the EIS fault details in one or more of the following sub-elements:

- `eis-error-code` (string): Captures the EIS error code propagated by the JCA framework and adapter, if available.
- `eis-error-message` (string): Captures the EIS error message propagated by the JCA framework and adapter, if available.
- `exception` (string): Captures the JCA framework and adapter stack trace.

Application errors cannot be retried with the JCA transport, and the transport has no control of this option. Set this option to **No**.

URI Rewriting with JCA Transports

The JCA transport supports URI rewriting in a pipeline.

JCA Transport Message Encoding

For inbound requests and outbound responses, the JCA adapter framework sends messages to JCA proxy services with UTF-8 encoding.

Rejected Messages

The JCA adapter framework automatically logs rejected messages (messages with data errors) to a `/domain/jca/read/rejectedMessages` directory for each adapter. For more information, see "Handling Rejected Messages" in *Understanding Technology Adapters*.

JCA Adapter Configuration Recommendations for Service Bus

The JCA transport is compatible with all JCA adapters, each of which can be configured through a set of endpoint properties, activation or interaction specification properties, or connection properties. Any runtime or binding properties defined for the JCA adapter itself are not propagated to the transport configuration in the proxy or business service. You need to manually configure those properties. For more information, see [JCA Transport Endpoint Properties](#).

The following topics provide configuration recommendations specific to Service Bus.

Configuring the JCA Adapter Connections

The Oracle JCA adapters are deployed automatically in a Service Bus domain. You must manually install and deploy other supported third-party resource adapters in the Oracle WebLogic Server Administration Console. In order to use the deployed Oracle Adapters, you must create a data source and connection pool for each adapter in Oracle WebLogic Server. For information and instructions on creating the required data source and connection pool, see "Adding an Adapter Connection Factory" in *Understanding Technology Adapters*. Additional information about data sources is provided in that guide for each type of adapter.

Configuring JCA Adapters that Poll a Database

By default, the Oracle adapters that poll a database use one thread to poll the database (`NumberOfThreads=1` property in the activation spec). Because the adapter never releases that thread, which is by design, you may see a stuck thread stack trace in the server log. If you set the `NumberOfThreads` to more than one, you may see stack traces for all of those threads. You can ignore stuck thread stack traces.

To prevent stuck thread stack traces from appearing for the adapter, configure a Work Manager that includes the following setting:

```
<ignore-stuck-threads>true</ignore-stuck-threads>
```

In the JCA proxy service, configure the transport's Dispatch Policy to use the Work Manager you created.

For information about Work Managers, see the following topics:

- [Using Work Managers with Service Bus](#)
- "Using Work Managers to Optimize Scheduled Work" in *Administering Server Environments for Oracle WebLogic Server*

Configuring the Oracle JCA Adapter for Database

Only one proxy service can poll a specific Oracle Database table. For the Oracle JCA Adapter for Database data source, set the following options:

Option	Setting
Initial Capacity	0
Test Connections on Reserve	selected
Test Frequency	5
Test Table Name	SQL SELECT 1 FROM DUAL
Seconds to Trust an Idle Pool Connection	0
Connection Creation Retry Frequency	10

Configuring the Oracle JCA Adapter for AQ

For the Oracle JCA Adapter for AQ data source, select the **Test Connections on Reserve** option.

Configuring the Oracle JCA Adapter for Coherence

If the objects in the Coherence cache are Plain Old Java Objects (POJOs), the adapter configuration includes specifying the archive resource that stores the POJO JAR files. You can create this resource in the Service Bus project before you can create the Coherence adapter and its associated business service. The configuration wizard also gives you the option to search the file system for a JAR file to upload, and lets you import the file and create the archive resource in the Service Bus project. For more information about creating archive resources, see [Working with JAR Files](#).

Configuring the Salesforce Cloud Adapter

In addition to a dependency on the standard WSDL file, the Salesforce adapter has a dependency on an *enterprise* WSDL file, which you generate directly from Salesforce. This file is associated with the JCA resource in addition to the standard JCA WSDL file, and is referenced in the proxy or business service's non-managed connection property as the value for the `targetWSDLURL` property.

Working with JCA Binding Resources

Using Oracle JDeveloper and other supported JCA development environments, you can create and configure a JCA adapter and create the JCA adapter resources (.jca files and WSDL files) used for the back-end integration.

In the Oracle Service Bus Console, you cannot create and configure a JCA adapter, but you can upload an adapter you created in JDeveloper into a JCA binding resource. Once you create the JCA binding resource, you can generate proxy and business services that use the Service Bus JCA transport to communicate with the EIS applications through the JCA adapter.

Once you create a JCA adapter in JDeveloper, you can import it into a JCA binding resource in the Oracle Service Bus Console if that is your development environment.

How to Create a JCA Adapter in JDeveloper

The Service Bus Overview Editor provides a simple drag-and-drop method of creating JCA adapters in Service Bus projects in JDeveloper. You can drag any supported JCA adapter from the Component window onto the canvas, which launches the configuration wizard for that adapter. Completing the wizard creates the JCA file and proxy or business service, and adds all the required files to the Service Bus project.

Note:

When you create a business or proxy service in JDeveloper by creating a JCA adapter directly in the overview, the JCA transport is still the intermediary between the service and the adapter.

For more information, see [Adding Service Bus Components](#) in this guide and Introduction to Oracle JCA Adapters in *Understanding Technology Adapters*

How to Import JCA Adapters in the Oracle Service Bus Console

The easiest way to create JCA binding resources in the Oracle Service Bus Console is by bulk importing the JCA resources you created in JDeveloper, including the JCA file, associated WSDL file, and TopLink mapping file). With a bulk import, Service Bus automatically creates JCA bindings out of JCA files, WSDL resources out of WSDL files, and XML document resources out of mapping files. Service Bus maintains the dependencies among the files. For more information on importing, see [Importing and Exporting Resources and Configurations](#).

Once you import your JCA resources, you can generate proxy or business services out of them, as described in [How to Generate a Proxy Service from a JCA Binding Resource](#) and [How to Generate a Business Service from a JCA Binding Resource](#).

How to Create a JCA Binding Resource in the Oracle Service Bus Console

If you do not bulk import your JCA resources into the Oracle Service Bus Console, you can manually create JCA binding resources and upload the JCA file to the console. A JCA file and WSDL file are required to create a JCA proxy or business service in the Oracle Service Bus Console. This section describes the process for creating a JCA binding resource. For information about creating a WSDL resource, see [How to Create a WSDL Resource in the Console](#).

To create a JCA binding resource in the console:

1. In Oracle JDeveloper, create and configure a JCA adapter. You can do this directly in a Service Bus project using the Service Bus Overview Editor.

For more information, see [Adding Service Bus Components](#).

2. In the Oracle Service Bus Console, upload any JCA dependencies for the JCA adapter you created in Oracle JDeveloper, such as the WSDL and XML resources.
3. In the Project Navigator, right-click the project or folder to contain the new JCA binding resource, point to **Create**, and select **JCA Binding**.
4. Do one of the following:

- To upload a JCA file when you create the resource, click **Browse** next to the **File Upload** field and then navigate to and select the JCA file to use.

The **Resource Name** field is automatically populated with the file name minus the file extension. You can change this name and optionally enter a description. JCA binding names must be unique.

- To create a JCA binding resource and upload the JCA file at a later time, enter a name and optional description for the resource.

5. Click **Create**.

The JCA Binding Definition Editor appears.

6. In the JCA Binding Definition Editor toolbar, click **Save**.

7. To end the session and deploy the configuration to the runtime, click **Activate**.

After you add JCA binding resources and its dependencies, you can generate proxy and business services from JCA bindings. For more information, see [How to Generate a Proxy Service from a JCA Binding Resource](#) and [How to Generate a Business Service from a JCA Binding Resource](#).

How to Edit JCA Binding Resources in the Console

If you are using the Oracle Service Bus Console, use the following procedure to edit JCA bindings.

For information about modifying JCA adapters in JDeveloper, see [How to Edit Components from the Service Bus Overview Editor](#).

To edit a JCA binding in the console:

1. In the Project Navigator, expand the project and folders containing the JCA binding to edit.
2. Right-click the JCA binding name, and select **Open**.
3. To search for and select a new WSDL file to reference, click the **Choose a WSDL** icon in the WSDL Dependency section.
4. To modify the contents of the JCA file or upload a new JCA file, do the following:
 - a. Click **Edit Source** in the toolbar.
The Edit Source dialog appears.
 - b. To browse to and select a new JCA file to upload, click **Browse**.
 - c. To modify the contents of the file, update the code directly in the Contents section of the dialog.
 - d. Click **Save**.
5. In the JCA Binding Definition Editor toolbar, click **Save**.
6. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Delete JCA Binding Resources

If any business service or proxy service is based on the JCA binding resource, remove the JCA binding resource from the business service or proxy service before deleting the JCA binding resource itself. In the Oracle Service Bus Console, open the JCA binding in the JCA Binding Definition Editor and click the **References** icon in the upper right to find out if it has any references. In JDeveloper, right-click the JCA file and select **Explore Dependencies**.

To delete a JCA binding resource:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the JCA binding to delete.
2. Right-click the name of the JCA binding, and select **Delete**.

The JCA binding resource is deleted. A Deletion Warning icon appears when other resources reference this resource. You can delete the resource with a warning confirmation. This might result in conflicts due to unresolved references to the deleted resource.

3. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Using Custom JCA Adapters

You can create and use custom JCA adapters in Service Bus projects. You can only create a custom adapter in JDeveloper. Once you create the adapter, you can either use it in JDeveloper or you can import the adapter's associated file into the Oracle Service Bus Console as described in [How to Create a JCA Binding Resource in the Oracle Service Bus Console](#). You can then use the adapter in conjunction with the JCA transport like you would any other JCA adapter.

Note:

Unlike some supported Oracle SOA Suite adapters that include dependency resources such as TopLink XML mapping files or XSLT files, Service Bus does not support the use of dependency resources with custom adapters.

About the Custom Adapter Registration File

All custom adapters need to be registered in the `OSBSupportedAdapters.xml` file, located in the Service Bus home directory in `OSB_ORACLE_HOME/config/adapter`. This file simply lists the names of all the custom JCA adapters for Service Bus. If you have not previously registered a custom JCA adapter, you may need to create the registration file. Below is a sample file you can use to create your own registration file.

```
<jca:osb-supported-adapters
  xmlns:jca="http://www.bea.com/wli/sb/transports/jca">
  <jca:adapter-type>CUSTOM_ADAPTER</jca:adapter-type>
  <jca:adapter-type>SAMPLE_ADAPTER</jca:adapter-type>
  <jca:adapter-type>MY_ADAPTER</jca:adapter-type>
</jca:osb-supported-adapters>
```


The value for the `adapter-type` element comes from the adapter JCA file. It is the same as the value of the `adapter` attribute in the `adapter-config` element with the following changes:

- Convert all letters to capital letters.
- Convert all single spaces to an underscore; for example, `My Adapter` would be `MY_ADAPTER`.
- Convert consecutive spaces to a single underscore; for example, `My Adapter` would also be `MY_ADAPTER`.

For example, given the following in the adapter JCA file:

```
<adapter-config name="custom-adapter-endpoint" adapter="My Adapter"
  wsdlLocation="custom-adapter-endpoint.wsdl"
  xmlns="http://platform.integration.oracle/blocks/adapter/fw/metadata">
  . . .
</adapter-config>
```

The entry in `OSBSupportedAdapters.xml` would be:

```
<jca:adapter-type>MY_ADAPTER</jca:adapter-type>
```

Registering and Using a Custom JCA Adapter with Service Bus

To register and use a custom JCA adapter with Service Bus:

1. In JDeveloper, create a custom JCA adapter.

You can do this using the Service Bus Overview Editor, as described in [Adding Service Bus Components](#). For more information about configuring a custom adapter, see *Creating a Custom Adapter* in *Understanding Technology Adapters*.

2. Do one of the following, using the sample registration file in [About the Custom Adapter Registration File](#) as a basis:
 - If the custom adapter registration file, `OSB_ORACLE_HOME/config/adapter/OSBSupportedAdapters.xml`, already exists, add your custom adapter to the list of supported adapter types.
 - If the custom adapter registration file does not exist, create the file. Replace the sample adapter type names with the actual names from your adapter JCA file.
3. If you are using the Oracle Service Bus Console for development, import the new custom adapter JCA resources into the console. For information and instructions, see [Working with JCA Binding Resources](#).
4. On the JCA transport configuration page of the business or proxy service, configure the endpoint properties, dynamic endpoint properties, and activation or interaction spec properties. For information about the properties that are available for each adapter, see *Oracle JCA Adapter Properties* in *Understanding Technology Adapters*.

Note:

- You can use the JCA file to generate a proxy or business service, or you can configure JCA proxy or business services to use your custom adapter. For information on generating a service from a JCA file, see [Creating and Configuring Business Services](#) or [Creating and Configuring Proxy Services](#).
 - The JCA transport supports normalized message properties for custom adapters, as described in [JCA Transport Headers and Normalized Message Properties](#).
-

Working with JavaScript Resources

The JCA Adapter for Sockets provides support for handling a handshake using custom JavaScript in addition to using XSLT and custom Java code. JavaScript is a light-weight scripting language used to add interactive features to HTML pages.

How to Create JavaScript Resources

If you are working in JDeveloper, you can add JavaScript to your projects using standard JDeveloper features. For more information, see "Developing Applications with Script Languages" in *Developing Applications with Oracle JDeveloper*.

In the Oracle Service Bus Console, you create a JavaScript resource and upload an existing JavaScript file into the resource.

To create a JavaScript resource in the Oracle Service Bus Console:

1. In Oracle JDeveloper or other editor, create and define the JavaScript file.
2. In the Project Navigator of the Oracle Service Bus Console, right-click the project or folder to contain the new JavaScript resource, point to **Create**, and select **JavaScript**.
3. Do one of the following:
 - To upload a JavaScript file when you create the resource, click **Browse** next to the **File Upload** field and then navigate to and select the file to use.

The **Resource Name** field is automatically populated with the file name minus the file extension. You can change this name and optionally enter a description. JavaScript resource names must be unique.
 - To create a JavaScript resource and upload the actual file at a later time, enter a name and optional description for the resource.
4. Click **Create**.

The JavaScript Definition Editor appears.
5. In the JavaScript Definition Editor toolbar, click **Save**.
6. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Edit JavaScript Resources

If you are using JDeveloper, you can edit the JavaScript file using the standard editors in the IDE. If you are using the Oracle Service Bus Console, use the following procedure to edit JavaScript resources.

To edit a JavaScript resource in the Oracle Service Bus Console:

1. In the Project Navigator, expand the project and folders containing the JavaScript resource to edit.
2. Right-click the JavaScript resource name, and select **Open**.
3. To modify the contents of the JavaScript resource or upload a new JavaScript resource, do the following:
 - a. Click **Edit Source** in the toolbar.
The Edit Source dialog appears.
 - b. To browse to and select a new JCA file to upload, click **Browse**.
 - c. To modify the contents of the file, update the code directly in the Contents section of the dialog.
 - d. Click **Save**.
4. In the JavaScript Definition Editor toolbar, click **Save**.
5. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Delete JavaScript Resources

If the JavaScript resource is associated with a JCA adapter, remove the resource from the adapter before deleting the JavaScript resource itself. In the Oracle Service Bus Console, open the JavaScript resource in the JavaScript Definition Editor and click the **References** icon in the upper right to find out if it has any references. In JDeveloper, right-click the JavaScript file and select **Explore Dependencies**.

To delete a JavaScript resource:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the JavaScript resource to delete.
2. Right-click the name of the resource to delete, and select **Delete**.

The JavaScript resource is deleted. A Deletion Warning icon appears when other resources reference this resource. You can delete the resource with a warning confirmation. This might result in conflicts due to unresolved references to the deleted resource.
3. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

JCA Transport Configuration Reference

This section provides descriptions for JCA transport-specific configuration options.

- [JCA Transport Endpoint URIs](#)

- [JCA Transport Headers and Normalized Message Properties](#)
- [JCA Transport Endpoint Properties](#)
- [JCA Transport Environment Variables](#)
- [Configuring Proxy and Business Services to Use the JCA Transport](#)
- [Proxy Service Operation Configuration](#)

Note:

Note that runtime and binding properties defined in the JCA adapter file are not propagated to the proxy or business service configuration. You need to configure those properties manually by modifying the endpoint, spec, and connection properties described in the following sections.

For descriptions of general business and proxy service configuration options, see the following topics:

- [Creating and Configuring Proxy Services](#)
 - [Creating and Configuring Business Services](#)
-

JCA Transport Endpoint URIs

Use the following endpoint URI format for JCA services:

```
jca://resource_adapter_jndi
```

where `resource_adapter_jndi` is the value of the `location` attribute in the `connection-factory` element in the JCA file. This value matches the adapter connection factory JNDI.

Endpoint Redeployment

JCA service endpoints are dependent on WSDL files, service accounts, JCA resources, XML schemas, and XML resources (TopLink mapping files). When any of those resource types is updated and saved for a service, the JCA service endpoint is automatically deleted, recreated, and redeployed. For a JCA proxy service, a new adapter listener is also initialized to listen for inbound requests.

JCA endpoint redeployment has a potential impact on services at runtime, depending on whether or not you select the **Always use configuration from JCA file** option for a service as described in [Table 25-2](#). For example, after a JCA endpoint is redeployed:

- If **Always use configuration from JCA file** is selected, the updated JCA file is used to connect and interact with the resource adapter at runtime.
- If **Always use configuration from JCA file** is not selected, existing service configuration overrides are used to interact with the resource adapter at runtime instead of corresponding JCA file properties that may have been updated.

JCA Transport Headers and Normalized Message Properties

Oracle JCA adapters transmit header information through normalized message properties (with the exception of the Oracle JCA Adapter for AQ's payload header). Service Bus supports the JCA adapter normalized message properties. On inbound

messages from Oracle JCA adapters, the Service Bus JCA transport automatically maps normalized message properties to transport headers as key value pairs. On outbound messages to JCA adapters, the JCA transport automatically converts transport headers to normalized message properties.

You can select each normalized message property from a list of options when creating an XQuery expression.

Note:

You cannot map normalized message properties to SOAP message headers in Service Bus.

The following sections list the predefined transport headers in Service Bus that support normalized message properties. Service Bus maps all other normalized message properties to user-defined transport headers.

Service Bus transport headers are predefined in the `$inbound` and `$outbound` request variables. For lists and descriptions of the normalized message properties, see the following topics in *Understanding Technology Adapters*:

- JCA Properties for Oracle AQ Adapter: Normalized Properties
- Coherence Adapter Normalized Properties
- JCA Properties for Oracle Database Adapter: Normalized Properties
- JCA Properties for Oracle File Adapter: Normalized Properties
- JCA Properties for Oracle FTP Adapter: Normalized Properties
- JCA Properties for Oracle JMS Adapter: Normalized Properties
- JCA Properties for Oracle MQ Adapter: Normalized Properties and RFH Version 2 (RFH2) Headers
- MSMQ Adapter Normalized Properties
- UMS Adapter Message Headers

JCA Transport Endpoint Properties

The JCA transport supports a number of service endpoint properties you can set in the JCA proxy or business service configuration, as described in [Configuring Proxy and Business Services to Use the JCA Transport](#). For more information about endpoint properties, see "Oracle JCA Adapter Properties" in *Understanding Technology Adapters*.

Caution:

Even though these properties might be defined in the JCA adapter file, they also need to be defined for the business or proxy service.

Standard Endpoint Properties

A standard set of endpoint properties applies to most JCA adapters. When you configure a JCA-based proxy or business service, you can select these properties from

a list of options and configure their value. [Table 25-1](#) lists and describes the endpoint properties you can select for a JCA transport.

Table 25-1 JCA Transport Endpoint Properties

Property Name	Description
<code>jca.retry.count</code>	Indicates the maximum number of retries before rejection.
<code>jca.retry.interval</code>	Indicates the time interval between retries (measured in seconds).
<code>jca.retry.backoff</code>	Indicates the retry interval growth factor (positive integer).
<code>jca.retry.maxInterval</code>	Indicates the maximum value of retry interval; that is, a cap if backoff is greater than 1.
<code>jca.retry.maxPeriod</code>	Indicates the maximum total retry period for outbound JCA adapters. Retries do not occur longer than the value specified in this property.
<code>activationInstances</code>	Increases the number of polling (worker) threads for an inbound JCA resource adapter. It is only meant to help increase concurrency for adapters that do not natively support multithreading. Since most of the adapters included with Oracle Fusion Middleware natively support multithreading, this setting is mostly useful to third party (custom) JCA adapters, which do not natively support multithreading. Set this property to the number of threads required for a particular JCA adapter.
<code>payloadSizeThreshold</code>	Specifies the payload size threshold in the adapter layer. The messages that have sizes beyond the configured threshold limit are rejected. If this property is not configured, it does not impose any restriction on the size of messages.
<code>rejectUncorrelatedMessages</code>	Specifies whether to reject messages that cannot be correlated. When native correlation is used to correlate an inbound asynchronous message with a previous outbound message, the JCA framework normally tries to post the message to the service, whether the inbound message can be correlated or not. By setting this property to <code>true</code> , the JCA framework rejects a message that cannot be correlated (when native correlation is active).
<code>UseWorkManager</code>	Specifies a custom work manager to use for starting polling (worker) threads instead of using the standard work manager used by default. Use this property to configure an inbound JCA endpoint to use a specific WebLogic Work Manager. The work manager is only used to start the JCA service for which this property is defined. The value for this property is just the name of the work manager. If the JNDI lookup fails, the default Service Bus work manager is used.

Dynamic Endpoint Properties

Most JCA adapters support a large set of endpoint properties that are unique to the type of system being integrated. When you configure a JCA-based proxy or business service, you can select these properties from a list of options and configure their value.

For lists and descriptions of the normalized message properties, see the following topics in *Understanding Technology Adapters*:

- Binding Properties for all Oracle JCA Adapters
- Binding Properties for Oracle AQ Adapter
- Oracle JCA Adapter for Database
- Binding Properties for Oracle File and FTP Adapters
- Binding Properties Specific to Oracle FTP Adapter
- Binding Properties for Oracle JMS Adapter
- Binding Properties for Oracle MQ Series Adapter
- MSMQ Adapter Binding Properties
- UMS Adapter Message Headers

Activation and Interaction Specification Properties

Activation specification properties are specific to proxy services and interaction specification properties are specific to business services in Service Bus. You cannot add or remove these properties; you can only change their values. The properties that appear on the Transport configuration page vary depending on how you defined the adapter properties when creating the adapter in the Adapter Configuration Wizard.

Updating these properties require the adapter endpoint to be recycled. These types of properties have various dependencies on each other. For lists and descriptions of the activation and interaction specification properties, see the following topics in *Understanding Technology Adapters*:

- Properties for all Oracle JCA Adapters
- JCA Properties for Oracle AQ Adapter
- Coherence Adapter JCA Properties
- Oracle JCA Adapter for Database
- JCA Properties for Oracle File and FTP Adapters
- JCA Properties Specific to Oracle FTP Adapter
- JCA Properties for Oracle JMS Adapter
- LDAP Adapter Connection Properties
- JCA Properties for Oracle MQ Series Adapter
- MSMQ Adapter JCA Properties
- JCA Properties for Oracle Socket Adapter
- UMS Adapter Activation Spec Properties
- UMS Adapter Interaction Specification Properties

JCA Transport Environment Variables

The JCA transport declares the following environment variable values, which can be maintained when moving a Service Bus configuration among different deployment environments.

For descriptions of these values, see [Configuring Proxy and Business Services to Use the JCA Transport](#). For more information about environment variables, see "Customizing Oracle Service Bus Environments" in *Administering Oracle Service Bus*.

- Work Manager
- JCA Always Use JCA File Flag
- JCA Connection Mode
- JCA Overwrite Connection Authentication Flag

Configuring Proxy and Business Services to Use the JCA Transport

[Table 25-2](#) describes the options available on the JCA transport configuration page in either JDeveloper or the Oracle Service Bus Console.

Table 25-2 JCA Transport Properties

Option	Description
JCA File	Click Browse to select a JCA resource. The JCA resource defines different aspects of the service, such as details about the adapter used, a binding to the WSDL and TopLink mapping files, and the activation/interaction spec properties required by the service. Once you select a valid JCA resource, the remaining transport configuration fields become available.
Adapter Name	Displays the name of the adapter that the JCA service will use.
Adapter Type	Displays the adapter type.
Dispatch Policy	Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists. For information about Work Managers, see the following topics: <ul style="list-style-type: none"> • Using Work Managers with Service Bus • Using Work Managers to Optimize Scheduled Work in Administering Server Environments for Oracle WebLogic Server

Table 25-2 (Cont.) JCA Transport Properties

Option	Description
JNDI Service Account	<p>Click Browse and select a JNDI service account, which is used for JNDI context security to access the EIS adapter managed connection factory. If no service account is specified, an anonymous subject is used.</p> <p>For JCA business services, there is no restriction on the type of JNDI service account that can be configured, such as static or pass-through, but the runtime must be able to access a user name and password. JCA proxy services can use only static JNDI service accounts.</p> <p>For more information on JNDI service accounts, see Security for JCA Transports.</p>
EndPoint Properties	<p>Click the Add icon to select endpoint properties from the available list and assign a value to each property. This field lets you assign values to endpoint properties, such as the number of retries for the type of adapter the service uses.</p> <p>For links to information about supported endpoint properties, see JCA Transport Endpoint Properties.</p>
Dynamic EndPoint Properties	<p>Click the Add icon to define dynamic properties for the endpoint and assign a value to each property. Enter a name/value pair for each dynamic endpoint property you want to provide. The endpoint property key matches the query parameter name.</p> <p>This option lets you pass request parameters to JCA-compliant services. For example, you can use a dynamic endpoint property to pass database query parameters to the Oracle JCA Adapter for Database.</p> <p>For more information on querying with parameters, see Oracle JCA Adapter for Database in <i>Understanding Technology Adapters</i>. For links to information about supported endpoint properties, see JCA Transport Endpoint Properties.</p>
Always use configuration from JCA file	<p>Select this option to specify whether Activation Spec Properties (proxy services) and Interaction Spec Properties (business services) are always used from the JCA file.</p> <p>If you select this option (default), the JCA transport interacts with the JCA framework using the activation or interaction spec properties in the JCA file. If you clear this option, you can override the activation and interaction spec Properties.</p> <p>For the redeployment impact of using this option, see Endpoint Redeployment.</p>
Operation Name	<p>Displays a read-only name of the selected WSDL operation. Each operation can have its own activation or interaction spec properties.</p>

Table 25-2 (Cont.) JCA Transport Properties

Option	Description
Activation/Interaction Spec Properties	<p>For business services, this field displays the outbound interaction spec properties for the JCA inbound operation shown in the Operation Name field. For proxy services, this field displays the activation spec properties for the JCA inbound operation shown in the Operation Name field. You can override the activation or interaction spec properties if you clear Always use configuration from JCA file.</p> <p>Note: For Oracle Adapter Suite adapters, activation/interaction spec properties are read-only. The Oracle Adapter Suite adapters store their own configurations, which you must change in the Oracle Adapter Suite management tools.</p>
Connection Mode	<p>Select from one of the following options to specify how the service connects to the associated JCA adapter.</p> <p>Managed: Recommended for production. The JCA transport connects to the JCA adapter through the JCA adapter-managed connection factory configured in WebLogic Server. For authentication, specify a JNDI service account. If no JNDI service account is specified, an anonymous subject is used.</p> <p>Non-Managed: The JCA transport connects to the JCA adapter through the JCA adapter framework, which acts as a container for the JCA adapter. For authentication, specify a JNDI service account. If no JNDI service account is specified, an anonymous subject is used. You can edit the connection factory properties for overrides.</p> <p>Notes: These options are only available for certain adapters, such as legacy and Cloud adapters. If you want to change from non-managed mode to managed mode, deselect the Overwrite Connection Authentication Properties option before changing modes.</p>
Overwrite Connection Authentication Properties	<p>Select this option to specify that the user name and password in the adapter connection factory be overwritten by the Connection Authentication Service Account credentials. If no service account is specified, an anonymous subject is used.</p> <p>This option is available only if the connection factory properties contain user name and password properties, the connection mode is Non-Managed, and Always use configuration from JCA file is not selected.</p>
Connection Authentication Service Account	<p>Browse to and select the service account to use for authentication if you selected Overwrite Connection Authentication Properties above (you must specify a service account if you selected Overwrite Connection Authentication Properties). For proxy services, only static service accounts are available.</p>
Connection Factory Properties	<p>Enter any connection factory override values.</p> <p>Notes: You can override property values if you deselect Always use configuration from JCA file and the connection mode is non-managed. In production environments, use managed mode so the JCA transport connects to the adapter connection factory configured in WebLogic Server.</p>

For more information on endpoint and activation and interaction spec properties, see Adapter Framework in *Understanding Technology Adapters*.

Proxy Service Operation Configuration

JCA WSDL files support document literal binding only. The only algorithm for Operation configuration in JCA proxy services is `SOAPAction` Header. Service Bus effective WSDL files always contain `SOAPAction`. The value of the `SOAPAction` field is the operation name.

Creating REST Services with Oracle Service Bus

This chapter describes how to integrate Representational State Transfer (REST) operations as binding components in Service Bus projects. It also describes Web Application Definition Language (WADL) documents, which are the WSDL equivalent for RESTful services.

You can only create REST proxy and business services from JDeveloper. Once you do this, you can export the REST services and then import them into the Oracle Service Bus Console if needed.

This chapter includes the following sections:

- [Oracle Service Bus and REST](#)
- [WADL Documents for REST Services in Service Bus](#)
- [Creating WADL Documents](#)
- [Modifying WADL Documents](#)
- [Creating REST Services](#)
- [Accessing WADL Documents in a Web Browser](#)

Oracle Service Bus and REST

REST is an architecture for designing network applications. RESTful applications use HTTP requests to post data (create and update), get data (for example, make queries), and delete data. REST provides a lightweight alternative to traditional WSDL-based web services.

REST Features in Service Bus

Service Bus provides the following REST support:

- REST support in proxy and business services
- Integration with external REST APIs
- XML, JavaScript Object Notation (JSON) with translation to and from XML, and URL-encoded data
- JDeveloper wizard for modeling REST interfaces and WSDL mappings
- Automatic creation of the required WADL file
- Ability to browse and consume Oracle REST endpoints from within JDeveloper

- Oracle Web Services Manager (OWSM) policy support for REST security

While Service Bus does not follow the Hypermedia as the Engine of Application State (HATEOAS) approach, it does support certain HATEOAS features by defining the actions in the pipeline. This includes the following capabilities:

- Setting the HTTP link header in a REST proxy service response.
- Reading the value of the HTTP link header in a REST business service response.
- Overriding the endpoint URI for a REST business service request.

When you create a REST proxy service, it can be invoked from standalone REST clients. The REST business services you create can be invoked as WSDL-based services.

Note:

Service Bus does not support MIME attachments with REST, though you can return payload that contains links to attachment content.

REST Implementation in Service Bus

Service Bus provides a virtualization layer to support REST, which means that only proxy and business services are REST-based. These REST services invoke, or are invoked by, a WSDL-based pipeline or split-join. REST proxy services convert the REST native payload to SOAP before invoking a pipeline or split-join, while REST business services invoked by a pipeline or split-join convert the payload from SOAP to REST. The internal interface is WSDL-based, while the external business and proxy services expose REST endpoints.

Only SOAP document-style WSDL files with operations where a single part is defined by an element are supported for REST services. You can create WADL files that do not have SOAP extension annotations or do not necessarily conform to the limitations above, as long as they are syntactically valid. However, Service Bus checks any WADL used by a REST service for both syntax and semantics to make sure it conforms to the rules.

REST Security

REST services in Service Bus can be secured using Oracle Web Services Manager (OWSM) policies. For more information, see "Which OWSM Policies Are Supported for RESTful Web Services and Clients?" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

WADL Documents for REST Services in Service Bus

When you create a proxy or business service based on the REST binding, JDeveloper automatically generates the required WADL document. The WADL file generated by the Create REST Binding wizard is based either on an existing WSDL file or a WSDL file that you create through the REST Binding wizard. It defines the structure of the service. In the runtime, Service Bus uses WADL documents to compute the relevant WSDL operation and to transform the message payload. The payload is translated from a REST media type, such as JSON, XML, or URL-encoded, into the format expected by the pipeline, and then translated back to the REST media type expected by

the service. Only SOAP document-style WSDL files with operations that define a single part by an element are supported for REST services.

WADL files are saved in JDeveloper with an extension of `.wadl`. You can use the standard JDeveloper XML Editor to edit the WADL files you create through the REST binding wizard. A WADL file can have dependencies on one or more XML schemas.

WADL Documents in the Design Time and Runtime

In the design time, Service Bus uses WADL documents to define the structures for new services. In the runtime, Service Bus uses WADL documents to compute the relevant WSDL operation and to transform the message payload. The payload must be translated from a REST media type, such as JSON, XML, or URL-encoded, into the format expected by the pipeline. It then must be translated back to the REST media type expected by the service.

Query Operations with WADL

When configuring query-style operation parameters, you can configure expressions based on either payload or properties. Payload-based expressions, such as `$msg.parameters/tns:symbol`, specify values of elements or attributes of the corresponding abstract request message part. Property-based expressions, such as `$property.PropertyName` specify values of request metadata of the `$inbound` variable at runtime.

For proxy services, or inbound requests, you can query the `$inbound` variable in the pipeline to retrieve the values of named user-metadata elements. As an example, for `$property.PropertyName`, you can query the value of `$inbound/ctx:transport/ctx:request/tp:user-metadata[@name = 'PropertyName']/@value`. For business services, or outbound requests, you can assign values of parts of the `$outbound` variable in the pipeline to the corresponding request parameters.

WADL Restrictions

Only SOAP document-style WSDL documents with operations where a single part is defined by an element are supported for REST services. SOAP RPC-style and generic XML-style WSDL documents are not supported. This is enforced during design-time validation. For each operation or method, only element types are supported for representation. Schema types cannot be used.

Effective WADL Documents

As with WSDL-based services, Service Bus uses *effective* WADL documents in the runtime instead of the actual `.wadl` files you create when you develop Service Bus RESTful services. The effective WADL document represents a service's WADL properties as configured in Service Bus and also includes additional properties configured outside of the source WADL document. The effective and design-time WADL documents differ in the following ways:

- The effective WADL document includes base URI information that contains the service endpoint URL.
- The effective WADL document does not include the SOA extension attributes.

For more information about the difference between the design-time files and the effective files, see [About Effective WSDL Documents and Generated WSDL Documents](#).

Creating WADL Documents

When you create a business or proxy service based on the REST binding in JDeveloper, the Create REST Binding wizard automatically generates the WADL file for the service. For more information and instructions, see [Creating REST Services](#).

If you are using the Oracle Service Bus Console, you can create WADL documents by importing them or by creating a WADL resource and uploading an existing WADL file to the new resource

How to Create a WADL Resource in the Oracle Service Bus Console

This section describes uploading an existing WADL file to a new WADL resource. For more information on importing resources, see [Importing and Exporting Resources and Configurations](#).

To create a WADL resource in the console:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.
2. In the Project Navigator, right-click the project or folder to contain the new WADL document, point to **Create**, and select **WADL**.

The Create WADL dialog appears.

3. Do one of the following:
 - To create the resource from an existing WADL file, click **Browse** next to the **File Upload** field and then navigate to and select the WADL file to use.
The **Resource Name** field is automatically populated with the file name minus the file extension. You can change this name.
 - To create a new WADL file, or to upload the WADL file at a later time, enter a unique name for the WADL resource.
4. Optionally, enter a brief description of the resource.
5. Click **Create**.

The WADL elements, if defined, appear in the WADL Definition Editor.

6. To modify the WADL file content, do the following:
 - a. Click **Edit Source** in the toolbar.
The Edit Source dialog appears.
 - b. To browse to and select a new WADL file to upload, click **Browse**.
 - c. To modify the contents of the file, update the code directly in the Contents section of the dialog.
 - d. Click **Save**.
7. In the WADL Definition Editor toolbar, click **Save**.

If there are any unresolved references for the new WADL document, a **Conflict** icon appears next to the editor's title bar. Use the previous and next arrow buttons to scroll through any errors.

8. To end the session and deploy the configuration to the runtime, click **Activate**.

Modifying WADL Documents

Once you create a WADL document in a Service Bus project, you can edit or remove the document as needed.

- [How to Edit a WADL Document](#)
- [How to Delete a WADL Document](#)

How to Edit a WADL Document

WADL documents are a standard feature in JDeveloper. You can display a WADL document in a standard editor, and modify the source code directly.

If you are using the Oracle Service Bus Console, use the following procedure to edit WADL documents. You can edit the code directly or upload an updated WADL file to the resource.

To edit a WADL document in the console:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.
2. In the Project Navigator, click the WADL resource to edit.

The WADL Definition Editor appears.

3. Click **Edit Source** in the toolbar.

The Edit Source dialog appears.

4. To browse to and select a new WADL file to upload, click **Browse**.
5. To modify the contents of the file, update the code directly in the Contents section of the dialog.
6. Click **Save**.
7. To end the session and deploy the configuration to the runtime, click **Activate**.

How to Delete a WADL Document

If any resources reference the WADL document you want to delete, remove those references before deleting the WADL resource. In the Oracle Service Bus Console, open the WADL document in the WADL Definition Editor and click the **References** icon in the upper right to find out if it has any references. In JDeveloper, right-click the WADL document and select **Explore Dependencies**.

To delete a WADL document:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the WADL document to delete.
2. Right-click the name of the WADL document, and select **Delete**.

The WADL resource is deleted. A Deletion Warning icon appears when other resources reference this resource. You can delete the resource with a warning confirmation. This might result in conflicts due to unresolved references to the deleted resource.

3. If you are using JDeveloper, a confirmation dialog displays the number of references for the WADL document. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Creating REST Services

You can only create REST business and proxy services using JDeveloper. You can either create new services using the Create REST Binding wizard in the Service Bus Overview Editor, or you can expose existing WSDL services as REST, including pipelines and split-joins. You can also create REST services from artifacts stored in the Oracle Metadata Services (MDS) repository, as described in [Consuming Artifacts Stored in the MDS Repository](#).

To use REST proxy and business services in the Oracle Service Bus Console, you need to import the projects or resources from a configuration JAR file.

A REST proxy service using a WSDL SOAP 1.1 or 1.2 binding can call any of the following:

- A WSDL-based proxy service, pipeline, split-join or business service with the same WSDL binding.
- Any SOAP 1.1 or 1.2 binding-type proxy service, pipeline, or business service. The SOAP version must be the same as the invoking proxy service.
- A REST business service with the same WADL reference and WSDL binding.

A REST proxy service can be invoked by REST and HTTP transport business services.

For more information about REST services and the REST Binding, see "Integrating REST Operations in SOA Composite Applications" in *Developing SOA Applications with Oracle SOA Suite*.

Note:

Once a REST binding has been created, do not modify the fault schema. For more information about the fault binding, see "What You May Need to Know About REST Fault Binding" in *Developing SOA Applications with Oracle SOA Suite*.

How to Create REST Services for Service Bus

When you use a REST binding to create a proxy or business service, Service Bus generates the required WADL file along with an HTTP-typed proxy or business service. The REST binding is based on a WSDL document. Depending on how you configure the REST binding, a proxy service can be based on an existing WSDL file or you can create the WSDL file when you generate the proxy service. When you create a REST business service, the wizard generates both the WSDL and WADL files.

To create a REST proxy or business service:

1. From the Components window, select **Service Bus** and drag a REST Binding into the Proxy Services or External Services lane in the designer.

Tip:

You can also right-click in a swimlane and select **REST** from the menu that appears.

The Create REST Binding wizard appears.

2. Enter a name and optionally enter a brief description of the REST service.

Tip:

For help with the configuration fields, click **Help** or press **F1**. Help is available for all dialogs you work with in this process.

3. Leave the **Type** field at the default value, which is selected depending on whether you are creating a proxy service (Service) or business service (Reference).
4. If you are creating a REST business service, enter the **Base URI**. This is the endpoint URI for the business service.
5. In the Resources section, do the following:
 - a. Double-click the default path entry of */*.
The Update REST Resource dialog appears.
 - b. In the **Relative Path** field, specify the resource path, and click **OK**.
 - c. To define additional resources to assign to individual operations, click the **Add** icon in the Resources section and repeat the above step.
6. In the **Operation Bindings** section, click the **Add** icon and do one of the following:
 - Select **Add operation binding** to manually create and define a new REST operation. You can add multiple bindings. For more information, see [How to Create or Configure a REST Operation](#)
 - (*Business services only*) Select **Add operations based on WADL service** to launch the Select WADL dialog and select an existing WADL file containing the operations to implement. You can browse for a WADL file in the file system, applications, an MDS repository, and so on.
 - (*Proxy services only*) Select **REST enable component or reference** to launch the Resource Chooser and select a Service Bus component in the current application from which to generate REST operations.
 - (*Proxy services only*) Select **REST enable external web service** to launch the WSDL Chooser and select a WSDL file from which to generate REST operations. You can browse for a WSDL file in the file system, applications, an MDS repository, a UDDI registry and so on.

The operations appear in the Operation Bindings table. If you selected one of the last two options, you must configure the bindings as described in the following step. This is optional for the first two options.

7. To configure any of the bindings, do the following:
 - a. Double-click a binding.
The REST Operation binding dialog appears.

- To enter a default value for a parameter, click in the **Default Value** column for the parameter you want to update, and enter the value to use.
 - To launch the Expression Builder for adding or updating an XPath expression function, click in the **Expression** column for the parameter you want to update and then click the Expression Builder icon. Use the Expression Builder to define the XPath expression to use.
 - To manually add a parameter, click **Add parameter** and enter the values in the new row that appears.
 - To remove a parameter, select the row and click **Delete parameter**.
7. Click the **Response** tab to configure the response.
 8. In the **HTTP Statuses** field, enter the HTTP status codes returned for a successful operation, separated by spaces.
 9. In the **Payload** field, select the type of payload for the response.
 10. To generate a sample payload that you can then save to a file, click **Generate Sample Payload**.
 11. To define the schema for the response, do one of the following:
 - Click **Browse for Schema File** to navigate to and select an XML schema type for the operation.
 - Click **Define Schema for Native Format** to launch the Native Format Builder to define a custom translation.

The **Element** field and the fault bindings are updated based on your selection.
 12. To manually add a fault binding, click **Add fault binding**. On the REST Fault Binding dialog, do the following:
 - a. Enter a name for the fault.
 - b. Enter a space-separated list of HTTP statuses returned for the fault.
 - c. Select the type of payload for the fault. To generate a sample payload for the fault, click **Generate Sample Payload**.
 - d. Select a schema and element that defines the fault.
 - e. Click **OK**.
 13. To update a fault's HTTP statuses or payload type, double-click it in the fault binding table and update the information on the REST Fault Binding dialog.
 14. On the REST Operation Binding dialog, click **OK**.

How to Expose an HTTP Proxy or Business Service as REST

In JDeveloper, you can expose an existing WSDL-based proxy service, pipeline, split-join, or business service as a REST service, which generates a proxy service and the associated WADL document. You can also expose a REST-based business service. The generated proxy service is automatically wired to the service from which it was created.

To expose a Service Bus service as REST:

1. In the Application Navigator, locate the proxy service, pipeline, split-join, or business service you want to expose as a REST service.
2. Right-click the service, point to **Service Bus**, and then select **Expose as REST**.

The Create REST Binding wizard appears.

3. Optionally, modify the name and enter a brief description of the REST service.

Tip:

The **Type** field cannot be modified. Because you are exposing a service, the default is Service (proxy service).

For help with the configuration fields, click **Help** or press **F1**. Help is available for all dialogs you work with in this process.

4. To specify that JSON payloads be reordered to match the order of elements in the XML schema, select **Enforce XMLSchema Ordering**.
5. To enter a new resource path, click the **Add** icon in the Resources section.
6. If necessary, double-click in the HTTP Verb column of the Operation Bindings section to configure the methods.
7. Click **OK**.
8. If the Localize Files dialog appears, clear the check box if you do not want to maintain the original directory structure, and click **OK**.
9. Do one of the following:
 - Continue configuring the business service, as described in [Configuring Business Services](#).
 - Continue configuring the proxy service, as described in [Configuring Proxy Services](#).
10. Click **Save All** in the JDeveloper toolbar.

What you May Need to Know About Configuring URI Parameters for REST

When you create a REST binding, you can use XPath expressions to define the value for the URI parameters for the operation. When you configure query-style operation parameters, the expression can either be based on payload or on a property. Expressions based on the payload, such as `$msg.parameters/tns:symbol`, specify values of elements or attributes of the corresponding abstract request message part. Expressions based on a property, such as `$property.SomeProperty`, specify values of request metadata of the `$inbound` variable at runtime.

For inbound requests, you can query the `$inbound` variable in the pipeline to get the values of named metadata elements. For outbound requests, you can assign the appropriate values of parts of the `$outbound` variable in the pipeline to corresponding request parameters.

Accessing WADL Documents in a Web Browser

You can view both the design-time and the effective WADL documents through a web browser. Accessing the files through a browser window displays the contents of the file in XML format. Service Bus also provides a human readable interface to make viewing the contents easier.

Viewing WADL Documents in XML Format

Do any of the following to view Service Bus WADL documents in a web browser.

- To view the service effective WADL, enter the fixed HTTP URL in a web browser using the following syntax:

```
http://host:port/sbresource?PROXY/project_path/proxy_service_name
```

or

```
http://host:port/sbresource?BIZ/project_path/business_service_name
```

This works for all services for which Service Bus can generate service effective WADL documents. While there is also a WSDL document associated with this service, the above syntax displays only the service effective WADL document.

- To view the design-time WADL document, enter the fixed HTTP URL in a web browser using the following syntax:

```
http://host:port/sbresource?WADL/project_path/wadl_name
```

Viewing WADL Documents in a Readable Format

You can view both design-time and service effective WADL documents in a browser using a format that is easier to read. To view the WADL document, open a web browser and enter the URL to the service as described below.

- To view the service effective WADL document, use the following syntax:

```
http://host:port/sbresource?PROXY/project_path/proxy_service_name&HTML=true
```

or

```
http://host:port/sbresource?BIZ/project_path/business_service_name&HTML=true
```

While there is also a WSDL document associated with this service, the above syntax displays only the service effective WADL document.

- To view the design-time WADL document, use the following syntax:

```
http://host:port/sbresource?WADL/project_path/wadl_name&HTML=true
```

The WADL content appears in the browser in a format similar to the image below.

Figure 26-1 Readable Interface for WADL Documents

Service

Grammars

```
http://localhost:7101/sbresource?BIZ%2F%2FStorageService%3ASCHEMA%2F%2FStorageGetContainers
http://localhost:7101/sbresource?BIZ%2F%2FStorageService%3ASCHEMA%2F%2FStorageAddItem
http://localhost:7101/sbresource?BIZ%2F%2FStorageService%3ASCHEMA%2F%2FStorageAddContainer
http://localhost:7101/sbresource?BIZ%2F%2FStorageService%3ASCHEMA%2F%2FStorageRaiseError
http://localhost:7101/sbresource?BIZ%2F%2FStorageService%3ASCHEMA%2F%2FStorageAlreadyExistsFault
```

Resources Detail

http://localhost:7001/StorageService

RestReference

Resources:

- /containers**
http://localhost:7001/StorageService/containers
Methods:
 - GET**

Using the DSP Transport

This chapter provides an overview of the DSP (Oracle Data Service Integrator) transport and describes how to use and configure it in your services.

This chapter includes the following sections:

- [Introduction to the DSP Transport](#)
- [Enabling Data Services for Service Bus](#)
- [Using the DSP Transport](#)
- [DSP Transport Configuration Reference](#)

For information on supported Service Bus interoperability with Oracle Data Service Integrator, see "Interoperability Scenarios and Considerations" in *Administering Oracle Service Bus*. For information about the Oracle Data Service Integrator, see *Developing Applications with Data Service Integrator*.

Introduction to the DSP Transport

Oracle Data Service Integrator can be accessed by Service Bus through the DSP transport, allowing Service Bus to make full use of data services. This approach also allows a more efficient and flexible approach to accessing data services compared to exposing such services as web services.

Enabling Data Services for Service Bus

To make an Oracle Data Service Integrator data service available to a Service Bus client requires the following steps:

- Generate a WSDL file for the data service and import the new WSDL file into Service Bus.
- Create and configure a business service based on the WSDL file.
- Create and configure a proxy service based on the business service.
- Create and configure a pipeline to route and transform messages between the proxy and business service.

Once these tasks are complete, you can invoke data services through Service Bus.

Using the DSP Transport

This section takes you through a sample project that illustrates the use of a data service in Service Bus. In order to follow the steps in this sample, you must be using

the sample WebLogic domain. For more information, see "Sample Applications and Code Examples" in *Understanding Oracle WebLogic Server*.

- [Generate the WSDL File in Oracle Data Service Integrator](#)
- [Create the Service Bus Project](#)

Generate the WSDL File in Oracle Data Service Integrator

Perform the following steps in Oracle Data Service Integrator.

- [Step 1. Start Your Server](#)
- [Step 2. Generate a WSDL File from the Data Service](#)
- [Step 3: Obtain the Web Service Address](#)

Step 1. Start Your Server

Start the Oracle Data Service Integrator server if it is not already running. This scenario uses the sample `RetailDataspacespace` provided with Oracle Data Service Integrator on the sample domain.

Step 2. Generate a WSDL File from the Data Service

You can generate the WSDL file from your data service using Data Services Studio. You can also export the WSDL file using the Oracle Data Service Integrator Console, or view and copy the WSDL definition from an existing web service map file. For information and instructions, see "Generating a Web Service Map and WSDL from a Data Service" in *Developing Applications with Data Service Integrator*.

Step 3: Obtain the Web Service Address

Use the following steps to obtain the URL address of the WSDL file.

1. Right-click the WS file (example: `OrderService.ws`).
2. Select **Test Web Service**.
3. When the Test Client opens, save the URL address.

Below is the address for the `OrderService` example:

```
http://localhost:7001/RetailDataspacespace/RetailApplication/OrderManagement/  
OrderService.ws?WSDL
```

Create the Service Bus Project

Perform the following steps in the Oracle Service Bus Console or in JDeveloper.

- [Step 4: Import the Data Service WSDL File into Service Bus](#)
- [Step 5: Create the Business Service](#)
- [Step 6: Create the Proxy Service](#)
- [Step 7: Create a Pipeline](#)
- [Step 8: Test Your Setup](#)

Step 4: Import the Data Service WSDL File into Service Bus

Service Bus lets you import a WSDL file generated in Oracle Data Service Integrator into Service Bus using the Oracle Service Bus Console or JDeveloper. This scenario uses the Service Bus example server and the Default project. The default user name is **weblogic**; you defined the password when you created the example server.

Create a new Service Bus project, and import the WSDL file and associated files into the new project. For more information about importing, see [Importing and Exporting Resources and Configurations](#).

Step 5: Create the Business Service

Create a business service from the WSDL file imported from Oracle Data Service Integrator. Use the following guidelines to configure the business service:

- For the business service transport, select **dsp**.
- For the service type, select **WSDL**, browse to and select the imported WSDL file, and then select the appropriate port or binding; for example, `OrderServiceSoapBinding`.
- For the endpoint URI, enter the URI to the Oracle Data Service Integrator project; for example, `t3://localhost:7001/RetailDataspac`.

On the Transport Details page of the Business Service Definition Editor, configure the transport settings. For more information, see [Table 27-1](#) or the online help for the transport detail page. For more information on creating business services, see [Creating and Configuring Business Services](#).

Step 6: Create the Proxy Service

Generate a proxy service from the business service, as described in [How to Generate a Proxy Service from an Existing Service in JDeveloper](#). You can only generate a proxy service from a business service using JDeveloper. Alternatively, you can create a proxy service in the console without generating it from the business service.

In practice you would most likely identify the encryption key, digital signature key, and SSL client authentication key. However, for the example, none of these need to be identified. The DSP transport uses the character set provided by the proxy service. Therefore if the default character set needs to be changed prior to invoking a data service transport, the conversion encoding needs to be handled within the proxy service itself.

Step 7: Create a Pipeline

Create a pipeline to perform any data transformations and to route the data from the proxy service to the business service. For more information, see [Working with Pipelines in Oracle Service Bus Console](#) or [Working with Pipelines in Oracle JDeveloper](#).

Step 8: Test Your Setup

Use the following steps to test access to the Oracle Data Service Integrator data service through the proxy service generated above.

1. Either deploy the project in JDeveloper or activate the session in the Oracle Service Bus Console.

2. Launch the test console for the proxy service you created. For more information, see [Accessing the Test Console](#).
3. From the Available Operations list, select your data service operation (for example, `getOrderByCustID`).
4. In the **Payload** field, enter the information needed by the data service. For example:

```
<ord:getOrderByCustID
  xmlns:ord="ld:RetailApplication/OrderManagement/OrderService.ws">
  <ord:custID>CUSTOMER3</ord:custID>
</ord:getOrderByCustID>
```

5. Click **Execute**. The data appears in the response document, as shown in [Figure 27-1](#).

Figure 27-1 Request and Response from the Service Bus Test Console

Proxy Service Testing - orderService2 Help

Back Close

Request Document

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  </soap:Header>
  <soapenv:Body>
    <ord:getOrderByCustID xmlns:ord="Id:RetailApplication/OrderManagement/OrderService.ws">
      <ord:custID>CUSTOMER3</ord:custID>
    </ord:getOrderByCustID>
  </soapenv:Body>
</soapenv:Envelope>
```

Response Document

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <env:Body xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <ns:getOrderByCustIDResponse xmlns:ns="Id:RetailApplication/OrderManagement/OrderService.ws">
      <ns0:ORDER xmlns:ns0="urn:retailer">
        <ns1:ORDER TYPE="APPL" xmlns:ns1="urn:retailerType">
          <OrderID>ORDER_3_0</OrderID>
          <CustomerID>CUSTOMER3</CustomerID>
          <OrderDate>2001-10-01</OrderDate>
          <ShippingMethod>PRIORITY-1</ShippingMethod>
          <HandlingCharge>6.8</HandlingCharge>
          <SubTotal>649.85</SubTotal>
          <TotalOrderAmount>656.65</TotalOrderAmount>
          <SaleTax>0</SaleTax>
          <EstimatedShipDate>2001-10-03</EstimatedShipDate>
          <Status>CLOSED</Status>
          <ShipTo>ADDR_3_0</ShipTo>
          <ShipToName>Britt Pierce</ShipToName>
          <BillTo>CC_3_1</BillTo>
          <TrackingNumber>ORDER_3_00379624444</TrackingNumber>
        <LINE_ITEM>
          <LineItemID>0</LineItemID>
          <OrderID>ORDER_3_0</OrderID>
          <ProductID>APPA_SH_4</ProductID>
          <ProductDescription>Debra Sandal at Nodstrom</ProductDescription>
          <Quantity>1</Quantity>
          <Price>249.95</Price>
          <Status>CLOSED</Status>
        </LINE_ITEM>
        <LINE_ITEM>
          <LineItemID>1</LineItemID>
          <OrderID>ORDER_3_0</OrderID>
          <ProductID>APPA_SH_5</ProductID>
          <ProductDescription>Audrey Hepbun from Farragamo</ProductDescription>
          <Quantity>1</Quantity>
          <Price>299.95</Price>
          <Status>CLOSED</Status>
        </LINE_ITEM>
        <LINE_ITEM>
          <LineItemID>2</LineItemID>
          <OrderID>ORDER_3_0</OrderID>
          <ProductID>APPA_BA_1</ProductID>
          <ProductDescription>Cucci Dejavu Hobo</ProductDescription>
          <Quantity>1</Quantity>
          <Price>99.95</Price>
        </LINE_ITEM>
      </ns0:ORDER>
    </ns:getOrderByCustIDResponse>
  </env:Body>
</soapenv:Envelope>
```

DSP Transport Configuration Reference

This section provides descriptions for DSP transport-specific properties for business services.

- [DSP Transport Endpoint URIs](#)
- [Configuring Business Services to Use the DSP Transport](#)

DSP Transport Endpoint URIs

When you create business services that use the DSP transport, enter the endpoint URI for the business service in the following format:

```
t3://dsp-ip-address:port/dsp-app-name
```

For example:

```
t3://localhost:7001/RetailDataspace
```

Configuring Business Services to Use the DSP Transport

The following table describes the properties you use to configure a DSP-based business service. For more information, see [Creating and Configuring Business Services](#).

Table 27-1 DSP Transport Configuration Properties for Business Services

Property	Description
Debug Level	Enter one of the following options to specify how to handle debug information: <ul style="list-style-type: none"> • 0: No debug information • 1: Output information on the request message • 3: Output information on the request and the response message
Service Account	Enter a service account that will be used for authentication to access the service. If no service account is specified, an anonymous subject is used. For more information, see Working with Service Accounts .
Dispatch Policy	Select the instance of WebLogic Server Work Manager that you want to use to post the reply message for response processing. The default Work Manager is used if no other Work Manager exists. For information about Work Managers, see: <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>

Using the EJB Transport

This chapter provides an overview of the EJB transport and describes how to use and configure it in your services. Using the EJB transport, Service Bus supports native RMI invocation of EJB 2.1 or EJB 3.0 stateless session beans deployed on supported platforms. It allows transactional and secure communications. You can also leverage the EJB transport to expose an EJB as a web service through Service Bus.

This chapter includes the following sections:

- [Introduction to the EJB Transport](#)
- [Prerequisites for Creating Services that Invoke EJBs](#)
- [Invoking EJB Business Services](#)
- [Exposing EJBs as Web Services](#)
- [Advanced EJB Transport Topics](#)
- [Troubleshooting EJB Transports](#)
- [EJB Transport Configuration Reference](#)

Introduction to the EJB Transport

In Service Bus, you can use business services configured with the EJB transport for publish, service callout, and service invocations. You cannot create proxy services that use the EJB transport.

The EJB transport provides the following capabilities:

- **Transactional Integrity:** EJB business services can be called in the context of a global transaction. The EJB transport can also suspend or start a global transaction before invoking an EJB.
- **Security Propagation:** The security context established from a Service Bus client is propagated to the other external system. For example, an incoming SOAP over HTTP request to Service Bus that requires authentication is authenticated by Service Bus and the authenticated subject can then be propagated to the EJB server. For more information on security propagation, see "Important Information Regarding Cross-Domain Security Support" in *Administering Security for Oracle WebLogic Server*.
- **HTTP Tunneling and Encrypted Communication:** You can access EJBs that are behind a firewall with HTTP tunneling. For additional security, use SSL to encrypt all of the communications with the EJB Server.
- **JNDI Provider:** EJB transport leverages the JNDI provider (a Service Bus resource). The JNDI provider defines communication protocols and security credentials for

accessing remote servers. A JNDI provider can be reused by multiple EJB business services. This provides a centralized way for administrators to manage remote EJB server configurations.

For information about JNDI provider resources, see [Working with JNDI Provider Resources](#).

- **High Performance Caching:** The EJB transport is built on high performance cache. This allows the reuse of established connections and minimizes EJB stubs lookups.
- **Failover and Load Balancing:** The EJB transport can take advantage of scenarios in which the same EJB is deployed in multiple domains or on a cluster for load balancing or failover or both.
- **Advanced XML to Java Binding Capabilities:** The EJB transport leverages the Oracle WebLogic Server JAX-RPC stack to perform Java to XML bindings. The JAX-RPC stack is a high performance engine that supports advanced Java objects such as XML Beans. If the Java type is not recognized by the stack, an extension mechanism is provided to facilitate support of these Java types. For information about this extension mechanism (using the converter classes), see [Supported Types and Converter Classes](#).
- **Intelligent Retries:** The EJB transport makes retry decisions based on the nature of the failure that can occur during the invocation of an EJB.

Prerequisites for Creating Services that Invoke EJBs

Before you can configure a business service to use the EJB transport, you must create a JNDI provider resource and a client JAR resource. The following topics outline the steps required to design and configure an EJB transport business service in Service Bus.

- [Registering a JNDI Provider Resource](#)
- [Registering an EJB Client or Converter JAR Resource](#)

Registering a JNDI Provider Resource

A JNDI provider resource allows you to specify the communication protocols and security credentials used to retrieve EJB stubs bound in the JNDI tree of remote Oracle WebLogic Server domains. Typically, the target EJB is not located in the same domain as Service Bus. In this case, you must register a JNDI provider resource. When the EJB is located in the same domain, you can define a provider to specify credentials and take advantage of stubs caching, although it is optional in this case.

The JNDI provider has a high performance caching mechanism for remote connections and EJB stubs. The preferred communication protocol from Service Bus to an Oracle WebLogic Server domain is `t3` or `t3s`. If messages need to go through a firewall, you can use HTTP tunneling.

Note:

Although it is possible to use an Oracle WebLogic Server foreign JNDI provider, Oracle recommends that you do not.

The transport does not support two-way SSL or client certificate to look-up JNDI tree or access a method on an EJB.

For information about registering and configuring a JNDI provider resource in Service Bus, see [Working with JNDI Provider Resources](#).

Registering an EJB Client or Converter JAR Resource

In order to be used by Service Bus services, a client JAR file must be registered as a resource in Service Bus. It becomes a part of the Service Bus configuration and can be exported from and imported into a project. An EJB client JAR file must contain the interfaces and classes needed by Service Bus to access an EJB. This includes the remote and home interfaces (EJB 2.1) or business interfaces (EJB 3.0) and any dependent types to which the client is exposed, such as method parameter types or application exceptions.

If your business service requires converter classes, you can register a JAR file containing the converter classes as a Service Bus resource and subsequently use these classes to help map parameter and return value types to Java classes that can be mapped to XML. Alternatively, you can package these converter classes in the EJB client JAR. For information about converter classes, see [Custom Converter Classes](#).

Consider the following guidelines when using EJB client JARs:

- Adding home and remote interfaces in the system classpath is bad practice and is not supported by Service Bus.
- Oracle recommends that you keep the client JAR file size small, include a single home interface per JAR file, and not register the entire `ejb-jar` file.
- Service Bus supports `client-jar` files compiled with JDK 1.4 or later.

Adding a Client or Converter JAR File

In order to use complex type inheritance by extension in your Service Bus EJB implementation, you need to create and use converter classes, as described in [Supported Types and Converter Classes](#). For information about registering and configuring a client or converter JAR resource in Service Bus, see [Working with JAR Files](#).

Create a Service Account (Optional)

If the EJB methods are protected, you can specify the credentials you want to use for the invocations using a service account. Those credentials are often different than the credentials used by the JNDI provider. For information about adding and using service accounts, see [Working with Service Accounts](#).

Locate an EJB in the JNDI Tree

If you do not know the JNDI name for an EJB, you can browse the EJB Server JNDI tree. For information about browsing the JNDI tree using the Oracle WebLogic Server Administration Console, see "View objects in the JNDI" tree in the *Oracle WebLogic Server Administration Console Online Help*.

Invoking EJB Business Services

An EJB business service can be used as a SOAP XML business service. You can publish to, route to, or callout to an EJB business service. If you need transaction support, set the quality of service to *Exactly-Once*. For more information, see [Advanced EJB Transport Topics](#).

You can also use the Test Console to validate your configuration and to help you to determine the shape of the XML request.

Exposing EJBs as Web Services

You can leverage the EJB transport to easily expose EJBs as web services.

Note:

You cannot create a proxy service from an existing EJB business service. You must first get the WSDL file generated from the EJB business service, and then create the proxy service based on that WSDL file. To do so, complete the following steps.

1. Create an EJB business service pointing to the EJB you want to expose.
2. From the service details page, get the WSDL file for the EJB business service.

The WSDL file is contained in a JAR file. You can obtain the WSDL file only if there is no pending session.
3. Extract the WSDL file from the JAR file and register it as a WSDL resource.

If the configuration of the business service changes, a new WSDL file is generated. If that happens, you must get the new WSDL file and re-register it as a WSDL resource.
4. Create a SOAP XML proxy service based on the WSDL file.
5. Create and edit a pipeline to route the proxy service to the EJB business service.

Advanced EJB Transport Topics

This section includes information about the EJB transport that will help you understand how EJB business services behave at runtime depending on how they are configured at design time.

- [EJB Transport Transactions](#)
- [EJB Transport Retries and Failover](#)
- [EJB Transport Error Handling](#)
- [Supported Types and Converter Classes](#)
- [Business Exception Classes](#)

EJB Transport Transactions

The EJB transport can create, suspend, and propagate transactions. The transactions between Service Bus and the EJB server are XA transactions. If you use transactions with HTTP tunneling or have a dedicated communication channel, you must set the *security interoperability* mode for the transaction manager to *performance*. For information about setting the security interoperability mode and other transaction configurations, see "Configuring Transactions" in *Developing JTA Applications for Oracle WebLogic Server*.

For the deployment descriptors to be set appropriately for XA-capable resources, you must set the XA attribute on the referenced connection factory before creating a proxy service.

To determine the behavior of the EJB business service, considerations include whether the pipeline has a transactional context, and what quality of service (QoS) settings are specified in the pipeline when invoking the service:

- **QoS Best-Effort:** If *Best-Effort* QoS is specified in the pipeline, no transaction is propagated to the EJB, and any ongoing transaction is suspended before invocation and resumed after invocation.
- **QoS Exactly-Once:** If *Exactly-Once* QoS is specified in the pipeline and the EJB does not support transactions (that is, the **Supports Transaction** option on the EJB transport configuration page is not checked), no transaction is propagated to the EJB. As in the case of *Best-Effort*, any ongoing transaction is suspended before invocation and resumed afterwards.

If the EJB supports transactions (that is, the **Supports Transaction** option on the EJB transport configuration page is checked), the EJB is invoked in the context of a transaction, and any ongoing transaction is propagated to the EJB. If no transaction is present, a transaction is created before invocation and committed afterwards.

For more information about QoS in Service Bus services, see [Quality of Service](#).

EJB Transport Retries and Failover

Assuming that an EJB business service is configured for retries or failovers, the EJB transport distinguishes the following types of exceptions:

- Runtime exceptions or remote exceptions: These are typically unexpected fatal errors or communication exceptions.
- Exception raised by the JAX-RPC engine: These are exceptions that occur during the XML to Java conversion.
- EJB checked exceptions: These are exceptions declared in the EJB method signature specific to the EJB implementation. They are also called business exceptions.

Retries and failover are based on the type of errors and also in the QoS, as described below.

QoS Best-Effort

- If a runtime or remote exception is thrown, the EJB transport attempts retries or failovers.
- If an exception occurs in the JAX-RPC engine, an error is raised to the pipeline and no retries or failover attempts are made.
- If an EJB checked exception is thrown, an error is raised to the pipeline and no retries or failover attempts are made.

QoS Exactly-Once

- If a runtime or remote exception is thrown and the ongoing transaction has been set as *rollback only* (likely by the EJB container), the EJB container has been reached and a fatal error occurred either within the EJB container or the EJB. In this case, no retries or failover attempts are made and an error is raised to the pipeline.

- If a runtime or remote exception is thrown but the ongoing transaction has not been set as *rollback only*, an error occurred before the invocation of the EJB container and the EJB transport will attempt retries or failovers. Note that in this case, the EJB transport still respects the *exactly-once* semantic.
- If an exception occurs in the JAX-RPC engine, the EJB transport sets the ongoing transaction to *rollback only* and an error is raised to the pipeline; no retries or failover attempts are made.
- If an EJB Checked Exception is thrown, an error is raised to the pipeline and no retries or failover attempts are made.

See [EJB Transport Transactions](#) for other repercussions of QoS specifications for an EJB business service.

EJB Transport Error Handling

When throwing a checked exception, according to the EJB specifications, the ongoing transaction can be specified as *rollback only*. If the ongoing transaction is set as *rollback only* by the EJB developer, the transaction is eventually rolled back by its creator (most likely the proxy service).

If the ongoing transaction is not set to *rollback only*, and a checked exception is raised, it is important to catch EJB checked exceptions in the pipeline with an error handler. If those exceptions are not caught, the pipeline errors are propagated back to the proxy service. The proxy service, in turn, is likely to rollback the ongoing transaction (depending of the transport implementation). This may not be the intended result.

For example, assume you have an EJB with the following method:

```
public void withdrawFunds(float amount) throws RemoteException,  
InsufficientFundsException {,Ä¶}
```

Also assume that when an `InsufficientFundsException` exception is thrown, the EJB does not set the current transaction as *rollback only*. In most scenarios, it is wrong to allow the proxy service to roll back the transaction; you may need to configure an error handler in the pipeline to catch the error and avoid this scenario.

Supported Types and Converter Classes

The EJB transport is responsible for the conversion between XML and Java. The conversion is performed by the Oracle WebLogic Server JAX-RPC engine. The EJB transport natively supports the following types:

- Primitive types
- XmlObject (both Apache and Oracle versions)
- Schema generated XMLBeans (both Apache and Oracle versions)
- JavaBean classes

For the full list of natively supported types, see "Using JAXB Data Binding" in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

An EJB method can use parameters and return types that are either not supported by the JAX-RPC engine (an error is reported at design time) or that do not map directly to XML (errors occur at runtime). The most commonly used unsupported types include the following:

- "Object", "Object[]"
- Java Collections, as they are not strongly typed (for example, List or Set)
- Java classes that do not follow the JavaBean pattern (for example, Map)

You can write a custom converter class that converts those types into types more suitable for conversion between XML and Java.

Custom Converter Classes

Service Bus generates an intermediate web service that precisely matches the EJB structure. Since not all EJBs and their data types can be mapped to web service data types, you can create a converter class to change the shape of the intermediate web service and enable the internal transformation of the SOAP data types into the required EJB types on the Service Bus side. A converter class is a Java class that implements and conforms to the contract defined by the `com.bea.wli.sb.transports.ejb.ITypeConverter` Java interface of the Service Bus public API.

The JAX-RPC specification requires certain criteria of EJB parameters and return types for wrapping them into a web service layer, with specific requirements for data types. For more information, see "Programming the JWS File" in *Developing JAX-RPC Web Services for Oracle WebLogic Server*. Converter classes let you customize the Service Bus EJB wrapper in a way that exposes data types over the web service layer, which can then be turned into the data types required by the EJB. The wrapper implementation is based on the provided EJB client classes, must match the EJB interface precisely, and is accessible through a web service layer (created using WebLogic Servers's JWS toolset).

Using a Converter Class for an EJB Business Service

To use a converter class for an EJB business service:

1. Create a converter class by implementing and compiling the interface.
2. Add the converter class to the client JAR file or to a converter class JAR file (see [Adding a Client or Converter JAR File](#)).
3. When configuring the methods for the EJB business service, navigate to one of the parameters or return types and select the desired converter.

For more information, see [EJB Transport Configuration Reference](#). When you configure the business service, the Transport Details page displays a list of the available converters that can be applied to a particular parameter or return type.

Business Exception Classes

Business exception classes thrown by an EJB method must comply with JAX-RPC 1.1 Specification Section 5.5.5 to ensure proper mapping of the exception class details into the WSDL type definition for the fault element. All private, non-transient fields with corresponding getter methods in the exception class get mapped to the exception type definition in the WSDL file.

Troubleshooting EJB Transports

The information in this section is provided to help you troubleshoot problems when designing or running an EJB business service. In addition, you can enable debugging, as described in [Debugging Oracle Service Bus Applications](#).

- [Temp Directories](#)
- [Deployed Application](#)
- [EJB Transport Errors](#)

Temp Directories

During design time, the EJB transport generates files in a subfolder in the `temp` directory. It is safe to delete those folders and files, and sometimes may be useful to check them to determine what went wrong during activation.

Deployed Application

When an EJB business service is created and activated, an application is deployed on the Service Bus server. You can use the Oracle WebLogic Server Administration Console and Fusion Middleware Control to monitor and tune this application.

EJB Transport Errors

The following information may help in the event that you need to troubleshoot a problem with an EJB business service:

- The following error when creating a business service is due to a Windows operating system limitation. Paths containing more than 255 characters are not supported.

The system cannot find the path specified):Probably the string length of the path of the file being extracted was too long

You can try to reduce the path length by creating a shorter path to the Service Bus domain, or you can use the following option to override the Oracle WebLogic Server `temp` directory when starting the server:

```
-Dweblogic.j2ee.application.tmpDir=$desired_short_dir
```

- If you get an XML marshalling error when invoking an EJB business service and you believe the request to be valid against the service WSDL document, you probably need to write a converter class. For information, see [Custom Converter Classes](#).
- If the EJB interfaces and stubs are changed on the remote server, the first time you try to invoke the new EJB, an error is thrown. Those changes on the remote server are not visible to Service Bus, so it tries to invoke the cached EJB stubs, which are no longer valid. However, when the invocation error occurs, the transport assumes that those stubs are now invalid, and removes them from the cache. In this way, the error is prevented on subsequent attempts to invoke the EJB. To avoid this first-time error, you can reset the JNDI provider in the Oracle Service Bus Console.

EJB Transport Configuration Reference

An EJB business service is a transport-typed service, which means the type of the transport is determined by the configuration of the service. The type of an EJB business service is equivalent to a SOAP XML service. In other words, you can use an EJB business service like any other SOAP XML business service. Service Bus generates a WSDL file when you save the EJB transport configuration. The generated WSDL file is based on the interface of the EJB. The EJB transport configuration page provides

configuration options for you to control the interface of the service and the WSDL file that is generated.

EJB Endpoint URI Format

Use the following URI pattern for the EJB transport endpoint URIs in business services:

```
ejb:jndi_provider_name:ejb_jndi_name
```

where *jndi_provider_name* is the name of the Service Bus JNDI provider resource.

If the EJB is deployed locally, the JNDI provider name is not required. In this case, use the following URI format:

```
ejb::ejb_jndi_name
```

For EJB 3.0 business services on Oracle WebLogic Server, *ejb_jndi_name* takes the form of *mappedName#BusinessInterface*.

If your EJBs are running on IBM WebSphere, *ejb_jndi_name* must be in one of the following formats:

```
cell/nodes/node_name/servers/server_name/ejb_jndi_name
```

or

```
cell/clusters/cluster_name/ejb_jndi_name
```

For more information, refer to the IBM WebSphere documentation.

Configuring Business Services to Use the EJB Transport

The following table describes the properties you use to configure an EJB transport for a business service. For more information, see [Creating and Configuring Business Services](#).

Table 28-1 EJB Transport Properties for Business Services

Properties	Description
Pass Caller's Subject	Select this check box to have Service Bus pass the authenticated subject from the proxy service when invoking the EJB and no service accounts are configured. This is an alternative to using a service account, and the Service Account field is disabled when this option is selected.
Service Account	Enter a service account that will be used for authentication to access the service. If no service account is specified, an anonymous subject is used. This option is not available if you select the Pass Caller's Subject option. For more information, see Working with Service Accounts .
Supports Transaction	Select this check box to specify that the EJB supports transactions.
EJB 3.0	Select this check box if the interface uses EJB version 3.0.
Client Jar	Click Browse to select an EJB client JAR resource from the list of available resources.

Table 28-1 (Cont.) EJB Transport Properties for Business Services

Properties	Description
Converter Jar	<p>Click Browse to select an EJB converter class JAR resource from the list of available options. This field appears after you select the client JAR file above.</p> <p>For more information, see Adding a Client or Converter JAR File and Custom Converter Classes.</p>
Home Interface	<p>Select the required EJBHome interface from the options populated by the JAR file. This field is available for EJB 2.1 only.</p> <p>The JNDI name in this URI sample must be associated with the EJBHome interface you select here. If the EJB is not of the required type or an EJBHome interface is not specified in the client JAR file, Service Bus displays a warning.</p>
Remote Interface	<p>Displays the remote interface, and is automatically populated depending on the configuration of the Home Interface. This for EJB 2.1 only.</p>
Business Interface	<p>Select the business interface in the client JAR that you want to invoke. This field is available for EJB 3.0 only.</p>
Target Namespace	<p>Displays the namespace determined from information picked up from the JAR file. This field only appears when you select EJB 3.0 and specify a client JAR file.</p>
Style	<p>Select Document Wrapped or RPC according to your requirements. If two or more methods of your stateless session EJB have the same number and data type of parameters, and you want the operations to be document-oriented, specify that they be document-wrapped.</p> <p>The style is important because when routing or publishing to the EJB, \$body must have content that matches the style. Also when calling out to an EJB, the style affects the parameter contents, especially for document wrapped. Secondly one usage pattern is to define an EJB business service and then create a proxy service with the same WSDL file that routes to the EJB. In this case, be careful with the WSDL style because the client tool that invokes the proxy might have style limitations.</p>
Encoding	<p>Specify the encoding of the SOAP message, either Encoded or Literal.</p>
Methods	<p>Configure the methods of the EJB remote or business interface you selected. Select the required methods (you can select multiple methods). Expand the method to configure, and then edit the default parameter values and select a converter if provided (or required).</p> <p>You can change the default operation name for each method. By default, the operation name is the method name. If an EJB contains methods with same name, you must change the operation names so they are unique; WSDL files require unique operation names.</p> <p>You can select methods to include or exclude. You must exclude the methods with parameters or return types that are not supported by the JAX-RPC stack or you must associate such arguments with converter classes.</p> <p>Note: If the credentials or transaction settings are different between the methods for an EJB, you can customize the methods for a specific business service, and create a business service per method. This gives you fine-grained control over transactions and credentials.</p>

Table 28-1 (Cont.) EJB Transport Properties for Business Services

Properties	Description
Exceptions	<p>Displays any business exceptions thrown by a method. If an EJB method throws an exception that has data types not supported by Java Web Services (JWS), such as an <code>ArrayList</code>, use the <code>Exceptions</code> field to select a converter class that converts the exception to a type supported by JWS.</p> <p>Your converter class must implement <code>com.bea.wli.sb.transports.ejb.ITypeConverter</code>. Converter classes can only be configured for checked exceptions and not for runtime exceptions. Package the converter class and the converted exception class in the client or converter JAR file.</p>

Using HTTP and Poller Transports

This chapter provides an overview of the HTTP(S) and poller transports (Email, File, FTP, and SFTP) and describes how to use and configure them in your services.

The HTTP transport section also provides information on using REST with Service Bus, but you can also use a REST binding, as described in [Creating REST Services with Oracle Service Bus](#).

This document includes the following sections:

- [Using the HTTP Transport](#)
- [Using the Email Transport](#)
- [Using the File Transport](#)
- [Using the FTP Transport](#)
- [Using the SFTP Transport](#)

Introduction to Poller Transports

You can use different types of transports to configure proxy services or business services in Service Bus. The transport protocol you select depends on the service type, the type of authentication required, the service type of the invoking service, and so on. Poll-based transports have transport pollers pinned to a Managed Server. These transports poll a source directory or email server for new messages. They use the JMS framework to ensure that messages are processed at least once. Email, File, FTP, and SFTP are poll-based transports.

By default, poller transports use WebLogic Server JMS, but you can configure a clustered domain to use Oracle Advanced Queueing (AQ) JMS when you create or extend the domain. Running the Repository Creation Utility (RCU) creates all the required Service Bus queues and queue tables. The **Sort by Arrival** feature is not supported for poll-based transports with AQ JMS.

For information about configuring your environment to use Oracle AQ, see "Using Oracle Advanced Queueing JMS" in *Administering Oracle Service Bus*.

Using the HTTP Transport

The HTTP transport lets you send messages between clients and service providers through Service Bus using the HTTP(S) protocol. The HTTP transport also provides support for working with Representational State Transfer (REST) environments, as described in [REST Support](#).

HTTP Session Stickiness

Service Bus supports HTTP session stickiness, or *session affinity*, for business services in load balancing, which means that the same server handles all requests for a specific session. Service Bus maintains stickiness by mapping the session to a specific entry in a service URI table. In a sticky session, the URI entry that handles the first request has the session; when subsequent messages in the session arrive at the load balancer, they are routed through the same service URI entry that handled the first request.

In a standard load-balancing environment, Service Bus can balance the load across multiple servers, so if messages in a session need to be handled by the same server in a cluster, you need to configure the business services for session stickiness.

Note:

in the Oracle Service Bus Console, you can configure sticky sessions at runtime without needing to restart the service.

Service Bus does not support the following scenarios for sticky sessions:

- Multiple business services with session stickiness in a single message flow.
- Multiple business services with session stickiness in the same pipeline.
- Split-join services that point to business services with session stickiness.
- Dynamic routing to different business services with session stickiness.

Note also that throttling does not work for business services configured for session stickiness.

Retrieving the HTTP Authorization Header in a Proxy Service

Service Bus does not pass the HTTP Authorization header from the request to the pipeline because it opens a security vulnerability. You could inadvertently create a log action that writes the user name and unencrypted password to a log file.

If your design pattern requires the HTTP Authorization header to be in the pipeline, do the following:

1. In the startup command for Service Bus, set the following system property to true:

```
com.bea.wli.sb.transports.http.GetHttpAuthorizationHeaderAllowed
```

2. On the Transport page of the service's definition editor in JDeveloper or the Oracle Service Bus Console, select **Get All Headers** or select **User-specified Headers** and specify Authorization.
3. Restart Service Bus.

HTTP Transport Configuration Reference

This section provides information about endpoint URI formats and configuring the HTTP transport in proxy and business services.

- [HTTP Transport Endpoint URIs](#)

- [Configuring Proxy Services to Use the HTTP Transport](#)
- [Configuring Business Services to Use the HTTP Transport](#)

HTTP Transport Endpoint URIs

You can select the HTTP transport protocol when you configure any type of proxy or business service. Use the following endpoint URI formats:

- **Proxy Services:** `/service_name`
- **Business Services:** `http://host:port/service_name`

where:

- `host` is the name of the system that hosts the service.
- `port` is the port number at which the connection is made.
- `service_name` is a target service.

Note:

You must specify the following endpoint URI when you configure a business service based on HTTPS.

`https://host:port/someService`

Configuring Proxy Services to Use the HTTP Transport

The following table describes the properties you use to configure an HTTP transport for a proxy service. For more information, see [Creating and Configuring Proxy Services](#).

Table 29-1 HTTP Transport Properties for Proxy Services

Property	Description
HTTPS Required	Select this check box for inbound HTTPS endpoints. The HTTPS protocol uses SSL to secure communication. SSL can be used to encrypt communication, ensure message integrity, and to require strong server and client authentication To learn more, see Configuring Transport-Level Security for HTTPS .

Table 29-1 (Cont.) HTTP Transport Properties for Proxy Services

Property	Description
Authentication	<p>Select one of the following authentication methods:</p> <ul style="list-style-type: none"> • None: Specifies that authentication is not required. Select this option if you will specify an OWSM policy for the proxy service. • Basic: Specifies that basic authentication is required to access this service. Basic authentication instructs Oracle WebLogic Server to authenticate the client using a user name and password against the authentication providers configured in the security realm, such as a Lightweight Directory Access Protocol (LDAP) directory service and Windows Active Directory. The client must send its user name and password on the HTTP request header. Basic authentication is strongly discouraged over HTTP because the password is sent in clear text. However, it is safe to send passwords over HTTPS because HTTPS provides an encrypted channel. <ul style="list-style-type: none"> Warning: By default, all users (authorized and anonymous) can access a proxy service. To limit the users who can access a proxy service, create a transport-level authorization policy. See Configuring Transport-Level Security. • Client Certificate: Specifies encrypted communication and strong client authentication (two-way SSL). To learn more, see Configuring Transport-Level Security for HTTPS. • Custom Authentication: Specifies that an authentication token is contained in an HTTP header. The client's identity is established through the use of this client-supplied token. You must configure an identity assertion provider that maps the token to a Service Bus user. <p>The custom authentication token can be of any active token type supported by a configured WebLogic Server Identity Assertion provider.</p>
Dispatch Policy	<p>Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists.</p> <p>For information about Work Managers, see:</p> <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>
Request Encoding	<p>Specify the character set encoding for the request message.</p> <ul style="list-style-type: none"> • For HTTP inbound transports: If the character set encoding parameter of the <code>Content-Type</code> header is not specified in Client Request, enter a character set encoding parameter. If you do not enter a value, the field defaults to ISO-8859-1. • For HTTP outbound transports: If you have not configured a request encoding, the Service Bus runtime determines the most appropriate encoding while it makes a request to the business service. In the case of a non-pass-through scenario, the default character encoding is UTF-8 at runtime. However if it is a pass-through scenario, the runtime will pass through the encoding received with the outbound response.
Response Encoding	<p>Specify the character set encoding for the response message.</p>

Table 29-1 (Cont.) HTTP Transport Properties for Proxy Services

Property	Description
Authentication Header	Enter the HTTP header (any header except <code>Authorization</code>) from which Service Bus will extract the token. This field is available only if you selected the Custom Authentication check box. For example, <code>client-xyz-token</code> .
Authentication Token Type	Select an authentication token type. Only the active token types configured for an Identity Assertion provider are available. This field is available only if you selected the Custom Authentication check box.

Configuring Business Services to Use the HTTP Transport

The following table describes the properties you use to configure an HTTP transport for a business service. For more information, see [Creating and Configuring Business Services](#).

Table 29-2 HTTP Transport Properties for Business Services

Property	Description
Read Timeout	Enter the read timeout interval in seconds. If the timeout expires before data is available on the connection, a connection error occurs. A zero (0) value indicates no timeout.
Connection Timeout	Enter the connection timeout interval in seconds. If the timeout expires before the connection can be established, Service Bus raises a connection error. A zero (0) value indicates no timeout.

Table 29-2 (Cont.) HTTP Transport Properties for Business Services

Property	Description
HTTP Request Method	<p>Select one of the following HTTP methods to use in requests:</p> <ul style="list-style-type: none"> • POST: Use POST to pass all its data, of unlimited length, directly over the socket connection as part of its HTTP request body. The exchange is invisible to the client, and the URL does not change. For REST-based requests, POST tells the transport to perform a create/replace operation or perform an action with the request. • GET: Use GET to include as part of the request some of its own information that better describes what to get. This information is passed as a sequence of characters appended to the request URL in a query string. You can use GET in a business service with an XML service type or a messaging service type when the Request Message Type is set to "None." For REST-based requests, GET retrieves a representation of a remote resource. • PUT: Use PUT to tell the transport to perform a create/replace operation with a REST-based request, such as uploading a file to a known location. You can use PUT in a business service with an XML or messaging service type. • HEAD: Use HEAD to tell the transport to get header information for a remote resource rather than getting a full representation of the resource in a REST-based request. You can use HEAD in a business service with an XML service type or a messaging service type when the Request Message Type is set to "None." • DELETE: Use DELETE to tell the transport to perform a delete operation with a REST-based request. You can use DELETE in a business service with an XML or messaging service type. <p>Note: If a method is already set in the <code>\$outbound/transport/request/http:http-method</code> variable, that value takes precedence over any method you select for HTTP Request Method.</p>

Table 29-2 (Cont.) HTTP Transport Properties for Business Services

Property	Description
Authentication	<p>Select one of the following types of HTTP authentication:</p> <ul style="list-style-type: none"> • None: Authentication is not required to access this service. • Basic: Basic authentication is required to access this service. Basic authentication instructs WebLogic Server to authenticate the client using a user name and password against the authentication providers configured in the security realm, such as a Lightweight Directory Access Protocol (LDAP) directory service and Windows Active Directory. The client must send its user name and password on the HTTP request header. <p>Basic authentication is strongly discouraged over HTTP because the password is sent in clear text. However, it is safe to send passwords over HTTPS because HTTPS provides an encrypted channel.</p> <p>Warning: By default, all users (authorized and anonymous) can access a business service. To limit the users who can access a business service, create a transport-level authorization policy. See Configuring Transport-Level Security.</p> <ul style="list-style-type: none"> • Client Certificate: Specifies encrypted communication and strong client authentication (two-way SSL). To learn more, see Configuring Transport-Level Security for HTTPS. • Custom Authentication: Specifies that a custom Java class defines authentication. You must specify the authentication class in the HTTP Custom Authentication Class Name field in the advanced settings (described below).
Service Account	<p>Enter a service account that will be used for authentication to access the service. This is a required field if you select basic authentication. It is optional if you select custom authentication. For more information, see Working with Service Accounts.</p>
Dispatch Policy	<p>Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists.</p> <p>For information about Work Managers, see:</p> <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>
Request Encoding	<p>Accept the default <code>iso-8859-1</code> as the character set encoding for requests in HTTP transports, or enter a different character set encoding.</p>
Response Encoding	<p>Accept the default <code>iso-8859-1</code> as the character set encoding for responses in HTTP transports, or enter a different character set encoding.</p>
HTTP Custom Authentication Class Name	<p>Enter the name of the Java class used for custom authentication. This field is only available if you selected Custom Authentication for the authentication type. For information about creating the custom authentication class, see How to Create a Custom Authentication Class for Outbound.</p>

Table 29-2 (Cont.) HTTP Transport Properties for Business Services

Property	Description
Proxy Server	Enter a proxy server resource or click Browse to choose an entry from the list of configured proxy server resources.
Follow HTTP redirects	Select this check box to specify that HTTP redirects (which are requests with a response code 3xx) should be automatically followed. A redirect occurs when you send an outbound request to the URL of a business service, and that service returns a response code (for example, 302) that says the URL is no longer valid and this request needs to be sent to another URL. If you select the Follow HTTP Redirects check box, Service Bus automatically resends the request to the new URL without any action on your part. Deselect this check box if you do not want the HTTP redirects to be automatically followed.
Use Chunked Streaming Mode	Select this option if you want to use HTTP chunked transfer encoding to send messages. Note: Do not use chunked streaming with the Follow HTTP Redirects option. Redirection and authentication cannot be handled automatically in chunked mode.
Session Stickiness	Select this option if you want to use session stickiness (also known as session affinity) for HTTP requests for the business service. For more information, see HTTP Session Stickiness .
Sticky sessionID Name	Enter a unique identifying name for the session if Session Stickiness is enabled.

REST Support

The HTTP transport provides support for working with REST environments through Service Bus, whether you have REST clients that need to interact with non-REST service providers, non-REST clients that need to interact with REST-based service providers, or REST-to-REST services you want to expose through Service Bus.

In a REST pattern, you invoke HTTP methods (such as GET, PUT, HEAD, POST, and DELETE) on resources that are located at specific URLs. For example, when a user updates his own profile information in a web application that uses REST, a POST action updates the user information in the database through the service's REST API.

While Service Bus incorporates the REST binding, as described in [Creating REST Services with Oracle Service Bus](#). Service Bus also provides the following placeholder variables for handling REST-based requests for inbound and outbound communication using the HTTP transport:

- **\$inbound or \$outbound/transport/request/http:http-method:** For handling HTTP methods such as GET, PUT, HEAD, POST, and DELETE.
- **\$inbound or \$outbound/transport/request/http:parameters:** For handling a query string in a URL. The `parameters` metadata contains one or more `parameter` elements that store name-value pairs. The name-value pairs are URLDecoded values in the query string. For example, in the URL `http://localhost:7021/myproxy/weather?operation=temperature&pincode=94065`, the query string is stored as the following query parameters:

```
<http:query-parameters>
  <http:parameter name="operation" value="temperature"/>
  <http:parameter name="pincode" value="94065"/>
</http:query-parameters>
```

- **\$inbound or \$outbound/transport/request/http:relative-URI:** For handling relative portions of a REST resource URL (relative to the proxy service URI). For example, in the URL `http://localhost:7021/myproxy/weather`, `/weather` is a relative URL to `http://localhost:7021/myproxy`.

REST in Proxy Services

With a proxy service, you have the flexibility to interact with REST patterns, whether you are receiving REST-based requests or generating REST-based actions.

For example, to develop REST-based applications and invoke services in a non-REST service provider, you can send REST operations through a proxy service and transform those operations into a format the service provider understands; or you could transform a non-REST request into a resource URL and invoke an operation in a REST-based service provider. You can also use Service Bus as an intermediary for monitoring, auditing, and reporting on REST-to-REST implementations.

For working samples that use REST with Service Bus, see the Oracle Service Bus Samples repository, accessible from <http://www.oracle.com/technetwork/middleware/service-bus/learnmore/index.html>.

XQuery Examples

Following are XQuery examples of URI parsing using HTTP variables in a proxy service. URI parsing lets you transform messages between REST and non-REST environments.

Relative-URI

A proxy service has a URI of `http://localhost:7001/weather`, and you want to capture the relative URI parts of a request. You create the following XQuery:

```
<relative-URI>
{
  for $c in
  fn:tokenize($inbound/ctx:transport/ctx:request/http:relative-URI, "/")
  where fn:string-length($c) != 0
  return
  <part>
  {$c}
  </part>
}
</relative-URI>
```

If a request comes with the URI of `http://localhost:7001/weather/temperature/35457`, the relative URI is `/temperature/35457`, and the XQuery output is as follows:

```
<relative-URI>
  <part>temperature</part>
  <part>35457</part>
</relative-URI>
```

Query-String Parameters

A proxy service has a URI of `http://localhost:7001/weather`, and you want to capture the URL query string parameters. You create the following XQuery:

```
<query-parameters>
{
return
<param name="$inbound/ctx:transport/ctx:request/http:parameters/param/name"
value="$inbound/ctx:transport/ctx:request/http:parameters/param/value"></param>
}
</query-parameters>
```

If a request comes with a URI of `http://server:7001/weather?operation=temperature&pincode=35457`, the query string is `operation=temperature&pincode=35457`, which are stored as individual parameters in the `http:parameters` metadata. The XQuery output is:

```
<query-parameters>
  <param name='operation' value='temperature' />
  <param name='pincode' value='35457' />
</query-parameters>
```

Headers

If your service requires specific headers to handle HTTP/REST methods, create user-defined HTTP header variables in your pipeline.

REST in Business Services

With a business service, you can invoke REST-based services. For REST operations, the HTTP transport uses the value in the `$outbound/transport/request/http:http-method` variable. If that variable does not supply an HTTP method, the HTTP transport lets you select one of the following HTTP request methods in the transport configuration: POST, PUT, HEAD, GET, AND DELETE.

Note:

If the business service uses a WSDL service type, only the POST method is available.

Using the `$outbound/transport/request-http/http-method` variable, you can also supply your own methods. For example, you can use COPY, MOVE, and LOCK for WebDAV environments (Web-based Distributed Authoring and Versioning).

Use the following guidelines for setting `$outbound` variables:

- The transport does not provide runtime validation for custom methods or for manually set supported methods that do not comply with the constraints described in this section.
- Since `$outbound` is only available in a routing node, you cannot specify a method name at runtime for publish and service callout actions.
- If the `$outbound` query string parameters are set, the business service uses outbound request encoding for building the query string from raw bytes and URL encoding for parameter name-value pairs retrieved from the `parameters` metadata element.
- If the `$outbound` relative URI is set, the business service uses that value to generate the URI, which is relative to the business service URI.

For example, in a business service with a URI of

```
http://service.com/purchaseOrder
```

with the following HTTP variables

```
$outbound/transport/request-http/relative-URI: "/P012367" and
$outbound/transport/request-http/parameters/parameter: "item=N01"
$outbound/transport/request-http/parameters/parameter: "color=black"
```

the final resolved URI is

```
http://service.com/purchaseOrder/P012367?item=N01&color=black
```

Response Codes and Error Handling for HTTP Business Services

Service Bus aligns with the HTTP 1.1 specification for handling all response codes in range 200-500. [Table 29-3](#) describes how Service Bus handles the different response code levels.

HTTP business services must be of the following types to receive response payloads containing errors 300 and greater:

- WSDL
- Any SOAP
- Messaging, with the following conditions:
 - The response message type must be MFL or XML (determined to be of type XML or SOAP). HTTP business services with response types that Service Bus determines to be other than XML or SOAP do not receive HTTP response payloads.
 - The Content-Type HTTP header in the response payload is text/xml, application/*any_string*+xml, or multipart/related.

Table 29-3 Response Code Handling for HTTP Business Services

Response Status Codes	Description
100s	100-level status codes indicate a provisional response consisting of only the Status-Line and optional headers, terminated by an empty line. The response behavior for 100-level codes is already handled by the internal <code>URLConnection</code> Java class, so Service Bus does not handle these.
200s	200-level status codes indicate a successful operation. Service Bus returns these as the response payload to HTTP business services regardless of the business service type.
300s	<p>300-level status codes are errors which indicate that further action needs to be taken by the caller in order to fulfill the request. Service Bus returns these as the response payload to HTTP business services with supported message response types.</p> <p>In the response pipeline, you can take appropriate action to handle 300-level codes.</p> <p>To handle 300-level response codes that indicate the need for a redirect, set the "Follow HTTP Redirects" option on the HTTP business service transport configuration.</p>

Table 29-3 (Cont.) Response Code Handling for HTTP Business Services

Response Status Codes	Description
400s	400-level status codes indicate a client error. Service Bus returns these as the response payload to HTTP business services with supported message response types. In the response pipeline, you can take appropriate action to handle 400-level codes.
500s	500-level status codes indicate a server error. Service Bus returns these as the response payload to HTTP business services with supported message response types. In the response pipeline, you can take appropriate action to handle 400-level codes.

Using the Email Transport

You can use the email transport with services that have a Messaging or Any XML service type. The following topics describe how to configure proxy services and business service using the email transport. The email transport supports one-way messaging for services with the Messaging service type. When you create a Messaging type proxy service or business service using the email transport you must set the response type to none in the service configuration.

Email Transport Configuration Reference

This section provides information about endpoint URI formats and configuring the email transport in proxy and business services.

- [Email Transport Endpoint URIs](#)
- [Configuring Proxy Services to Use the Email Transport](#)
- [Configuring Business Services to Use the Email Transport](#)

Email Transport Endpoint URIs

When you configure a *proxy service* using the email transport, specify the endpoint URI in the following format:

```
mailfrom:mailserver-host:port
```

where *mailserver-host* is the name of the server hosting the mail service and *port* is the port number used by that server.

When you configure a *business service* using the email transport, you can specify one or more endpoint URIs in the following formats, which lets you send email messages to multiple recipients in multiple domains:

```
mailto:name@domain_name.com
```

```
mailto:name@domain_name.com?smtp=smtp_server_resource
```

```
mailto:name@domain_name.com?mailsession=jndi_mail_session
```

For example:

mailto:user1@example1.com

mailto:user2@example2.com?smtplib=exampleSMTP

mailto:user3@example3.com?mailsession=my.mail.Session

Configuring Proxy Services to Use the Email Transport

The following table describes the properties you use to configure an email transport for a proxy service. For more information, see [Creating and Configuring Proxy Services](#).

Table 29-4 Email Transport Properties for Proxy Services

Property	Description
SSL Required	Select this option to communicate using Secure Sockets Layer. The default SSL port numbers are: <ul style="list-style-type: none"> POP3(S): 995 IMAP: 993 Note: The email server certificate must be present in the Service Bus truststore in order for SSL to work. The Email Transport supports one-way SSL communication.
Service Account	Enter a service account that will be used for authentication to access the email account. The service account consists of a user name and password combination required to access the email account. For more information, see Working with Service Accounts .
Managed Server	Select the Managed Server to act as the polling server. All of the Managed Servers can process the message, but only one can poll for the message. This field is available only in a clustered domain.
Polling Interval	Enter an interval in seconds between attempts to poll for new messages to process. The default value is 60.
Email Protocol	Select POP3 or IMAP as the email protocol to use to connect to the email server.
Read Limit	Specify the maximum number of messages to read for each polling sweep. Enter 0 to specify no limit. The default value is 10.
Pass By Reference	Select this check box to stage the file in the archive directory and pass it as a reference in the headers. By default when you create a new service, the Pass By Reference option is selected and you must specify the archive directory location.
Pass Attachments by Reference	Select this check box to stage the attachments in the archive directory and pass them as a reference in the headers. By default, when the Pass By Reference option is selected, the Pass Attachments By Reference option is implicitly true and you must specify the archive directory location.

Table 29-4 (Cont.) Email Transport Properties for Proxy Services

Property	Description
Post Read Action	Select what happens to a message after it has been read: <ul style="list-style-type: none"> • Archive: The message is archived. If you select this option, specify an archive directory where Service Bus will archive the messages. • Delete: The message is deleted. • Move: The message is moved. If you select this option, specify an IMAP move folder to which Service Bus will move the message. Move is only available with the IMAP protocol.
Attachments	Select how attachments are handled: <ul style="list-style-type: none"> • Archive: Attachments are saved to the archive directory. • Ignore: Attachments are ignored. <p>Note: If attachments are archived, the attachment files are passed as a reference in the message headers irrespective of the settings for the Pass By Reference parameter.</p>
IMAP Move Folder	Enter the folder to which the message is moved if the Post Read Action field is set to Move . You must configure this field if Post Read Action is set to move.
Download Directory	Enter a temporary location for downloading emails.
Archive Directory	Specify the path to the archive location if the Post Read Action field is set to Archive . This field is required if the Pass By Reference or Pass Attachments By Reference option is selected. It is active only when Post Read Action property is set to archive.
Error Directory	Enter the file system directory path to write the message and any attachments if there is a problem.
Request Encoding	Accept the default ISO-8859-1 as the character set encoding for requests in email transports, or enter a different character set encoding.

Configuring Business Services to Use the Email Transport

The following table describes the properties you use to configure an email transport for a business service. For more information, see [Creating and Configuring Business Services](#).

Table 29-5 Email Transport Properties for Business Services

Property	Description
SSL Required	Select this option to communicate using Secure Sockets Layer. The Email Transport supports one-way SSL communication. The default SMTP port number when using SSL is 465. Note: The email server certificate must be present in the Service Bus truststore in order for SSL/TLS to work.

Table 29-5 (Cont.) Email Transport Properties for Business Services

Property	Description
SMTP Server	Select the default SMTP server to use for endpoint URI entries of <code>name@domain_name.com</code> . If you provide SMTP server parameters in the endpoint URI, those server resources are used instead of this SMTP server setting. Do not select an SMTP server if you use the Mail Session option. You must first create the SMTP server resource. For more information, see How to Create SMTP Server Resources .
Mail Session	Enter the JNDI name of the configured mail session to use for endpoint URI entries of <code>name@domain_name.com</code> . If you provide JNDI mail session parameters in the endpoint URI, those mail sessions are used instead of this Mail Session setting. Do not enter a Mail Session if you use the SMTP server option.
From Name	Enter a display name for the originating email account for this service. You must first configure mail sessions in the Oracle WebLogic Server Administration Console.
From Address	Enter the originating email account for this service. You need to create a Mail Session in Oracle WebLogic Server Administration Console. You must also set the Mail Session parameter or the SMTP server parameter.
Reply To Name	Enter a display name for the reply to email account. This is the name from which the reply should be sent.
Reply To Address	Enter an email address to send replies to.
Connection Timeout	Enter the timeout interval, in milliseconds, before the connection is dropped. If you enter 0, there is no timeout.
Socket I/O Timeout	Enter the socket I/O timeout interval in milliseconds. If this value is zero (0), there is no timeout.
Request Encoding	Accept the default ISO-8859-1 as the character set encoding for requests in email transports, or enter a different character set encoding.

Using the File Transport

Use the File transport to poll messages from files on a shared file system located in the directory you specify. Once a file is read, Service Bus can either delete or archive the file depending on how you configure the transport. To ensure that no programs can access a file while Service Bus is writing it to the remote location, a File transport business service temporarily appends ".a" to the file name until the file is completely written.

The File transport supports the Messaging and Any XML service types. For the Messaging service type, the supported message types for the request are binary, text, MFL, and XML. The response message type must be **None** because the File transport supports only one-way messaging.

File Transport Configuration Reference

This section provides information about endpoint URI formats and configuring the File transport in proxy and business services.

- [File Transport Endpoint URIs](#)
- [Configuring Proxy Services to Use the File Transport](#)
- [Special Considerations for NFS File Systems](#)
- [Configuring Business Services to Use the File Transport](#)

File Transport Endpoint URIs

For both proxy services and business services using the File transport, enter the endpoint URI in the following format:

```
file:///<root-dir/dir1>
```

where `root-dir/dir1` is the absolute path to the directory where a proxy service polls for files or where a business service writes files.

Configuring Proxy Services to Use the File Transport

The following table describes the properties you use to configure a File transport for a proxy service. For more information, see [Creating and Configuring Proxy Services](#).

Table 29-6 *File Transport Properties for Proxy Services*

Property	Description
File Mask	Specify the files that the proxy service should poll. If the URI is a directory and you specify <code>*.*</code> , the service polls for all the files in the directory. You can use the wildcard characters <code>*</code> and <code>?</code> in the file mask. Regular expressions are not supported.
Managed Server	Select the Managed Server to act as the polling server. All of the Managed Servers can process the message, but only one can poll for the message. This field is available only in a clustered domain.
Polling Interval	Specify an interval in seconds between polling attempts for new files to process. The default value is 60. Note: In order to verify that a file is not being modified when it is picked up for processing, the transport polls for files once and then polls again 5 seconds later and compares the file size. This might result in a delay of 5 seconds over the polling interval set here. Make sure to set this value high enough to give your system enough time to complete processing and unlock the file.
Read Limit	Specify the number of files to be read in each poll. The default value is 10. If 0 is specified, all the files are read.

Table 29-6 (Cont.) File Transport Properties for Proxy Services

Property	Description
Sort By Arrival	Specify the sequence of events raised in the order of the arrival of files. The default value for this parameter is False . When this option is selected for a proxy service that is executed in a clustered environment, messages are always sent to the same server. In other words, load balancing across servers is ignored when this option is selected.
Scan Subdirectories	Select this check box to recursively scan all the subdirectories in the polling directory.
Pass By Reference	Select this check box to stage the file in the archive directory and pass it as a reference in the headers. If you select this option, you must specify an archive directory as well.
Post Read Action	Select what happens to a message after it has been read: <ul style="list-style-type: none"> • Archive: The message is archived. If you select this option, you must specify an archive directory as well. • Delete: The message is deleted.
Stage Directory	Enter an intermediate directory to temporarily stage the files while processing them. This is mandatory regardless of the options you selected above. Do not put the stage directory inside of the polling directory (the directory identified in the URL of the file transport proxy service).
Archive Directory	Specify the path to the archive location if the Post Read Action option is set to Archive. The Archive Directory field is also a required field if you selected the Pass By Reference field. Note: You must not put the archive directory inside the polling directory
Error Directory	Enter the directory in which the contents of the file will be stored in case of an error. This is mandatory regardless of the options you selected above. Do not put the error directory inside of the polling directory.
Request Encoding	Accept the default <code>utf-8</code> as the character set encoding for requests in file transports, or enter a different character set encoding.

Special Considerations for NFS File Systems

When the File transport polls for files, it first gets the list of files in the directory and their sizes. It then polls again after either 5 seconds or the polling interval, whichever is smaller. The transport gets the list of files again, and compares the file sizes to the first list. It processes only those files whose size remained constant throughout the interval. The standard NFS file system does not support the file locking mechanism required for this process. There are a few options you can use to address this.

- Use extra signaling, such as file renaming after the file is transferred, in combination with a special file name mask.

- To enable file growth protection on NFS, use the `noac` NFS mount option. This prevents caching of the files, making sure that the `ls` command is the actual file size, not the end-result claimed file size.
- Use NFS version 4 or later, which introduced additional file locking features.
- If you are using NetApp filer storage, use **Mixed Qtree** security on the volume, which makes the file locking mechanism usable for CIFS to NFS.

Configuring Business Services to Use the File Transport

The following table describes the properties you use to configure a File transport for a business service. For more information, see [Creating and Configuring Business Services](#).

Table 29-7 File Transport Properties for Business Services

Property	Description
Prefix	Enter a prefix that the transport prepends to the file name on the remote server. Do not enter an asterisk (*) in this field; this character causes a runtime exception. Do not use any characters that are invalid for the target operating system.
Suffix	Enter a suffix that the transport appends to the file name on the remote server. This is a required field. Do not enter an asterisk (*) in this field; this character causes a runtime exception. Do not use any characters that are invalid for the target operating system.
Request Encoding	Accept the default <code>utf-8</code> as the character set encoding for requests in File transports, or enter a different character set encoding.

Using the FTP Transport

Use the FTP transport to poll messages from files on a remote file system located in the directory you specify. Once a file is read, Service Bus can either delete or archive the file depending on how you configure the transport. To ensure that no programs can access a file while Service Bus is writing it to the remote location, an FTP transport business service temporarily appends ".a" to the file name until the file is completely written.

The FTP transport supports the Messaging and Any XML service types. For the Messaging service type, the supported message types for the request are binary, text, MFL, and XML. The response message type must be **None** because the FTP transport supports only one-way messaging.

FTP Transport Configuration Reference

This section provides information about endpoint URI formats and configuring the FTP transport in proxy and business services.

- [FTP Transport Endpoint URIs](#)
- [Configuring Proxy Services to Use the FTP Transport](#)

- [Configuring Business Services to Use the FTP Transport](#)

FTP Transport Endpoint URIs

For both proxy and business services using the FTP transport, enter the URI in the following format:

```
ftp://<hostname:port/directory>
```

where

- `hostname` is the name of the FTP server on which the source or destination directory is located.
- `port` is the port number at which the FTP connection is made.
- `directory` is the destination directory on the FTP server. For proxy services, this is the location the service polls for new files. For business services, this is the location where files are written.

The directory is relative to the working directory of the FTP session. For example, if the working directory is `/home/my_ftp/` and the proxy service endpoint URI is `ftp://ftp_server:21/documents`, the service polls for files in `/home/my_ftp/documents`. Using the same working directory, if the business service endpoint URI is `ftp://ftp_server:21/output`, files are written to `/home/my_ftp/output`.

Configuring Proxy Services to Use the FTP Transport

The following table describes the properties you use to configure an FTP transport for a proxy service. For more information, see [Creating and Configuring Proxy Services](#).

Table 29-8 FTP Transport Properties for Proxy Services

Property	Description
User Authentication	Select one of the following types of user authentication: <ul style="list-style-type: none"> • anonymous: This option does not require any login credentials to log in to the FTP server, but you optionally supply your email ID for identification. • external user: This option references a service account resource, which contains your user name and password for the FTP server. You must select the service account to use if you select this option.
Identity (email ID)	Enter the mail ID for the anonymous user. This field is available only if the User Authentication option is set to anonymous .
Service Account	Enter a service account that will be used for authentication to access the FTP server. This field is only available and is required when the User Authentication option is set to external user . You must have already created the service account resource in Service Bus. For more information, see Working with Service Accounts .
Pass By Reference	Select this check box to stage the file in the archive directory and pass it as a reference in the headers. If you select this option, you must specify an archive directory below.

Table 29-8 (Cont.) FTP Transport Properties for Proxy Services

Property	Description
Remote Streaming	Select this check box to stream the FTP files directly from the remote server at the time of processing. When you select this option, the archive directory is the remote directory on the remote FTP server machine. Therefore, you should specify the archive directory as relative to the FTP user directory
File Mask	Enter the regular expression for the files to be polled by the proxy service. If the URI is a directory and the file mask is *.*, the service polls all files in the directory.
Managed Server	Select the Managed Server to act as the polling server. All of the Managed Servers can process the message, but only one can poll for the message. This field is available only in a clustered domain.
Polling Interval	Enter the interval (in seconds) at which the service polls for files to process. The default value is 60. Note: In order to verify that a file is not being modified when it is picked up for processing, the transport polls for files once and then polls again 5 seconds later and compares the file size. This might result in a delay of 5 seconds over the polling interval set here. Make sure to set this value high enough to give your system enough time to complete processing and unlock the file.
Read Limit	Specify the maximum number of messages to read per polling sweep. Enter 0 to specify no limit. The default is 10.
Post Read Actions	Select what happens to a message after it has been read. <ul style="list-style-type: none"> • Archive: The message is archived. If you select this option, you must also specify an archive directory below. • Delete: The message is deleted.
Transfer Mode	Select binary or ASCII as the file transfer mode. By default the transfer is binary.
Archive Directory	Specify the path to the archive location if the Post Read Action option is set to Archive . This field is also required if the Pass By Reference option is selected. Note: The Archive, Download, and Error directories are absolute paths, and they are automatically created. If you specify a relative path, the files are created relative to the Java process that starts the WebLogic Server.
Download Directory	Enter the directory on your local machine where files are downloaded during the file transfer. Note: The Archive, Download, and Error directories are absolute paths, and they are automatically created. If you specify a relative path, the files are created relative to the Java process that starts the WebLogic Server.

Table 29-8 (Cont.) FTP Transport Properties for Proxy Services

Property	Description
Error Directory	Enter the location where the contents of the file is stored in case of a error. Note: The Archive, Download, and Error directories are absolute paths, and they are automatically created. If you specify a relative path, the files are created relative to the Java process that starts the WebLogic Server.
Request Encoding	Specify the type of encoding to read the request message. The default encoding is <code>utf-8</code> .
Scan Subdirectories	Select this check box to recursively scan all directories when polling for files.
Sort By Arrival	Select this check box to deliver events in the order of arrival.
Timeout	Enter the socket timeout interval, in seconds, before the connection is dropped. If you enter 0, there is no timeout.
Retry Count	Specify the number of retries for FTP connection failures.

Configuring Business Services to Use the FTP Transport

The following table describes the properties you use to configure an FTP transport for a business service. For more information, see [Creating and Configuring Business Services](#).

Table 29-9 FTP Transport Properties for Business Service

Property	Description
User Authentication	Select one of the following types of user authentication: <ul style="list-style-type: none"> anonymous: This option does not require any login credentials to log in to the FTP server, but you optionally supply your email ID for identification. external user: This option references a service account resource, which contains your user name and password for the FTP server. You must select the service account to use if you select this option.
Identity (Email id)	Enter the mail ID for the anonymous user. This field is available only if the User Authentication option is set to anonymous .
Service Account	Enter a service account that will be used for authentication to access the service. This field is only available and is required when the User Authentication option is set to external user . You must have already created the service account resource in Service Bus. For more information, see Working with Service Accounts .
Timeout	Enter the socket timeout interval, in seconds, before the connection is dropped. The default is 60 seconds.

Table 29-9 (Cont.) FTP Transport Properties for Business Service

Property	Description
Prefix for destination File Name	Enter an optional prefix that the transport prepends to the file name on the remote server. Do not enter an asterisk (*) in this field. This character causes a runtime exception.
Suffix for the destination File Name	Enter an optional suffix that the transport appends to the file name on the remote server. Do not enter an asterisk (*) in this field. This character causes a runtime exception.
Transfer Mode	Select ASCII or binary as the file transfer mode.
Request Encoding	Accept the default UTF-8 as the character set encoding for requests in ftp transports, or enter a different character set encoding.

Using the SFTP Transport

The SFTP transport is a poll-based transport that allows you to transfer files securely over the SSH File Transfer Protocol (SFTP) using SSH version 2. It polls a specified directory at regular intervals based on a predefined polling interval. After authentication, a connection is established between Service Bus services and the SFTP server, enabling file transfer. The SFTP transport supports one-way inbound and outbound connectivity.

To ensure that no programs can access a file while Service Bus is writing it to the remote location, an SFTP transport business service temporarily appends ".a" to the file name until the file is completely written.

Note:

The SFTP transport may not correctly handle file names in multi-byte character sets.

The SFTP transport supports only Messaging and Any XML service types. For the Messaging service type, the supported message types for the request are binary, text, MFL, and XML. The response message type must be **None** because the SFTP transport supports only one-way messaging.

SFTP Transport Features

The SFTP transport provides the following features:

- The SFTP transport is available for the following service types: Any XML and Messaging (with request message type specified). For more information about configuring service types, see [Proxy Service Definitions](#) and [Business Service Definitions](#).
- The SFTP transport supports processing of large messages. When you configure a proxy service, you can enable content streaming and specify whether large

messages must be buffered in memory or in a disk file. For more information, see [Streaming Body Content](#).

- The SFTP transport allows you to select a cipher suite and security algorithms to use, or you can allow Service Bus to find a suitable match.
- For inbound message transfer, the QoS is set to **exactly-once**, which ensures that every message is processed at least once. For outbound message processing, the QoS is **best-effort**.

Note:

For messages that are not transferred, you must create the error-handling logic (including any retry logic) in the pipeline error handler.

For more information about QoS in Service Bus messaging, see [Quality of Service](#).

General Principles of SFTP Authentication

The following principles are applicable to the SFTP authentication process for both proxy and business services:

- **Connection:** The Service Bus proxy or business service always acts as the SFTP client and connects to the SFTP server.
- **Authentication by the SFTP server:** Server authentication works in one of the following ways:
 - For public key and host-based authentication, the SFTP server authenticates the connection with the public key of the Service Bus service.
 - For user name and password authentication, the SFTP server authenticates the connection with the user name and password.
- **Authentication by the SFTP client:** The Service Bus service always authenticates the SFTP server with the public-key/host/IP combination defined in the known_hosts file. For more information, see [Creating the Known Hosts File](#).
- **Connection establishment:** The connection is established only when both the server and client authentications are successful.
- **File transfer:**
 - If the client is a proxy service, the file (message) is downloaded from the SFTP server.
 - If the client is a business service, the file (message) is uploaded to the SFTP server.

SFTP Transport Runtime Behavior

Transferring files using the SFTP transport involves the following steps:

1. The proxy service polls the input directory at regular intervals.
A new connection is created for each poll cycle.

2. If the proxy service finds a file in the input directory, it renames the file with a `.stage` extension. This renaming ensures that the service does not pick up the same files during the next polling cycle.

The `.stage` file exists in the input directory until it is delivered.

Note:

If the file cannot be retrieved from the input directory (due to network failure, for example), the `.stage` file is processed during a clean-up cycle. The clean-up cycle is performed every 15 minutes or after 15 polling cycles, whichever occurs later. If a `.stage` file exists during two consecutive clean-up cycles, it is processed again.

3. A JMS task is created for the message and added to the domain-wide JMS queue.
4. A domain-wide MDB receives the task and processes the message.
The task uses a pooled connection for processing the message. If a connection is not available in the pool, a new connection is created.
5. The message is delivered to the pipeline and the `.stage` file is deleted.
6. If an SFTP business service is configured, the service puts the message in the outbound directory through a pooled connection.

If the message is not delivered, the transport attempts to transfer it again and repeats the process up to a predefined number of attempts. If the message cannot be delivered, it is moved to the error directory.

Enabling SFTP Authentication

The SFTP transport supports the following authentication methods:

- User name and password authentication
- Host-based authentication
- Public key authentication

Service Bus services authenticate the SFTP server based on the server details defined in a `known_hosts` file. To enable server authentication, you must create a `known_hosts` file on the client machine.

Creating the Known Hosts File

The `known_hosts` file must exist in the server on which the Service Bus proxy services (inbound requests) or business services (outbound requests) run. The file must contain the host name, IP address, and public key of the remote SFTP servers to which the proxy service or business service can connect.

To create the known hosts file:

1. Create a `known_hosts` file and enter information in the following format:

```
Hostname,IP algorithm publickey
```

where:

- `Hostname` is the host name of the SFTP server. The hostname is optional.
- `IP` is the IP address of the SFTP server.
If you use an IPv6 address, do not use a double colon to represent multiple zeros. Write out all zeros. For example, use this format `":0:0:0:"` instead of this format `":::"`.
- `algorithm` can be either DSA or RSA, based on the SFTP server configuration. Enter `ssh-rsa` or `ssh-dss` depending on the algorithm that is supported.
- `publickey` is the public key of the SFTP server. It must be in the Open SSH public key format.

Example - Known Hosts File

```
getafix,172.22.52.130 ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAIEAtR+M3Z9HFxnKZTx66fZdnQqAHQcF1vQe1+EjJ/HWYtg
Anqsn0hMJzqWMatb/u9yFwUpZBirjm3g2I9Qd8VocmeHwoGPhDGfQ5LQ/PPo3esE+CGwdnC
OyRcktNHeuKxo4kiCCJ/bph5dRpgHCQIvsQvRE3sks+XwQ7Wuswz8pv58=
```

The `known_hosts` file can contain multiple entries, but each entry must be on a separate line.

2. Move the `known_hosts` file to the `DOMAIN_HOME/config/osb/transports/sftp` directory.

The directory `/transports/sftp` is not created automatically. You must create it manually.

Enabling User Name and Password Authentication

User name and password authentication is the simplest and quickest method of authentication. It is based on the credentials of the user.

To enable user name and password authentication for a service:

1. Create a static service account by using the user credentials on the SFTP server. For more information, see [Working with Service Accounts](#).
2. Create a `known_hosts` file. For more information, see [Creating the Known Hosts File](#).

Enabling Host-Based Authentication

Host-based authentication uses a private host key. This method can be used when all the users share a private host.

For host-based authentication, Open SSH compares the host name provided by the client against a reverse lookup on the client IP address. Because of scenarios where the request comes from a multi-homed machine or through NAT gateway, host names may not match, resulting in authentication failure. To work around such scenarios, turn off the DNS check by setting the `HostbasedUsesNameFromPacketOnly` property to "yes" in `/etc/ssh/sshd_config`. This work around does not subvert security, because the real security is in the public key matching between the `known_hosts` file and the SFTP server.

To enable host-based authentication for a service:

1. Configure a service key provider with an SSL client authentication key. For more information, see [Working with Service Key Providers](#).

Note:

You can extract the public key from the key store that was used while creating the service key provider. The public key must be in the Open SSH format.

2. Create a `known_hosts` file. For more information, see [Creating the Known Hosts File](#).
3. Configure the SFTP server to accept requests from Service Bus, which is a client to the SFTP server.

For example, for an SFTP server on Linux, do the following:

- Edit the `/etc/ssh/shosts.equiv` file and add the host name or IP address of the machine on which the Service Bus domain runs.
- Edit the `/etc/ssh/ssh_known_hosts` file and add the host name or IP address of the machine on which the Service Bus domain runs, followed by a space and the public key.

Enabling Public Key Authentication

Public key authentication is performed using your own private key. This method can be used when each user has a private key.

To enable public key authentication:

1. Configure a service key provider with SSL client authentication key. For more information, see [Working with Service Key Providers](#).
2. Configure the SFTP server to accept requests from Service Bus (SFTP client).

For example, for an SFTP server on Linux, extract the public key from the key store and enter the key in the `$HOME/.ssh/authorized_keys` file on the SFTP server. Ensure that the path and file exist.

3. Create a `known_hosts` file. For more information, see [Creating the Known Hosts File](#).

Handling SFTP Transport Communication Errors

You can configure the SFTP transport-based business services to handle communications errors, which can occur when a connection or user authentication fails while connecting to the remote SFTP server. When you configure the business service, you can enable the business service endpoint URIs to be taken offline after a specified retry interval.

For more information, see "Managing and Monitoring Endpoint URIs for Business Services" in *Administering Oracle Service Bus*.

Troubleshooting the SFTP Transport

Most of the errors occur while configuring an SFTP proxy or business service. Below are some tips to help you understand and solve the errors.

- Make sure that you have an appropriately configured `known_hosts` file in place.
- For public key authentication, store the public key file on the server. For more information, see the documentation accompanying your SFTP server.
- A `Connection refused` error message indicates that the SFTP server is not available on the configured host and port.
- An `Authentication failed` error message indicates that the user name or password is not valid, or that the public key is not configured correctly.
- A `Connection did not complete` error message is displayed after the actual error that caused the connection failure (for example: `Key not found`).
- A `Key not found for IP, host` error message indicates that the `known_hosts` file does not contain an entry that corresponds to the specified IP-host combination. If the entry exists, then try with another algorithm key; for example, if the earlier attempt was with an RSA key, try again with a DSA key.

Importing SFTP Transport Services

This section provides information about importing Service Bus resources from configuration JAR files and UDDI registries.

- [Importing Resources Used by the SFTP Transport](#)
- [Importing and Publishing Services: UDDI Registries](#)

Importing Resources Used by the SFTP Transport

When you import a resource that already exists in an Service Bus domain, you can preserve the existing security and policy configuration details of the resource by selecting the **Preserve Security and Policy Configuration** option. The following SFTP service-specific details are preserved when you import a resource:

- Client authentication method
- Reference to the service account (for services associated with user name and password authentication)
- Reference to the service key provider (for services associated with host-based or public key authentication)
- User name (for services associated with host-based or public key authentication)

For more information about importing resources, see [Importing and Exporting Resources and Configurations](#).

Importing and Publishing Services: UDDI Registries

When an SFTP service is published to the UDDI registry, `Authentication mode`, `Request encoding`, `Sort by arrival`, and `File mask` are the properties that are published. After the service is imported, the default value of the load balancing algorithm is **round-robin**.

[Table 29-10](#) lists the properties that are imported from the registry when an SFTP service is imported from the UDDI registry.

Table 29-10 Properties Imported from UDDI Registry

Property	Description
Prefix for destination file name	The prefix for the name of the file that is stored on the remote server. The default value is "" (null).
Suffix for destination file name	The suffix for the name of the file that is stored on the remote server. The default value is "" (null).
Authentication mode	The authentication method that is imported from the registry. When an SFTP business service with user authentication is imported from an UDDI registry to Service Bus, a conflict is generated. <ul style="list-style-type: none"> • For user name and password authentication, you must create a service account and associate it with the service. • For host-based or public key authentication, you must create a service key provider and associate it with the service.

For more information, see [Working with UDDI Registries](#).

SFTP Transport Configuration Reference

This section provides information about endpoint URI formats, environment values, and configuring the SFTP transport in proxy and business services.

- [SFTP Transport Endpoint URIs](#)
- [Configuring Proxy Services to Use the SFTP Transport](#)
- [Configuring Transport Headers in the Pipeline](#)
- [Configuring Transports Headers and Metadata in the Test Console](#)
- [Configuring Business Services to Use the SFTP Transport](#)
- [SFTP Transport Environment Values](#)

SFTP Transport Endpoint URIs

For both business and proxy services using the SFTP transport, enter the endpoint URI in the following format:

```
sftp://hostname:port/directory
```

where:

- `hostname` is the host name or IP address of the SFTP server.
- `port` is the port on which SFTP server is listening. The default port for SFTP is 22.
- `directory` is the location where a business service writes outbound messages, or where a proxy service polls for files at regular intervals.

The directory is relative to the SFTP session's working directory. For example, if the working directory is `home/my_sftp/` and you want the SFTP proxy service to read files from `home/my_sftp/documents`, the URI would be similar to

sftp://sftp_server:22/documents. Using the same working directory for a business service, if the URI is sftp://sftp_server:22/output, the business service writes files to home/my_sftp/output, x

Configuring Proxy Services to Use the SFTP Transport

The following table describes the properties you use to configure an SFTP transport for a proxy service. For more information, see [Creating and Configuring Proxy Services](#).

Table 29-11 SFTP Transport Properties for Proxy Services

Property	Description
User Authentication	<p>Select the required authentication method from the following:</p> <ul style="list-style-type: none"> • Username Password Authentication: Specifies that a static service account is associated with this authentication method and the client is authenticated using the credentials provided in the service account. • Host-Based Authentication: Specifies that a user name and service key provider are required. Any user connecting from a known host is authenticated using the private key of the host. • Public Key Authentication: Specifies that a user name and service key provider are required. Users have their own private keys. <p>Note: The service does not use the service key provider to authenticate any credentials from the SFTP server. It uses only the <code>known_hosts</code> file to authenticate the SFTP server, as described in Configuring Transport-Level Security for SFTP Transport. The proxy service is authenticated by the SFTP server based on the specified user authentication method.</p>
Service Account	<p>Enter a service account that will be used for authentication to access the service. For information about using service accounts, see Working with Service Accounts.</p>
Service Key Provider	<p>Enter a service key provider to use for authentication.</p> <p>This field is available only for the public key and host-based authentication methods. For more information, see Working with Service Key Providers.</p>
Username	<p>Enter the user name to log in to the SFTP server. This value is required only when you select either the host-based or public key authentication method.</p> <ul style="list-style-type: none"> • In host-based authentication, the user name is required for polling the home directory of the user on the SFTP server. • In public key authentication, the user name is required for polling the home directory of the user and for identifying the location of the public key on the SFTP server.
Pass By Reference	<p>Select this option to stage the file in the archive directory and pass it as a reference in the headers.</p> <p>Note: This option is available only when remote streaming is disabled.</p>
Remote Streaming	<p>Select this option to stream the SFTP files directly from the remote server at the time of processing. When you select this option, the archive directory is the remote directory on the SFTP server machine. Therefore, you must specify the archive directory relative to the SFTP user directory.</p>

Table 29-11 (Cont.) SFTP Transport Properties for Proxy Services

Property	Description
File Mask	Enter a regular expression to select the files that you want to pick from the directory. Use the file mask for transferring files of specific types. The default value is *.* , which selects all files.
Managed Server	Select the Managed Server to act as the polling server. All of the Managed Servers can process the message, but only one can poll for the message. This field is available only in a clustered domain.
Polling Interval	Enter the interval (in seconds) at which the input directory is polled for messages to process. Polling involves the creation of an SFTP connection. The default value is 60. Avoid setting a low polling interval, which can cause polling to occur too frequently. Note: In order to verify that a file is not being modified when it is picked up for processing, the transport polls for files once and then polls again 5 seconds later and compares the file size. This might result in a delay of 5 seconds over the polling interval set here. Make sure to set this value high enough to give your system enough time to complete processing and unlock the file.
Read Limit	Specify the maximum number of messages to read per polling sweep. If numerous files exist in the poll directory, you can limit the number of concurrent transfers by the value you specify here. If you do not want to specify a limit, enter 0 (zero). The default value is 10. Note: In some cases, the SFTP server might limit the number of concurrent connections; make sure that the read limit you define is lower than the server-defined limit.
Post Read Action	Select what happens to the message after it has been read. <ul style="list-style-type: none"> • Archive: The message is archived in the specified directory. If you select this option, you must specify an archive directory below. • Delete: The message is deleted.
Archive Directory	Specify the path to the archive location if the Post Read Action option is set to Archive . This field is required if the Pass By Reference option is selected. When files are archived after reading, the files are moved (from either the download directory or the remote location) to the archive directory after they are read. If remote streaming is enabled, the archive directory is with respect to the SFTP server. If remote streaming is disabled, the archive directory is available on the Service Bus machine. Note: The archive, download, and error directories are absolute paths, and they are automatically created. If you specify a relative path, the files are created relative to the Java process that starts the WebLogic Server.

Table 29-11 (Cont.) SFTP Transport Properties for Proxy Services

Property	Description
Download Directory	<p>Enter the directory on your local machine where files are downloaded during the file transfer. If remote streaming is enabled, this option is disabled.</p> <p>Note: The archive, download, and error directories are absolute paths, and they are automatically created. If you specify a relative path, the files are created relative to the Java process that starts the WebLogic Server.</p>
Error Directory	<p>Enter the location where messages are posted if there is a problem. If remote streaming is enabled, the error directory is with respect to the SFTP server. If remote streaming is disabled, the error directory is available on the Service Bus machine.</p> <p>Note: The Archive, Download, and Error directories are absolute paths, and they are automatically created. If you specify a relative path, the files are created relative to the Java process that starts the WebLogic Server.</p>
Request encoding	<p>Accept the default value (UTF-8) as the character set encoding for requests in the SFTP transports.</p>
Scan Subdirectories	<p>Select this option if you want all subdirectories within the directory that is specified in the endpoint URI to be scanned recursively.</p> <p>Note: Scanning subdirectories requires additional processing overhead, so use this option judiciously.</p>
Sort By Arrival	<p>Select this option to deliver events in the order of arrival. This ensures that message delivery is not random, but based on the time at which the file is downloaded to the destination directory.</p>
Timeout	<p>Enter the socket timeout interval, in seconds, after which the connection must be dropped. If you do not want the connection to time out, enter 0. The default value is 60.</p>
Retry Count	<p>Specify the number of retries for SFTP connection failures. Use this setting to enable multiple attempts in case of errors such as network failure. The default value is 3.</p>
Preferred Cipher Suite	<p>Select the cipher suite to use when communicating with the server from the list of available options. The cipher suite you use determines authentication and encryption settings for the network connection.</p> <p>If you use the default value, Use Runtime Default, the list of supported cipher suites is sent to the server and each is tried until a match is found.</p>
Preferred Data Integrity Algorithm	<p>Select the bulk-hashing algorithm for data integrity checks from the list of available options.</p> <p>If you use the default value, Use Runtime Default, Service Bus sends the preferred algorithm, <code>hmac-sha1</code>. If that is not supported by the server, the list of supported algorithms is sent to the server and each is tried until a match is found.</p>

Table 29-11 (Cont.) SFTP Transport Properties for Proxy Services

Property	Description
Preferred Public Key Algorithm	Select the asymmetric key algorithm for public-key cryptography from the list of available options. If you use the default value, Use Runtime Default , Service Bus sends the preferred algorithm, <code>ssh-dss</code> . If that is not supported by the server, the list of supported algorithms is sent to the server and each is tried until a match is found.
Preferred Key Exchange Algorithm	Select the default key exchange protocol for negotiating the session key for encrypting the message. If you use the default value, Use Runtime Default , Service Bus sends the preferred algorithm, <code>diffie-hellman-group1-sha1</code> . If that is not supported by the server, the list of supported algorithms is sent to the server and each is tried until a match is found.
Preferred Compression Algorithm	Select whether to compress in-flight data using zlib. Select zlib or zlib@openssh.com to compress data; otherwise select None . The default is None .

Configuring Transport Headers and Metadata

When you configure a proxy service, you can use a Transport Header action in the pipeline to set the header values in messages. The following table lists the transport header and metadata related to the SFTP transport.

Table 29-12 Transport Headers and Metadata

Header / Metadata	Description
FileName	This value is used as the file name in the destination directory.
isFilePath	This is a metadata field. The possible values are true and false. <ul style="list-style-type: none"> • True: FileName is interpreted as the absolute path of the file. • False: FileName is interpreted as the name of the file.
filePath	This is a response metadata field that indicates the absolute path at which the file specified in the FileName header is written.

Configuring Transport Headers in the Pipeline

You can configure the transport headers only for outbound requests in the pipeline. Use a transport header action to set the header values in messages. For more information, see [Adding Transport Header Actions in the Console](#).

Configuring Transports Headers and Metadata in the Test Console

You can configure the `FileName` transport header and the `isFilePath` metadata values in the Test Console when you test the SFTP transport-based services during development. For more information, see [Using the Test Console](#).

Configuring Business Services to Use the SFTP Transport

The following table describes the properties you use to configure an SFTP transport for a business service. For more information, see [Creating and Configuring Business Services](#).

Table 29-13 SFTP Transport Properties for Business Services

Property	Description
User Authentication	<p>Select the required authentication method from the following:</p> <ul style="list-style-type: none"> • Username Password Authentication: Specifies that a static service account is associated with this authentication method and the client is authenticated using the provided credentials. • Host Based Authentication: Specifies that a user name and service key provider is required to use this authentication method. Any user connecting from a known host is authenticated using the private key of the host. • Public Key Authentication: Specifies that a user name and service key provider is required to use this authentication method. Every user has their own private key. <p>Note: The Service Bus service does not use the service key provider to authenticate any credentials from the SFTP server. It uses only the <code>known_hosts</code> file to authenticate the SFTP server, as described in Configuring Transport-Level Security for SFTP Transport.</p>
Service Account	<p>Enter the service account that will be used for authentication to access the server. For information about using service accounts, see Working with Service Accounts.</p>
Service Key Provider	<p>Enter a service key provider to use for authentication.</p> <p>This field is available only for the public key and host-based authentication methods. For more information, see Working with Service Key Providers.</p>
Username	<p>Enter the user name. This value is required only when you select either the host-based or public key authentication method.</p> <ul style="list-style-type: none"> • In host-based authentication, the user name is required for polling the home directory of the user on the SFTP server. • In public key authentication, the user name is required for polling the home directory of the user and for identifying the location of the public key on the SFTP server.
Timeout	<p>Enter the socket timeout interval, in seconds, before the connection is dropped. If you enter 0, there is no timeout. The default value is 60.</p>
Prefix for destination File Name	<p>Enter an optional prefix that the transport prepends to the file name on the remote server.</p> <p>Do not enter * in this field. This character causes a runtime exception.</p>
Suffix for the destination File Name	<p>Enter an optional suffix that the transport appends to the file name on the remote server.</p> <p>Do not enter * in this field. This character causes a runtime exception.</p>

Table 29-13 (Cont.) SFTP Transport Properties for Business Services

Property	Description
Request encoding	Accept the default UTF-8 as the character set encoding for requests in SFTP transports.
Preferred Cipher Suite	Select the cipher suite to use when communicating with the server. The cipher suite you use determines authentication and encryption settings for the network connection. If you use the default value, Use Runtime Default , the list of supported cipher suites is sent to the server and each is tried until a match is found.
Preferred Data Integrity Algorithm	Select the bulk-hashing algorithm for data integrity checks from the list of available options. If you use the default value, Use Runtime Default , Service Bus sends the preferred algorithm, <code>hmac-sha1</code> . If that is not supported by the server, the list of supported algorithms is sent to the server and each is tried until a match is found.
Preferred Public Key Algorithm	Select the asymmetric key algorithm for public-key cryptography from the list of available options. If you use the default value, Use Runtime Default , Service Bus sends the preferred algorithm, <code>ssh-dss</code> . If that is not supported by the server, the list of supported algorithms is sent to the server and each is tried until a match is found.
Preferred Key Exchange Algorithm	Select the default key exchange protocol for negotiating the session key for encrypting the message. If you use the default value, Use Runtime Default , Service Bus sends the preferred algorithm, <code>diffie-hellman-group1-sha1</code> . If that is not supported by the server, the list of supported algorithms is sent to the server and each is tried until a match is found.
Preferred Compression Algorithm	Select whether to compress in-flight data using zlib. Select zlib or zlib@openssh.com to compress data; otherwise select None . The default is None .

SFTP Transport Environment Values

Environment values are predefined fields in the configuration data that are domain-specific. Their values are likely to change when you move the configuration from one domain to another (for example, from test to production). The following table lists the environment values associated with the SFTP transport.

Table 29-14 SFTP Transport Environment Values

Environment Value	Description
SFTP Archive Directory	The archive directory for a SFTP proxy service. If direct-streaming is on, the archive directory is present on the remote SFTP server; otherwise, it is present locally.
SFTP Download Directory	The download directory for a SFTP proxy service.

Table 29-14 (Cont.) SFTP Transport Environment Values

Environment Value	Description
SFTP Error Directory	The error directory for a SFTP proxy service. If direct-streaming is on, the error directory is present on the remote SFTP server; otherwise, it is present locally.
Managed Server for Polling	The managed server for polling in a clustered domain.
SFTP Preferred Cipher Suite	The cipher suite to use for authentication and encryption in SFTP proxy and business services.
SFTP Preferred Compression Algorithm	The compression library to use to compress in-flight data in SFTP proxy and business services.
SFTP Preferred Data Integrity Algorithm	The bulk-hashing algorithm to use integrity checks in SFTP proxy and business services.
SFTP Preferred Key Exchange Algorithm	The default key exchange protocol for negotiating the session key for encrypting the message in SFTP proxy and business services.
SFTP Preferred Public Key Algorithm	The asymmetric key algorithm for public-key cryptography in SFTP proxy and business services.

For more information about environment variables and the SFTP variables, see:

- Customizing Oracle Service Bus Environments in *Administering Oracle Service Bus*
- [Configuring Proxy Services to Use the SFTP Transport](#)
- [Configuring Business Services to Use the SFTP Transport](#)

Using the JEJB Transport

This chapter provides an overview of the JEJB transport and describes how to use the transport to handle Plain Old Java Objects (POJOs) in Service Bus.

This chapter includes the following sections:

- [Introduction to the JEJB Transport](#)
- [Prerequisites for Creating JEJB Services](#)
- [Use Cases](#)
- [UDDI Integration](#)
- [JEJB Transport Configuration Reference](#)

Introduction to the JEJB Transport

The JEJB transport lets you pass Plain Old Java Objects (POJOs) through Service Bus. For example, you can use an EJB to invoke a remote EJB operation or a non-EJB service, or you can invoke an EJB operation with a non-EJB request. Use case details are described in [Use Cases](#).

To a J2EE client, a JEJB proxy service looks like a stateless session bean. A JEJB proxy service, on receiving the method arguments, passes their XML representation in the pipeline `$body` variable. POJO arguments are represented as the XML fragment, which contains the location of the actual POJO stored in the object repository within the pipeline. XML arguments can either be passed by value or by reference (referencing the actual object stored in the object repository). Primitive types are always passed by value.

For more information on POJOs in pipelines, see [Java Content in the Body Variable](#) and [Using Java Callouts and POJOs](#).

The JEJB transport is always synchronous, so the messaging pattern is always request-response. For deployment, Service Bus automatically packages JEJB proxy services as enterprise archives (EARs).

Differences Between the JEJB Transport and the EJB Transport

The EJB transport, available only for business services, invokes remote EJBs through the Java Web Services (JWS) framework. The JEJB transport, which lets you invoke remote EJBs and external services with POJOs, passes POJOs directly through Service Bus to the target EJB methods using an RMI serialization/deserialization cycle. The EJB transport provides a "Support Transaction" flag, but all proxy services provide transactional support, making the transaction option unnecessary for JEJB business services.

JEJB Transport WSDL Generation

For proxy and business services, the JEJB transport generates a Document-style WSDL file with Literal encoding that is used solely for describing the message passed to the pipeline. The WSDL format lets you leverage Service Bus WSDL features such as per-operation monitoring. The message structure defined in the WSDL file may differ from the actual pipeline message at runtime if you inline your POJO arguments in the message using the `Pass XMLBeans by value` option, described in [Table 30-1](#).

Following is the behavior of the pipeline message format for XMLBeans parameter types:

Proxy Services

- **Request Parameters:** Request parameters in the pipeline message refer to an inline XML object if `Pass XMLBeans by value` is true; otherwise the reference is to `java-content ref`.
- **Response Parameter:** The response may refer to an inline XML object or the `java-content ref`, as the response may come in either form from the business service.

Business Services

- **Response Parameter:** Return parameters in the pipeline message refer to an inline XML object if `Pass XMLBeans by value` is true; otherwise the reference is to `java-content ref`.
- **Request Parameters:** Request method parameters in the pipeline message may refer to an inline XML object or the `java-content ref`, as the request may come in either form from the proxy service.

JEJB Transport Error Handling

This section describes how the JEJB transport handles errors.

- [Exception Propagation in the Response](#)
- [Application and Connection Errors](#)

Exception Propagation in the Response

The JEJB transport stores request exceptions in the object repository and propagates them to the JEJB proxy service through the `$fault` variable. The `$fault` variable would contain the location of the exception instance within the `<java-exception>` `<java-content ref="jcid"/>` `</java-exception>` element, where `jcid` is the reference to the exception instance stored in the object repository.

To propagate the user exception to the client, the JEJB proxy service expects the response in one of the following formats. In all cases, `jcid` is a reference to the error in the object repository.

- `env:Envelope/env:Fault/detail/mc:java-exception`

```
<detail>
  <mc:java-exception>
    <mc:java-content ref="jcid"/>
  </mc:java-exception>
  ...
</detail>
```


- `env:Envelope/env:Fault/detail/mc:fault/mc:java-exception`

```
<detail>
  <mc:fault xmlns:mc="http://www.bea.com/wli/sb/context">
    <mc:java-exception>
      <mc:java-content ref="jcid"/>
    </mc:java-exception>
    ...
  </mc:fault>
</detail>
```

- `env:Envelope/ env:Fault/detail/mc:fault/mc:details/ con1:ReceivedFaultDetail/ con1:detail/mc:java-exception`

```
<con:details>
  <con1:ReceivedFaultDetail
    xmlns:con1="http://www.bea.com/wli/sb/stages/transform/config">
    <con1:faultcode>soapenv:Server</con1:faultcode>
    <con1:faultstring>checkExceptionConversion</con1:faultstring>
    <con1:detail>
      <mc:java-exception>
        <mc:java-content ref="jcid"/>
      </mc:java-exception>
    </con1:detail>
  </con1:ReceivedFaultDetail>
</con:details>
```

If you want to raise your own exceptions to return to the caller, raise them in a Java Callout in the pipeline.

If you configure a Java Callout or Service Callout in an error handler for "Reply with Failure," format the \$body so that it conforms to one of the previously described fault structures.

Application and Connection Errors

This section describes the conditions under which the JEJB transport throws application and communication errors, which are subject to the retry configuration of a service.

- [Connection Errors](#)
- [Application Errors](#)

Connection Errors

The JEJB transport throws connection errors in the following situations:

- `NamingExceptions` looking up the EJBs raised during the remote call.
- A runtime or remote exception is thrown, but the ongoing transaction has *not* been set as `rollback-only`, signifying that the error occurred before the invocation of the EJB container.

Application Errors

The JEJB transport throws application errors in the following situations:

- A runtime or remote exception is thrown and the ongoing transaction has been set as `rollback-only` (likely by the EJB container), signifying the EJB container has been reached and a fatal error either occurred within the container or within the EJB itself.

- Business exceptions defined in the EJB business interface.
- An exception caused by a faulty encoding of the parameters in XML.

Prerequisites for Creating JEJB Services

This section provides information you need to know before you create JEJB proxy and business services.

- [Creating and Packaging Your Client EJB JAR File](#)
- [Registering a JNDI Provider Resource \(Business Services\)](#)

Creating and Packaging Your Client EJB JAR File

When you use JEJB services, you need to create and package POJOs to represent EJB invocations and operations for the JEJB proxy and business services. Use the following guidelines.

- Define an interface of type `java.io.Serializable` and include any necessary helper classes, such as business exceptions. The interface does not need to extend any class as long as the interface is valid for one of the RMI protocols described in [JEJB Transport Endpoint URI](#), or is valid for JMS messages if you are using JMS to invoke EJBs.

Though not required, you can make the interface a remote interface as defined by the EJB 2.1 specification or annotate methods with the `javax.ejb.Remote` annotation to designate it as an EJB 3.0 business interface. For a simple POJO interface (no EJB remote interface) or an interface annotated with `javax.ejb.Remote`, the JEJB transport provider generates the 3.0 EJB interface out of the JEJB proxy service. For a remote interface, the JEJB transport provider generates the 2.1 EJB interface out of the JEJB proxy service.

- The objects received as arguments must be passable to any required classes in a Java Callout archive resource.
- An array of any type is considered a POJO.
- To avoid unnecessary serialization/deserialization cycles, do not duplicate the JAR files uploaded as Service Bus archive resources to support Java callouts. Package all archive resource classes in a single JAR file so multiple Java callouts do not serialize/deserialize the objects.
- Package your interface and dependent classes in a single client JAR file and import it into Service Bus. While this is the client JAR file you will select when configuring a service, it is not technically a fully expanded EJB client JAR file because it contains no stubs. The actual bean (hence WebLogic Server stub generation) does not exist until a JEJB proxy service is created and activated.

Registering a JNDI Provider Resource (Business Services)

A JNDI provider resource allows you to specify the communication protocols and security credentials used to retrieve EJB stubs bound in the JNDI tree of remote Oracle WebLogic Server domains. Typically, the target EJB is not located in the same domain as Service Bus. In this case, you must register a JNDI provider resource. When the EJB is located in the same domain, you can define a provider to specify credentials and take advantage of stubs caching, though doing so is optional.

The JNDI provider has a high performance caching mechanism for remote connections and EJB stubs. The preferred communication protocol from Service Bus to an Oracle WebLogic Server domain is `t3` or `t3s`. If messages need to go through a firewall, you can use HTTP tunneling.

Note:

Although it is possible to use a WebLogic Server Foreign JNDI provider, Oracle recommends that you do not.

The transport does not support two-way SSL or client certificate to look-up the JNDI tree or access a method on an EJB.

For information about registering and configuring a JNDI provider resource in Service Bus, see [Working with JNDI Provider Resources](#).

Use Cases

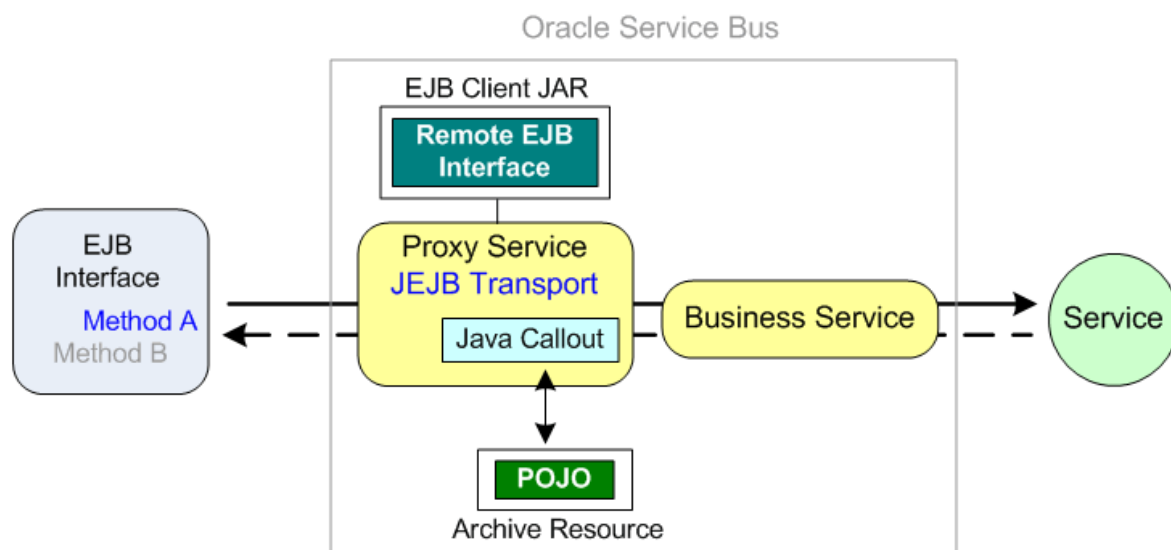
Following are the supported use cases for using the JEJB transport in proxy and business services. Each use case provides implementation guidelines for you to use in conjunction with the general service configuration, as described in [JEJB Transport Configuration Reference](#).

- [EJB Invoking an External Service](#)
- [Non-EJB Client Invoking an EJB](#)
- [EJB Invoking EJB](#)

EJB Invoking an External Service

You can invoke an external service with an EJB through Service Bus, as illustrated by [Figure 30-1](#).

Figure 30-1 An EJB Invoking an External Service



In [Figure 30-1](#), the JEJB proxy service serves as a stateless session bean to the EJB client interface. The JEJB transport provider for the proxy service generates a stateless session EJB from the remote/business interface in the client JAR and the pipeline, then deploys it as an EAR at the JNDI address specified in the endpoint URI.

Note:

Be sure to install policies that protect the JNDI entries from being modified.

The EJB makes a call to a remote interface provided by the proxy service EJB client JAR, passing transaction and security details to the proxy service as well.

The EJB client interface is a POJO with method arguments that the JEJB transport provider represents as a WSDL file and passes into the proxy service \$body variable as XML. You can introspect the \$body content to transform the message into the required format to pass to the business service and invoke the external service. The actual POJO is stored in the object repository, and the XML in the \$body references it with a <code><java-content ref=""></code> element.

In the response, provide a Java callout that converts the response to an EJB return format that is passed to the calling EJB method. View the proxy service's generated WSDL file to see the expected message format.

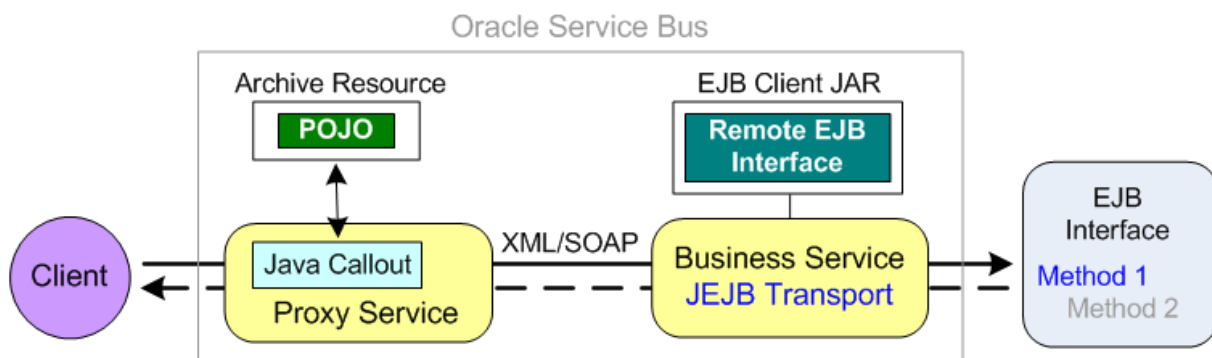
Note:

In the proxy service pipeline, you can pass POJO arguments to Java callouts to a business service or to another proxy service using, for example, a service callout or a publish action.

Non-EJB Client Invoking an EJB

You can invoke an EJB with a non-EJB client through Service Bus, as illustrated in [Figure 30-2](#).

Figure 30-2 A Non-EJB Client Invoking an EJB



In [Figure 30-2](#), a non-EJB client makes a call to a proxy service configured with a transport that matches the request; for example, a JMS proxy service making an invocation with a JMS topic or queue.

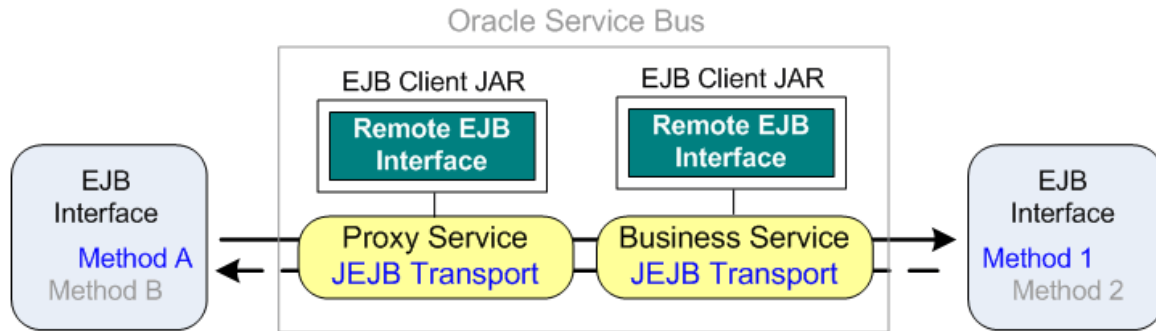
You configure a Java callout in the request, which converts the request into an XML representation of an EJB call in the \$body variable. Put operations in the \$operation variable. View the business service's generated WSDL file to see the expected message

format. The JEJB business service uses its generated WSDL file to map the incoming message to the EJB remote interface and invoke the remote EJB method directly.

EJB Invoking EJB

You can invoke an EJB with an EJB through Service Bus, as illustrated in [Figure 30-3](#).

Figure 30-3 An EJB Invoking an EJB



In [Figure 30-3](#), the EJB call is passed through the proxy and business services to invoke another EJB method. Rather than making a direct RMI call outside of Service Bus, this architecture lets you leverage Service Bus features such as message routing, UDDI integration, alerts, monitoring, reporting, and result caching.

The JEJB transport provider for the proxy service generates a stateless session EJB from the remote/business interface in the client JAR file and the pipeline, and then deploys it as an EAR file at the JNDI address specified in the endpoint URI.

At runtime the JEJB proxy service receives a POJO as method argument, stores it in the object repository, and generates an XML representation of the POJO in the proxy service `$body` variable according to the generated proxy service WSDL file. The proxy service passes the message to the business service, and the business service uses its generated WSDL file to map the message to the remote interface and invoke the remote method directly.

UDDI Integration

You can publish and import JEJB proxy service properties to and from UDDI registries.

- [UDDI Publish](#)
- [UDDI Import](#)

UDDI Publish

JEJB proxy services publish the following properties to a UDDI registry:

- URI
- EJB Spec Version
- Client JAR
- Home Interface (for EJB 2.1 only)
- Remote Interface (the Business Interface for EJB 3.0)

- **Method names.** The following are not included: operation aliases, parameters, and return details. Method names are passed in one property with all the method signatures appended. Method signatures are separated by the pound (#) character.

UDDI Import

This section describes how the JEJB transport handles service import from a UDDI registry.

- **URI:** The JEJB transport provider attempts to match the host and port information from the URI property in the UDDI registry with a JNDI provider resource registered on the server.

If the transport provider cannot find a JNDI provider, the import fails. However, if no JNDI provider is found but the host and port match the localhost IP and listen port, the resulting business service will be local (no JNDI provider).

- **Client JAR:** The transport provider downloads the client JAR files and, if the manifest classpath exists in the JAR files, creates the corresponding JAR resources in the matching directory structure. The first URI in the list is the root client JAR file. If no manifest classpath exists in the JAR files, you must manually add the resource JAR files as dependencies to the root JAR file. If a resource in the imported client JAR file has the same name as another resource in the domain, the imported resource overwrites the existing resource.

Make sure that the client JAR file you are importing does not already exist in your domain.

- **Method Names:** Methods included in the corresponding property are automatically selected in the endpoint configuration. All the other methods are marked as excluded.

JEJB Transport Configuration Reference

This section provides information about endpoint URI formats, environment values, and configuring the JEJB transport in proxy and business services.

- [JEJB Transport Endpoint URI](#)
- [Configuring Proxy Services to Use the JEJB Transport](#)
- [Configuring Business Services](#)
- [JEJB Transport Environment Values](#)

JEJB Transport Endpoint URI

The format for the endpoint URIs depend on whether you are configuring a proxy service or a business service.

Note:

JEJB services do not support co-located calls.

- [Proxy Service JEJB Endpoint URI](#)
- [Business Service JEJB Endpoint URI](#)

Proxy Service JEJB Endpoint URI

The URI configured for a JEJB proxy service becomes the global JNDI name for locating the stateless session bean generated by the JEJB transport from the remote/business interface in the client JAR. The URL format is `ejb_jndi_name`.

Note:

For EJB 3.0, `ejb_jndi_name` is the `mappedName` attribute of the `@javax.ejb.Stateless` annotation in the generated bean. The lookup JNDI name for the generated EJB service is suffixed with `#interface_class`, which is the fully qualified name of the business interface.

You can access the JEJB proxy service as:

- **EJB 2.1:** `protocol://host:port/ejb_jndi_name`
- **EJB 3.0:** `protocol://host:port/ejb_jndi_name#interface_class`

The `protocol` can be one of the following RMI protocols:

- `iiop/iiops`: For generic, server-agnostic use.
- `t3/t3s`: For use with Oracle WebLogic Server.
- `http/https`: For tunneling and use with Oracle WebLogic Server.

For example:

- **EJB 2.1:** `t3://localhost:7001/osb.jejb.myJejbProxy`
- **EJB 3.0:** `t3://localhost:7001/osb.jejb.myJejbProxy#com.example.MyEjb3`

Business Service JEJB Endpoint URI

Use the following endpoint URI format for a JEJB business service:

`jejb:jndi_provider_name:ejb_jndi_name`

The `jndi_provider_name` is the remote JNDI context, and the `ejb_jndi_name` is the remote EJB's JNDI name.

For example:

- **EJB 2.1:** `jejb:myProvider:osb.jejb.myJejbBiz21`
- **EJB 3.0:** `jejb:myProvider:myBiz31#osb.jejb.myJejbBiz`
where `#osb.jejb.myJejbBiz` is the fully qualified business interface.

If your EJBs are running on IBM WebSphere, `ejb_jndi_name` must be in the on of following formats:

- `cell/nodes/node_name/servers/server_name/ejb_jndi_name`
- or

- `cell/clusters/cluster_name/ejb_jndi_name`

For more information, refer to the IBM WebSphere documentation.

Configuring Proxy Services to Use the JEJB Transport

The following table describes the properties you use to configure a JEJB transport for a proxy service. For instructions on creating a proxy service, see [Creating and Configuring Proxy Services](#).

Table 30-1 JEJB Transport Properties for Proxy Services

Property	Description
Dispatch Policy	Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists. For information about Work Managers, see: <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>
EJB Spec Version	Select the EJB version of the remote EJB interface.
Pass XMLBeans by value	Select this option if you want the transport to generate an "inlined" XML representation of POJO arguments (an XMLObject) whose parameters you can access and manipulate with XQuery expressions. Note: Type information is not available inline for XMLObjects passed by value. If you use this option, you cannot pass the typed XMLObject as the argument in a Java Callout in a proxy service pipeline. Do not select this option if you want to pass the POJO by reference, which also results in better performance. For more information, see JEJB Transport WSDL Generation .
Transaction Attribute	Select one of the following options for handling transactions: <ul style="list-style-type: none"> • Supports: The transport accepts an incoming transaction. Quality of service is exactly-once if the operation is invoked in a transaction and best-effort if the operation is invoked outside of a transaction. • Required: The transport accepts an incoming transaction. If no ongoing transaction exists, the transport starts one. Quality of service is exactly-once. • RequiresNew: The transport always starts a new transaction, suspending an ongoing transaction. Quality of service is exactly-once. • Mandatory: The transport invokes the method in the existing transaction. Quality of service is exactly-once. • NotSupported: The transport suspends an existing transaction and resumes it on invocation. Quality of service is best-effort. • Never: The transport does not invoke the method in a transaction. Quality of service is best-effort.
Remote Client Timeout	Specify the length of time in seconds that a remote RMI client will wait before timing out.

Table 30-1 (Cont.) JEJB Transport Properties for Proxy Services

Property	Description
Client JAR	Click Browse and select an EJB client JAR resource from the list displayed. The client JAR contains the remote or business interface for the remote EJB. The client JAR is registered as a generic archive resource.
Home Interface	For EJB 2.1 only , select the required EJBHome interface from the options populated by the client JAR.
Remote Interface	For EJB 2.1 only , this field is automatically populated based on the configuration of the home interface.
Business Interface	For EJB 3.0 only , select the business interface from the client JAR file that you want to invoke.
Target Namespace	The target namespace of the generated WSDL file. This field is automatically populated by information picked up from the JAR.
Methods	Select the required methods from the list of available methods. The available methods depend on the JAR file being used. By default, all methods are selected. Expand a method to edit the default parameter values. You can change the default operation name for a given method. By default, the operation name is the method name. If an EJB contains methods with same name (overloaded), you must change the operation names so that they are unique. WSDL files require unique operation names.

Configuring Business Services

The following table describes the properties you use to configure a JEJB transport for a business service. For more information, see [Creating and Configuring Business Services](#).

Table 30-2 JEJB Transport Configuration for Business Services

Option	Description
Dispatch Policy	Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists. For information about Work Managers, see: <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>
EJB Spec Version	Select the EJB version of the remote EJB interface.

Table 30-2 (Cont.) JEJB Transport Configuration for Business Services

Option	Description
Pass XMLBeans by value	<p>Select this option if you want the transport to generate an "inlined" XML representation of POJO arguments (an XMLObject) whose parameters you can access and manipulate with XQuery expressions.</p> <p>Note: Type information is not available inline for XMLObjects passed by value. If you use this option, you cannot pass the typed XMLObject as the argument in a Java Callout in a proxy service pipeline.</p> <p>Do not select this option if you want to pass the POJO by reference, which also results in better performance.</p> <p>Do not select this option if you want to pass the POJO by reference, which also results in better performance.</p> <p>For more information, see JEJB Transport WSDL Generation.</p>
Pass Caller's Subject	<p>Select this option as an alternative to selecting a service account. When you select this option, Service Bus passes the authenticated subject from the proxy service when invoking the EJB.</p>
Service Account	<p>Enter a service account that will be used for authentication to access the service. If no service account is specified, an anonymous subject is used. This field is required if you selected Basic authentication.</p> <p>For more information, see Working with Service Accounts.</p>
Client JAR	<p>Click Browse and select an EJB client JAR resource from the list displayed. The client JAR contains the remote or business interface for the remote EJB. The Client JAR is registered as a generic Archive Resource.</p>
Home Interface	<p>For EJB 2.1 only, select the required EJBHome interface from the options populated by the client JAR.</p>
Remote Interface	<p>For EJB 2.1 only, this field is automatically populated based on the configuration of the home interface.</p>
Business Interface	<p>For EJB 3.0 only, select the business interface from the client JAR file that you want to invoke.</p>
Target Namespace	<p>The target namespace of the generated WSDL file. This field is automatically populated by information picked up from the JAR.</p>
Methods	<p>Select the required methods from the list of available methods. The available methods depend on the JAR file being used. By default, all methods are selected. Expand a method to edit the default parameter values.</p> <p>You can change the default operation name for a given method. By default, the operation name is the method name. If an EJB contains methods with same name (overloaded), you must change the operation names so that they are unique. WSDL files require unique operation names.</p>

JEJB Transport Environment Values

The JEJB transport stores the following environment values for JEJB services:

- Service URI
- Work Manager
- UDDI Auto Publish (Proxy Services)
- Service Account (Business Services)

These values correspond to transport properties in business and proxy services. For more information, see [JEJB Transport Configuration Reference](#).

Using the JMS Transport

This chapter provides an overview of the JMS transport and describes how to use and configure it in your Service Bus services. It also describes features and concepts related to interoperability between Service Bus and WebLogic JMS and between Service Bus and WebSphere MQ.

This chapter includes the following sections:

- [Introduction to the JMS Transport](#)
- [Using SOAP Over JMS Transport](#)
- [Naming Guidelines for Domains, Servers, and URIs](#)
- [JMS Client ID in Proxy Services](#)
- [JMS Transport Error Handling](#)
- [WSDL-Defined SOAP Fault Messages](#)
- [Message ID and Correlation ID Patterns for JMS Request/Response](#)
- [JMS Transport Configuration Reference](#)

Introduction to the JMS Transport

The JMS transport lets you send and receive messages from JMS queues and topics of a JMS service. You enqueue messages when you configure a business service to use the JMS transport, and you read (or poll) messages when you configure a proxy service to use the JMS transport. The JMS queues or topics can reside in a local WebLogic Server or on a remote server.

JMS is a standard API for accessing enterprise messaging systems. For an overview and features of WebLogic JMS, see *Overview of JMS and WebLogic Server* in *Administering JMS Resources for Oracle WebLogic Server*.

JMS Content Type for Services

To support interoperability with heterogeneous endpoints, Service Bus allows you to control the content type used, the JMS type used, and the encoding used when configuring message flows. The JMS type can be byte or text for non-Java-type messages. For more information, see [Content Types, JMS Type, and Encoding](#).

JMS Transport Security

The JMS transport supports one-way SSL, but not two-way SSL.

Asynchronous Request-Response Messaging

Messaging can be one-way, synchronous request-response, or asynchronous request-response. However, messaging over JMS is only one-way or is asynchronous request-response. Asynchronous request-response messaging using JMS is an alternative to messaging using HTTP or HTTP(S).

Using asynchronous request-response messaging has the following advantages:

- The request thread does not get blocked while waiting for the response. This removes a thread management issue that can occur when numerous blocking request-response invocations are made. However, HTTP and HTTP(S) support a nonblocking mode of operation.
- The messaging is more reliable than HTTP because it can be:
 - Persisted on disk
 - Queued when the service is not available
 - Re-delivered if the server has an error or fails when the message is being processed
 - Transactional

For IBM WebSphere MQ, asynchronous request-response messages may be the best approach for interacting with some mainframes. The asynchronous service must echo the correlation ID or the message ID depending on the JMS request-response pattern that you use. The format of either ID used by Service Bus is compatible with IBM WebSphere MQ and with target services that use MQ native interfaces. For more information, see [Message ID and Correlation ID Patterns for JMS Request/Response](#).

Asynchronous request-response messages are handled by the outbound and inbound transports. That is, the message flow, except for the `$outbound` and `$inbound` transport specific data, does not distinguish between JMS request-response and HTTP request-response.

Service Bus supports bridging between synchronous and asynchronous request and response. For example, a proxy service can be invoked using HTTP, and the proxy service routes to a JMS request-response business service. This is called synchronous-to-asynchronous service switching.

Sending and Receiving Java Objects in Messages

You can directly send Java Objects using the JMS transport. To enable Java Object support in the request or response, create a proxy or business service of type **Messaging Service**, and select **Java** on the Messaging page for the request or response, depending on whether you are sending or receiving the Java Objects through Service Bus.

Dequeuing a Java object message from the JMS destination involves de-serializing the Java object. For this to work, you must package the Java classes for the Java objects that are to be dequeued into a JAR file and import the JAR into your Service Bus project. Then, in the JMS transport-specific configuration page for a service, select the JAR in the **Client Jar** field. The Client Jar field is available in JMS proxy services when you select Java as the message type for the request, and in business services when you select Java as the message type for the response.

Java Objects in Service Bus are stored in the pipeline object repository and referenced in the SOAP body by a `<java-content ref="jcid" />` element and attribute, where `jcid` is the key to the object in the object repository. If a Java Object is null, the object is represented in the pipeline as `ref="jcid:null"`.

Only one Java Object is allowed in each message. For Java-type message types, Service Bus does not support large messages (content streaming) or testing Java-type services in the Test Console.

Required JMS Resources

In addition to configuring JMS file stores in the Oracle Fusion Middleware Configuration Wizard, proxy services and business services that use JMS require configuration of the following resources:

- JMS connection factories. You must configure XA or non-XA JMS connection factories for all business services and proxy services implemented using JMS.
- JMS queues/topics. Service Bus automatically configures JMS queues for proxy services that are implemented using JMS. You must configure JMS queues/topics for all business services using JMS and for any proxy services that are implemented using non- JMS.

If you want to concentrate all Service Bus JMS resources in a single JMS module, use the Oracle WebLogic Server Administration Console to create a new JMS module containing the destination to be used for the proxy services' endpoint. For more information about configuring JMS resources, see "Methods for Configuring JMS Resources" in the *Administering JMS Resources for Oracle WebLogic Server*.

Platform Interoperability

The following sections provide information and links for interoperability with different JMS platforms.

- [Interoperability with WebLogic JMS](#)
- [Interoperability with WebSphere MQ](#)
- [Interoperability with Tibco EMS](#)

Interoperability with WebLogic JMS

For information about WebLogic Server JMS, see the following topics:

- "Managing Your Applications" in *Developing JMS Applications for Oracle WebLogic Server*
- "Configure JMS Servers" in the *Oracle WebLogic Server Administration Console Online Help*

Note:

Service Bus supports the MQ Extended Transactional Client, which is vital for remote transactional support configuration.

Interoperability with WebSphere MQ

Service Bus connects to WebSphere MQ through the WebSphere MQ JMS interface, so Service Bus is a WebSphere MQ JMS client. WebSphere MQ can interface with Service Bus in the following ways:

- Service Bus acts as the front-end of WebSphere MQ to accept service requests from other applications and converts them to WebSphere MQ requests.
- WebSphere MQ sends messages to other applications through Service Bus.

For more information, see [Using the WebSphere MQ JMS Interface](#).

Interoperability with Tibco EMS

When using Tibco Enterprise Messaging Service (EMS) as a messaging provider, you must modify `setDomainEnv.cmd/sh` so `EXT_PRE_CLASSPATH` references the Tibco EMS client JAR files.

Using SOAP Over JMS Transport

You can use JMS for SOAP transport instead of HTTP, because SOAP is transport-agnostic. Service Bus supports SOAP messages with JMS request-response, and supports interoperability with WebLogic Server SOAP-based clients and services. JMS is also an approach for reliable messaging.

Interoperating with WebLogic Server

When you configure JMS resources in WebLogic Server, you use the following SOAP-JMS URI format in WebLogic Server:

```
jms://host:port/contextURI/serviceName?URI=destJndiName
```

When you configure the service in Service Bus, the URI must have the following format:

```
jms://host:port/connection_factory/jndi_destination
```

Both formats use the same `jndi_destination`. The `jndi_destination` must be the JNDI name of an existing `QueueConnectionFactory` in the target WebLogic Server. For more information, see "WebLogic Server Messaging" in *Understanding Oracle WebLogic Server*. This document provides an overview of JMS and links to more information.

Note:

While WebLogic Server allows forward slashes in JNDI names, such as "myqueues/myqueue", JNDI names with forward slashes interfere with the URI format required by Service Bus, and you cannot use those names. To work around this issue, define a JMS foreign server and reference that foreign server in the URI. For more information, see "Configure foreign servers" in the *Oracle WebLogic Server Administration Console Online Help*.

When you invoke WebLogic Server services from Service Bus, you must set the URI as a JMS property with the value as `/contextURI/serviceName` in the pipeline on the outbound variable (`$outbound`) before a request is sent to the business service. Use

the Transport Headers action to set this property. For information about setting `$outbound`, see [Inbound and Outbound Variables](#).

When a WebLogic Server web services client invokes a Service Bus proxy service, the URI property is ignored. However, it can be passed through to an invoked service using the pass through options of the Transport Headers action. For more information, see [Adding Transport Header Actions in the Console](#).

Service Bus can only invoke Oracle WebLogic request-response services running on version 9.2 or later. However, it can also invoke one-way JMS services.

Configuring the Response Queues for Cross-Domain JMS Calls

When you configure the response queue for cross-domain JMS calls, make sure that there is a separate response queue corresponding to each requesting Managed Server.

For example, two Service Bus clustered domains (domain A and domain B) are communicating with a WebLogic Server domain that has two Managed Servers. Domain A has three Managed Servers and domain B has four Managed Servers. You need to configure seven distinct queues to serve as response queues ($3 + 4 = 7$) on the WebLogic Server domain for sending responses back to domain A and domain B. These seven queues could be distributed queues (with members on both the Managed Servers of the WebLogic Server domain).

Note:

When the JMS requests come from multiple proxy services hosted by different remote domains, you must configure the back-end domain hosting the JMS business service with the separate sets of response queues corresponding to each requesting domain.

Naming Guidelines for Domains, Servers, and URIs

If you are working with more than one domain, make sure that your configuration conforms to the following requirements:

- WebLogic Server instances and domain names are unique.
- WebLogic JMS server names are unique across domains.
- If a JMS file store is being used for persistent messages, the JMS file store name must be unique across domains.

JMS Server Names

Note the following rules for JMS Server names:

- You cannot have duplicate JMS server names within the same domain. If you do, when messages are sent to a destination at a particular JMS server, Service Bus cannot determine the server to which the message should be sent.
- If you use Store and Forward (SAF), duplicate JMS Server names in different domains do not pose a problem.
- In the case of cross-domain communication, duplicate JMS server names can be a problem when you use the `ReplyTo` function. The `ReplyTo` message sent from a

given domain is returned to the JMS server on the same domain that received the message instead of being returned to the domain that sent the original message.

For more information on how to configure and manage WebLogic JMS:

- "Managing Your Applications" in *Developing JMS Applications for Oracle WebLogic Server*
- "Configure JMS Servers" in the *Oracle WebLogic Server Administration Console Online Help*

JNDI Names and Service Bus

While Oracle WebLogic Server allows forward slashes in JNDI names, such as "myqueues/myqueue", JNDI names with forward slashes interfere with the endpoint URI format required by Service Bus, and you cannot use those names. To work around this issue, define a JMS foreign server and reference that foreign server in the endpoint URI. For more information, see "Configure Foreign Servers" in the *Oracle WebLogic Server Administration Console Online Help*.

JMS Client ID in Proxy Services

The JMS client ID in a proxy service is the `jms-client-id` descriptor value, which is used to generate subscriber names and the client ID value for the topic's subscribers. When you configure a JMS proxy service with a topic destination, you can specify the JMS client ID to use when generating subscriber names and the subscriber client ID. This makes it easier to identify the subscriber and its corresponding proxy service when monitoring and managing the service and topic at runtime. Note that this is only effective for durable subscriptions.

About the Client ID and Subscriber Name

A subscription for a topic destination can be identified by the client ID and subscriber name. These values are generated for different `TopicMessagesDistribution` configurations using the following descriptors.

- `jms-client-id`: The value configured for the **JMS Client ID** property on a JMS proxy service targeted to a topic destination.
- `ejb-name`: The name of the MDB generated by Service Bus for the proxy service. This is a unique ID value.
- `DomainName`: The name of the Service Bus domain.
- `ServerName`: The name of the runtime Service Bus server.
- `UniqueKey`: The unique key generated by WebLogic Server when the `generate-unique-client-id` descriptor is set to true. The `generate-unique-client-id` descriptor is always set to true for MDBs deployed by JMS proxy services.

The following table shows how the subscriber client ID and name are generated for different distribution modes.

Topic Messages Distribution	Subscriber Client ID	Subscriber Name
One Copy Per Application	jms-client-id	ejb-name
One Copy Per Server	{jms-client-id}_{DomainName}_{ServerName}	ejb-name
Compatibility	{jms-client-id}_{DomainName}_{ServerName}_{UniqueKey}	Same as client ID

Additional descriptors affect the subscriber client ID. The `distributed-destination-connection` descriptor is always set to **local**, and the `generate-unique-client-id` descriptor is always set to **true**. For more information, see "Topic Subscription Identifiers" in *Programming Message-Driven Beans for Oracle WebLogic Server*.

Recommended Usage

It is not recommended to use the Topic Messages Distribution mode of Compatibility. Instead, use either One Copy Per Application or One Copy Per Server for better scalability (depending on your requirements). Compatibility mode is provided primarily for backward compatibility, and is typically used for durable subscriptions where cleanup activity might be done on topic subscribers related to a specific proxy service.

JMS Transport Error Handling

Configure JMS-transport business services to handle application and communications errors as follows:

- [Application Errors](#)
- [Communication Errors](#)
- [Pipeline Exceptions with Java Objects](#)

Application Errors

You can specify whether or not to retry business service endpoint URIs when application errors occur. For more information, see "Managing and Monitoring Endpoint URIs for Business Services" in *Administering Oracle Service Bus*.

An application error occurs when for a JMS business service configured with request/response, the `System.getProperty("com.bea.wli.sb.transports.jms.ErrorPropertyName", "SERVER_ERROR")` property is `true` in the response message. In this scenario, the JMS transport provider calls the error method with the `TRANSPORT_ERROR_APPLICATION` error code.

Communication Errors

You can configure business service URIs to be taken offline when such communication errors occur. When you configure the operational settings for the business service, you can enable the business service endpoint URIs to be taken offline after the specified

retry interval. For more information, see "Managing and Monitoring Endpoint URIs for Business Services" in *Administering Oracle Service Bus*.

Connection errors occur when any JMS exceptions or naming exceptions occur during any of the following activities:

- Looking up a JMS connection factory.
- Creating a JMS connection from a JMS connection factory.
- Creating a JMS session using a connection object.
- Looking up of a JMS destination.
- Creating a sender from the session object.
- Sending a JMS message using the sender and destination object.

Pipeline Exceptions with Java Objects

After an exception occurs in a JMS proxy service, such as while routing a Java Object message to a business service or to a Java Callout, the message payload must be properly formed so that the proxy service can access the exception instance and return it to the calling client. Make sure the payload conforms to the following format:

```
<soap:Body xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <ctx:java-content ref="key1"
    xmlns:ctx="http://www.bea.com/wli/sb/context" />
</soap:Body>
```

where `key1` is the key to the Java Object in the object repository. If the payload is not in this format, the pipeline passes a null payload to the inbound JMS transport.

Using an Error Handler

You can catch pipeline errors involving Java Objects using an error handler. The `$fault` variable in the error handler contains the reference to the exception instance ("java-exception" element).

In situations where the `$fault` variable does not contain a reference to the exception instance, you can use a Java Callout within the error handler that uses the available `$fault` information to generate an exception instance in the previously described `$body` payload format. You must use a Reply with Failure action so that the `$body` is made available as the payload to the inbound JMS transport.

WSDL-Defined SOAP Fault Messages

When consuming a WSDL file that explicitly defines a fault, the WebLogic clientgen tool generates a subclass of `java.lang.Exception` for the XML fault type. When the WebLogic Server JAX-RPC stack inspects a SOAP response message and determines that the response message contains a SOAP fault, it tries to map the fault to a clientgen-generated exception Java class.

For example, if a WSDL file contains the definitions shown in the following listing, the clientgen tool generates a Java class `com.bea.test.TheFaultType` that extends `java.lang.Exception`. A JAX-RPC client can catch `com.bea.test.TheFaultType` when invoking the related method of the service stub.

Example - Sample WSDL Definitions

```

<definitions ... xmlns:s0="http://www.oracle.com/test/">
  ...
  <types>
    <xsd:schema targetNamespace="http://www.oracle.com/test/">
      ...
      <xsd:complexType name="theFaultType">
        <xsd:sequence>
          <xsd:element name="ID" type="xsd:int" />
          <xsd:element name="message" type="xsd:string" />
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="theFault" type="theFaultType" />
    </xsd:schema>
  </types>
  ...
  <message name="theFaultMessage">
    <part element="s0:theFaultPart" name="theFault" />
  </message>
  ...
  <binding ...>
    <operation ...>
      <soap:operation soapAction="..." style="document" />
      <input ...>
        ...
      </input>
      <output ...>
        ...
      </output>
      <fault ...>
        <soap:fault name="theFaultPart" use="literal" />
      </fault>
    </operation>
  </binding>
  ...
</definitions>

```

The SOAP message must contain a fault of the correct format so the JAX-RPC stack throws the correct exception. See [Adding a Fault in a SOAP Message if the Fault is Constructed from inside a Service Bus Pipeline](#) for details.

Adding a Fault in a SOAP Message if the Fault is Constructed from inside a Service Bus Pipeline

The SOAP message must contain a fault of the correct format so the JAX-RPC stack throws the correct exception.

If the fault is constructed from inside a Service Bus pipeline, you must do the following:

1. Replace the node for the \$body variable with the following sample SOAP:

Example - Sample SOAP

```

<soap-env:Body>
  <soap-env:Fault>
    <faultcode>
      xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">soap:Server</faultcode>
    <faultstring>Some literal string</faultstring>
    <detail>
      <test:theFault>
        <test:ID>Any user defined code</test:Id>
      </test:theFault>
    </detail>
  </soap-env:Fault>
</soap-env:Body>

```

```
        <test:message>A specific literal message</test:message>
      </test:theFault>
    </detail>
  </soap-env:Fault>
</soap-env:Body>
```

where:

- `soap-env` is the system prefix for the namespace `http://schemas.xmlsoap.org/soap/envelope/`
- `test` is the prefix for the namespace `http://www.oracle.com/test/`
If the prefix `test` is not already known to Service Bus, you must declare it.

2. Configure a reply action with failure.

For information about configuring reply actions, see [Adding Reply Actions in the Console](#).

The clientgen tool is used to generate the client-side artifacts, such as the JAX-RPC stubs, needed to invoke a web service. See "Ant Task Reference" in *WebLogic Web Services Reference for Oracle WebLogic Server*.

Message ID and Correlation ID Patterns for JMS Request/Response

This section describes the Message ID and Correlation ID patterns supported in Service Bus for JMS request-response and also describes how Service Bus uses these patterns to interoperate with Java API for Remote Procedure Call (JAX-RPC) web services. Examples are also provided.

- [Overview of JMS Request-Response and Design Patterns](#)
- [JMS Message ID Pattern](#)
- [JMS Correlation ID Pattern](#)
- [Comparison of Message ID and Correlation ID Patterns](#)
- [Interoperating with JAX-RPC Over JMS](#)
- [JMS Message ID Pattern Examples](#)

Overview of JMS Request-Response and Design Patterns

Messaging provides high-speed, asynchronous, program-to-program communication with guaranteed delivery and is often implemented as a layer of software called Message Oriented Middleware (MOM). In enterprise computing, messaging makes communication between processes reliable, even when the processes and the connection between them are not so reliable. Processes may need to communicate for the following reasons:

- One process has data that needs to be transmitted to another process.
- One process needs to remotely invoke a procedure in another process.

Asynchronous request-response messaging is the best approach to interacting with some mainframes if you are using IBM WebSphere MQ.

In cluster domains and request-response scenarios, JMS Transport deploys the MDBs to individual managed servers of the cluster. But in dynamic cluster domains, because a dynamic server cannot be individually targeted for any application, JMS Transport can only deploy the MDBs to a configured managed server for a request-response scenario. For more information about this limitation of dynamic clusters, see *Limitations and Considerations When Using Dynamic Clusters in the Oracle Fusion Middleware Administering Clusters for Oracle WebLogic Server* guide.

Patterns for Messaging

Messaging patterns describe the format of messages that flow between parts of a system built with a MOM. There are several types of messages as described below:

- A Command Message enables procedure call semantics to be executed in a messaging system.
- A Document Message enables a messaging system to transport information, such as the information that should be returned to a sender as a result of a command message.
- An Event Message uses messaging to perform event notification.
- A Reply Message handles the semantics of remote procedure call results, that require the ability to handle both successful and unsuccessful outcomes.
- A Reply Specifier enables a program making a request to identify the channel through which a reply should be sent.
- A Correlation Identifier enables a requesting program to associate a specific response with its request. When the data to be conveyed spans several messages, a Sequence Identifier enables the receiver to accurately reconstruct the original data.
- Message Expiration enables a sender to set a deadline by which the message should either be delivered or ignored.
- Message Throttle enables a receiver to control the rate at which it receives messages.

In the case of Service Bus, each reply message should contain a unique identifier called the correlation ID, which correlates the request message and its reply.

When the caller creates a request message, it assigns a unique identifier to the request that is different from those for all other currently outstanding requests (for example, requests that do not yet have replies). When the receiver processes the request, it saves the identifier and adds the request's identifier to the reply. When the caller processes the reply, it uses the request identifier to correlate the request with the reply. This is called a correlation identifier because of the way the caller uses the identifier to correlate each reply with the request.

A correlation ID is usually put in the header of a message. The ID is not a part of the command or data the caller is trying to communicate to the receiver. The receiver saves the ID from the request and adds it to the reply for the caller's benefit. Since the message body is the content being transmitted between the two systems and the ID is not a part of that, the ID is added to the header. In the request message, the ID can be stored as a correlation ID property or simply a message ID property. When used as a correlation ID, this can cause confusion about which message is the request and which is the reply. If a request has a message ID but no correlation ID, then a reply has a correlation ID that is the same as the request's message ID. The correlation ID format

used internally by Service Bus is compatible with WebSphere MQ and works with target services that are using MQ native interfaces.

The outbound transport handles asynchronous request-response messages. That is, the pipeline, except for the \$outbound transport specific data, does not distinguish between JMS request-response and HTTP request-response.

When you define a JMS request-response business or proxy service, you must first choose a design pattern. To do this, select the **Is Response Required** option for a JMS proxy service or a **Response Queues** option for a JMS business service, then select one of the following correlation patterns on the JMS Transport Configuration page:

- JMS Correlation ID (this is the default pattern)
- JMS Message ID

Note:

JMS request-response messaging does not have reliable responses because the mapping of the correlationID to the proxy service is stored in memory. If there is a failure or system restart between sending the request and receiving the response, the response is discarded.

To work around that situation, instead of using JMS request-response, use two one-way JMS proxy services: one for delivering the message and a second for delivering the response.

The following sections discuss these patterns. To compare the two patterns, see [Comparison of Message ID and Correlation ID Patterns](#).

JMS Message ID Pattern

When you create a business service using the JMS Message ID pattern, you can configure the responses to go to a single URI, or, for failover support, you can configure a response queue for each request URI on each Managed Server in the Service Bus cluster. These queues must be collocated with the request queues for the service. The proxy service uses this information to set the `JMSReplyTo` property when invoking the business service so the response is processed by the Managed Server that issued the request.

When you define a proxy service using the JMS Message ID pattern, you do not need to define the `Response URI` because the proxy service replies to the queue specified in the `JMSReplyTo` property. However, you can enter the JNDI name of the JMS connection factory for the response message. The `JMSReplyTo` property is accessible through the transport metadata, as described in [Access to the JMSReplyTo Property](#).

Note:

By default, the connection factory of the request message is used. This is useful when you use a non-XA connection factory for JMS responses, but have an XA connection factory for the request.

For the deployment descriptors to be set appropriately for XA-capable resources, you must set the XA attribute on the referenced connection factory before creating a proxy service.

The invoked service must copy the message ID from the request (the value of the JMS header field `JMSMessageID`) to the correlation ID of the response (setting the JMS header field `JMSCorrelationID`). In addition, the invoked service must reply to the queue specified in the `JMSReplyTo` header field. If you choose the JMS Message ID Pattern, the response arrives at the appropriate managed node.

Note:

The JMS Message ID correlation pattern is not supported when a proxy service invokes another proxy service.

Access to the `JMSReplyTo` Property

For JMS proxy services, the `JMSReplyTo` property in the incoming message is stored as a Java object in a JMS transport metadata element, also named `JMSReplyTo`. The value of the metadata element in the outbound request can be passed to the business service, supporting dynamic "reply to" destinations. In the pipeline, you can pass this value to a Java callout action for transformation. If a JMS proxy service invokes a JMS business service directly, with no pipeline, the JMS transport metadata is passed automatically to the business service. If the proxy service does invoke a pipeline, you need to configure the pipeline to copy the header from the inbound message and to set it in the outbound request header.

Note that in the first case, where the JMS business service is invoked directly with no pipeline, the consumer listening on this queue reads the `JMSReplyTo` header and also sends the message to the destination in the header. This means that two messages are written to the queue. To avoid this, use a pipeline between the services.

For request-response proxy services, the `JMSReplyTo` metadata element is only set when the correlation pattern is Message ID. When the pattern is Correlation ID, the proxy service determines the `JMSReplyTo` value from the transport configuration, so it is not set in the metadata element.

Note:

The `JMSReplyTo` metadata element does not appear in the Test Console because the element contains a `java-content` representation of a Java object.

JMS Message ID Pattern in a Cluster

A JMS proxy service using this pattern can be used in a cluster without further configuration. A JMS business service is available in a cluster. However, when a Managed Server is added to the cluster, all the business services become invalid. To correct this, ensure that the number of response queues equals the number of Managed Servers that specify the JMS Message ID correlation pattern in the Service Bus cluster.

JMS Correlation ID Pattern

When you design a business service in Java, make sure that you set the value of JMS Correlation ID on the response to the value of JMS Correlation ID on the request before sending the JMS response to a queue. You can obtain the JMS Correlation ID when you receive a message using the following method:

```
String getJMSCorrelationID()
```

The above method returns correlation ID values as Strings that provide specific message IDs or application-specific values.

To set the JMS Correlation ID when you send a message:

```
void setJMSCorrelationID(String correlationID)
```

Comparison of Message ID and Correlation ID Patterns

The JMS request-response patterns differ in the following ways:

- The method by which the response is correlated with the request
- The choice of the response queue

The differences between these two patterns are summarized in [Table 31-1](#).

Table 31-1 Differences Between Message ID and Correlation ID Patterns

JMS Pattern Name	Response Queue	CorrelationID
Correlation ID Pattern	All responses go to the same fixed queue(s).	The server copies the request Correlation ID to the response Correlation ID.
Message ID Pattern	The responses dynamically go to the queue indicated by the <code>JMSReplyTo</code> property.	The server copies the request Message ID to the response Correlation ID.

When the Correlation ID pattern is used, the service that is invoked replies to the queue that corresponds to the request URI. The response always arrives on the same queue and the client has no control over the queue to which the response arrives. For example, if 10 clients send a message to request URI "A", they all get the response to the queue that corresponds with request URI "A". Therefore, clients must filter the messages in the response queue to select the ones that pertain to them. Filtering criteria are configured in the request message Correlation ID property, and the server is configured to echo this to the response Correlation ID property.

In the case of Message ID pattern, the client's `JMSReplyTo` property tells the server where the response should be sent. This queue is specific to the client's server, so responses to different clients will go to different queues. The server sets the JMS Correlation ID of the response to the JMS Message ID of the request.

Correlation by MessageID is commonly used by many IBM MQ applications as well as JMS applications and is the standard method to correlate request and response.

If you have multiple Oracle WebLogic client domains invoking a target Oracle WebLogic domain using JMS request-response with the Message ID pattern, you can set up both the request and response queues as SAF queues. However, this is not possible with the Correlation ID pattern that uses a single queue for all the responses for a given request URI.

The Correlation ID pattern has two major advantages:

- The response queue configuration is simple and it need not change every time a new Managed Server is added to the cluster.
- Correlation ID can also be used in cases where a proxy service in the domain needs to invoke another proxy service in the same domain.

Interoperating with JAX-RPC Over JMS

The Service Bus development environment lets you create JAX-RPC web services that use the JMS transport, in addition to HTTP-HTTPS. These JMS transport JAX-RPC web services use a JMS queue as the mechanism for retrieving and returning values associated with operations. You can use the JMS Message ID pattern to invoke a JMS transport JAX-RPC web service.

You can also invoke a JMS request-response proxy service from a JAX-RPC static stub, which the Oracle WebLogic clientgen Ant task generates.

Invoking a JAX-RPC Web Service Using the JMS Message ID Pattern

To invoke a JMS transport JAX-RPC web service using the JMS Message ID pattern, complete the following steps:

1. Create a JMS Request-Response business service that uses the JMS Message ID pattern to invoke the JMS transport JAX-RPC web service. For more information, see the online help provided with Service Bus.

This business service uses JMS transport. The JMS queue JNDI name portion of the endpoint URI must be the same as the queue attribute specified in the `@WLJmsTransport` annotation of the JMS transport JAX-RPC web service. For example:

```
jms://localhost:7001/AJMSConnectionFactoryJNDIName/
JmsTransportServiceRequestQueue
```

The JNDI name of the JMS queue (or queues) assigned to the Destination field must be associated with a JMS server targeted at the WebLogic Server name that is displayed in the Target field.

Note:

While WebLogic Server allows forward slashes in JNDI names, such as "myqueues/myqueue", JNDI names with forward slashes interfere with the URI format required by Service Bus, and you cannot use those names. To work around this issue, define a JMS foreign server and reference that foreign server in the URI. For more information, see "Configure foreign servers" in the *Oracle WebLogic Server Administration Console Online Help*.

2. Create a proxy service that contains a Routing (or Service Callout) action to the JMS Request/Response business service that you created in step 1.

The Request Actions area of the Routing action must contain a `Set Transport Headers` action for the outbound request. When you configure the Transport Headers action, you must add two JMS headers for the outbound request action. For detailed instructions about how to configure a Transport Headers action, see [Adding Transport Header Actions in the Console](#).

In brief:

- a. Configure a Transport Headers Action by selecting **Other** in the **Add Header** field and entering a URI in the field provided.
- b. Select **Set Header to <Expression>** and create the expression by entering a concatenation of the values specified for the `contextPath` and `serviceUri`

attributes (in the `@WLJmsTransport` annotation of the JMS transport JAX-RPC web service), preceded by a forward-slash. For example, you have the following `@WLJmsTransport` annotation:

```
@WLJmsTransport(
  contextPath="transports",
  serviceUri="JmsTransportService",
  portName="JmsTransportPort",
  queue="JmsTransportServiceRequestQueue"
)
```

You would enter the following expression in the XQuery Text input area when you configure the Transport Headers:

```
/transports/JmsTransportService
```

- c. To specify the second JMS Header, select **Other** in the **Add Header** field again, and enter `_wls_mimehdrContent_Type` in the associated field.
- d. Select **Set Header to <Expression>** and enter `text/xml; charset=UTF-8` in the XQuery Text input area.

Invoking a JMS Request-Response Proxy Service from a JAX-RPC Client

For a scenario in which a JAX-RPC WebLogic Server client invokes a proxy service, set the `_wls_mimehdrContent_Type` JMS header for the proxy service's inbound response and specify the header when you issue the response to the incoming JMS Message ID Pattern request.

For example, for the scenario in which you have a JAX-RPC client calling a proxy service, which subsequently calls a WebLogic Server web service, the route node configuration is as follows:

For the Request Pipeline

1. Set the transport header for web service context URI (for example: `interop/AllocJmsDocLit`).
2. Set the transport header for `_wls_mimehdrContent_Type` with `text/xml; charset=UTF-8`.
3. Select **Outbound request** from the **Set Transport headers** menu items.
4. Enable `Pass all Headers` through pipeline.

For the Response Pipeline

1. Add an empty transport header and select **Inbound response** from the **Set Transport headers** menu.
2. Enable `Pass all Headers` through pipeline.

Note:

For instructions on configuring a Transport Headers action, see [Adding Transport Header Actions in the Console](#).

JMS Message ID Pattern Examples

The following examples describe the different methods by which the JMS Message ID pattern can be used.

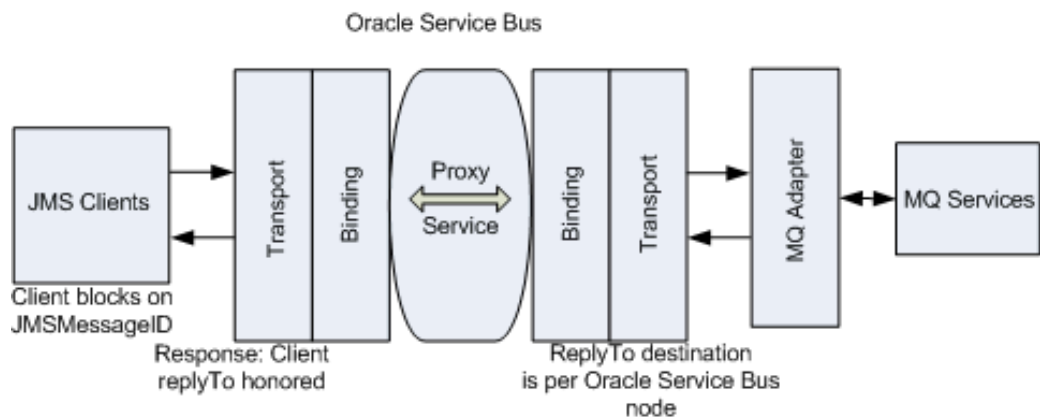
- [MQ Service Using a JMS Message ID to Correlate the Request-Response Message](#)
- [JAX-RPC Client with a Proxy Service](#)
- [Service Bus as a Client of a WebLogic Server JAX-RPC Service](#)

MQ Service Using a JMS Message ID to Correlate the Request-Response Message

In [Figure 31-1](#), the server that hosts the MQ service in the request-response communication echoes the request message ID to the response correlation ID, and sends the response to the `replyTo` queue. The response travels back and is correlated using the JMS MessageID. The Service Bus `replyTo` destination is set, one per Service Bus node in a cluster, when the business service is configured. A JMS or MQ native client can also invoke a JMS request-reply proxy service using the JMS Message ID pattern. The client needs to set the `replyTo` property to the queue where it expects the response.

The key to supporting this use case is that JMS Message ID is expected to correlate the request-response message. You also need to create as many MQ series outbound response queues as there are cluster servers.

Figure 31-1 MQ Service Using a JMS Message ID for Correlation



JAX-RPC Client with a Proxy Service

[Figure 31-2](#) represents a JAX-RPC client sending a message to a Service Bus proxy service, that is, the JAX-RPC inbound case. The JAX-RPC stack employs a temporary queue to receive the response. The JMS transport honors this temporary queue during runtime.

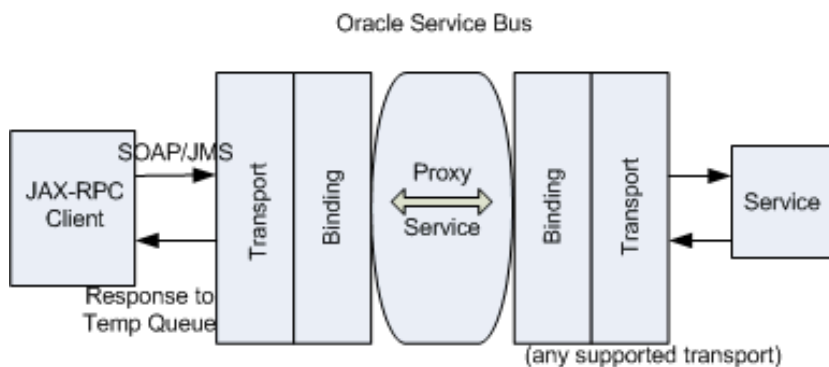
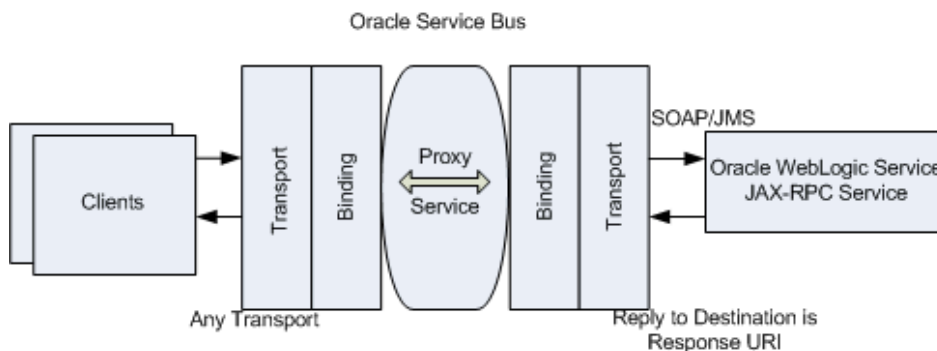
Figure 31-2 JAX-RPC Client with Service Bus Proxy Service**Service Bus as a Client of a WebLogic Server JAX-RPC Service**

Figure 31-3 represents the JAX-RPC outbound case or the interoperability of a WebLogic Server JAX-RPC request/response service with a Service Bus proxy service.

Figure 31-3 Service Bus as a Client of a WebLogic Server JAX-RPC Service**Note:**

When a proxy service in one WebLogic Server domain needs to send a message to a proxy service in a second domain, the message must first be routed to a pass-through business service in domain 1. JMS Store and Forward between domain 1 and domain 2 forwards the inbound request message to the proxy service in domain 2. When you use JMS request/response, you can choose to forward the inbound response message from domain 2 to domain 1 using JMS Store and Forward as well. In the latter case, exported inbound request and imported inbound response queues must be configured in domain 2 for the proxy service in domain 2. Pay close attention to the JMS Store and Forward configuration.

JMS Transport Configuration Reference

You can select JMS as the transport protocol for proxy and business services of any service type. This section describes the properties you can configure for the JMS transport for business and proxy services. For information about error handling for business services, see [JMS Transport Error Handling](#).

When you configure a proxy service, you can use a Transport Header action to set the header values in messages. For more information, see [JMS Transport Headers](#).

- [JMS Transport Endpoint URIs](#)
- [Configuring Proxy Services to Use the JMS Transport](#)
- [JMS Transport Headers](#)
- [Configuring Business Services to Use the JMS Transport](#)

JMS Transport Endpoint URIs

Enter the endpoint URI for the JMS transport in the following format:

```
jms://host:port[,host:port]*/connection_factory/jndi_destination
```

where

- `host` is the name of the system that hosts the service.
- `port` is the port number at which the connection is made.
- `[,host:port]*` indicates that you can configure multiple hosts with corresponding ports.
- `connection_factory` is the name of the JNDI Connection Factory. For more information on how to define a connection factory queue, see "Configure resources for JMS system modules" in the *Oracle WebLogic Server Administration Console Online Help*.
- `jndi_destination` is the name of the JNDI destination.

To target a JMS destination to multiple servers, use the following format for the URI:

```
jms://host1:port,host2:port/connection_factory/jndi_destination
```

where `connection_factory` is the name of the connection factory queue. For more information on how to define a connection factory queue, see "Configure resources for JMS system modules" in the *Oracle WebLogic Server Administration Console Online Help*.

Note:

While WebLogic Server allows forward slashes in JNDI names, such as "myqueues/myqueue", JNDI names with forward slashes interfere with the URI format required by Service Bus, and you cannot use those names. To work around this issue, define a JMS foreign server and reference that foreign server in the URI. For more information, see "Configure foreign servers" in the *Oracle WebLogic Server Administration Console Online Help*.

Configuring Proxy Services to Use the JMS Transport

The following table describes the properties you use to configure a JMS-based proxy service. For more information, see [Creating and Configuring Proxy Services](#).

Table 31-2 JMS Transport Properties for Proxy Services

Property	Description
Destination Type	Select one of the following: <ul style="list-style-type: none"> • Queue (for a point-to-point destination) • Topic (for a publish/subscribe destination)
Is Response Required	Select this option to specify that a response is expected after an outbound message is sent. This option is available only when the Destination Type is Queue .
Response Pattern	Select one of the following to specify the design pattern for the response: <ul style="list-style-type: none"> • JMSMessageID: For JAX-RPC services running on WebLogic Server. • JMSCorrelationID: For all other services. When you select this option, you must also enter a Response URI. This option is available only when the Is Response Required check box is selected.
Response Message Type	Select one of the following types for the response messages: <ul style="list-style-type: none"> • Bytes (for a stream of uninterpreted bytes) • Text (for text messages) This option is disabled if you select a Message Type of Java for the response. It is available only when the Is Response Required check box is selected.
Client Jar	Select the client JAR to be used for dequeuing messages that contain Java Objects. Selecting the client JAR ensures it is on the classpath. This option is available when the service is a Messaging Service with a request type of Java. For more information, see Sending and Receiving Java Objects in Messages .
Dispatch Policy	Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists. For information about Work Managers, see: <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>
Request Encoding	Enter the character set for encoding requests. The default is UTF-8.
Response Encoding	Enter the character set for encoding responses. The default is UTF-8. This option is available only when the Is Response Required check box is selected.
Client Response Timeout	Enter the number of seconds to wait for a server response before dropping the connection. This only applies if the client is another proxy service in the same domain. This option is available only when the Is Response Required check box is selected.

Table 31-2 (Cont.) JMS Transport Properties for Proxy Services

Property	Description
Response URI	<p>Enter a response URI in one of the formats described below. This option is available only when JMSCorrelationID is selected for the Response Pattern.</p> <pre>jms://host:port/connection_factory/jndi_destination</pre> <p>To target multiple servers, use the following format:</p> <pre>jms://host1:port,host2:port/connection_factory/jndi_destination</pre> <p>You can also omit the host and port in the response URI. For example:</p> <pre>jms:///connection_factory/jndi_destination</pre> <p>When you omit host and port, the connection factory/destination lookup occurs on the local server. This is useful, for example, if the request URI goes to a foreign connection factory/destination, but you want the response sent to the local server.</p> <p>Note: While WebLogic Server allows forward slashes in JNDI names, such as "myqueues/myqueue", JNDI names with forward slashes interfere with the URI format required by Service Bus, and you cannot use those names. To work around this issue, define a JMS foreign server and reference that foreign server in the URI. For more information, see "Configure Foreign Servers" in <i>Oracle WebLogic Server Administration Console Online Help</i>.</p>
Response Connection Factory	<p>Enter a response connection factory URI. This option is available only when JMSMessageID is selected for the Response Pattern. If a connection factory is not specified, the connection factory for the request is used for the response.</p>
JMS Service Account	<p>Select a service account to use for the JMS resource managed by the JMS server. A service account is an alias resource for a user ID and password, used for both the request and response. The same service account is used for both JMS and JNDI purposes. For more information, see Working with Service Accounts.</p>
Use SSL	<p>Select this check box only if the requests are made over a TLS/SSL connection.</p> <p>TLS/SSL (Secure Sockets Layer) provides secure connections by allowing two applications connecting over a network to authenticate the other's identity and by encrypting the data exchanged between the applications. Authentication allows a server, and optionally a client, to verify the identity of the application on the other end of a network connection. Additionally, if the administrator has restricted access to individual JMS destinations (queues or topics) by setting access control on the JNDI entry for the destination, the service must authenticate when looking up the entry in the JNDI tree.</p> <p>Note: The JMS transport does not support two-way SSL.</p>
Message Selector	<p>Enter a message selector expression. Only messages with properties matching the expression are processed.</p>

Table 31-2 (Cont.) JMS Transport Properties for Proxy Services

Property	Description
Client ID	<p>For topics only, enter the client ID to use for the subscriber. If no value is entered, a client ID is automatically generated. Assigning a custom ID helps you identify this component when monitoring the service. Note that this is only effective for durable subscribers.</p> <p>The JMS Client ID is one of several MDB descriptors that are used to generate subscriber names and the ClientID value for the topic's subscribers. Defining a JMS client ID makes it easier to identify and view subscribers for a specific topic by looking at the subscriber name or client ID.</p> <p>For more information, see JMS Client ID in Proxy Services.</p>
Durable Subscription	<p>Select this check box if the subscription is durable or leave it empty if the subscription is not durable. If a subscription is durable, it receives all messages published on a topic, even if the subscription was inactive at the time the messages were published.</p> <p>This option is available only if Topic is selected for the Destination Type.</p>
Retry Non XA Connection	<p>Select this check box to retry non-XA connections using the given retry count and interval specified below. If this check box is not selected, Service Bus only retries XA connections and the retry count and interval only apply only to XA connections.</p> <p>This check box only applies to JMS proxy services that have a non-XA connection factory in the service URL.</p>
Retry Count	<p>Enter the number of delivery retries a message can have before it is moved to the error destination. This field only applies to WebLogic Server JMS destinations.</p>
Retry Interval	<p>Enter the amount of time, in milliseconds, before rolled back or recovered messages are redelivered. This field only applies to WebLogic Server JMS destinations.</p>
Error Destination	<p>Enter the name of the target destination for messages that have reached their redelivery limit. This field only applies to WebLogic Server JMS destinations.</p>
Expiration Policy	<p>Select a policy that defines how to process an expired message encountered on a destination. You can specify to either discard or redirect the message. Redirecting a message moves it to the error destination specified above.</p> <p>This field only applies to WebLogic Server JMS destinations.</p>
Is XA Required	<p>Select this check box if your connection factory is XA.</p> <p>This option takes into account when the remote connection factory is unavailable. If your connection factory is available and this option is enabled, make sure that the connection factory is defined as transactional.</p>
Transaction Timeout	<p>Enter the amount of time in seconds for the JMS proxy service to wait for a transaction to be processed. This option only applies to services using an XA connection factory. The default is 600 seconds.</p>

Table 31-2 (Cont.) JMS Transport Properties for Proxy Services

Property	Description
No MDB on Response Queue	Select this option if you do not want to generate a default message-driven bean (MDB) on the inbound response queue. Use this option to improve performance. When this option is selected, a proxy to proxy routing format is not supported.
JNDI Timeout	The JNDI connection timeout (in seconds) used while looking up the destination or connection factory in the JNDI tree.
Topic Messages Distribution	<p>Select one of the following properties to determine how message-driven beans handle incoming JMS messages and high-availability and failover:</p> <ul style="list-style-type: none"> • One Copy Per Application: Select this option to provide high availability and scalability when the JMS proxy service is deployed to a cluster. This ensures that an inbound JMS message is processed on only one of the available servers in a cluster. This is the default value. • One Copy Per Server: Select this option if you want inbound JMS messages published to a topic to be received by the proxy service on every member of the cluster. • Compatibility: Select this option if you want inbound JMS messages to be processed on a specific Managed Server or all Managed Servers in a cluster. If you select this property, select a Target server. <p>This option is available when you select Topic for the Destination Type.</p> <p>Note: When using compatibility mode in a cluster, you might receive duplicate messages. To prevent this, use one of the other options.</p> <p>The "One Copy" options override the Subscription Sharing Policy and Client ID Policy configured on the JMS Connection Factory.</p>
Target	<p>Select the target server that will handle incoming JMS messages. If you select one of the "One Copy" options for Topic Messages Distribution, this field displays the name of the cluster. This option is available only in a Service Bus cluster when you select Compatibility for the Topic Messages Distribution option.</p> <p>If you do not set a target, the JMS proxy service instance reading messages off the Topic on each Managed Server in the cluster gets a copy of the message.</p>

JMS Transport Headers

The various headers related to the JMS transport are listed in [Table 31-3](#). All the headers except the Unit of Order header are common to both outbound requests and inbound response.

Table 31-3 JMS Transport Headers

Header	Description
JMSCorrelation ID	An identifier used to link one message with another. For example to link a request message with a response message.

Table 31-3 (Cont.) JMS Transport Headers

Header	Description
JMSDeliveryMode	The delivery mode specified when the message was sent.
JMSExpiration	The expiration time of the message, which is an absolute value that indicates the specific date and time the message should expire. You can use an XQuery expression to calculate the precise expiration date and time for each message.
JMSMessageID	A value that uniquely identifies the message sent by a provider.
JMSPriority	The processing priority of the message. When a message is sent, this field is ignored. After the send operation is complete, the field holds the value that is specified by the method sending the message.
JMSType	The message type identifier that is specified by a client when a message is sent.
JMSXAppID	This is one of the JMS defined properties that specifies the identity of the application that sends the message. This is set by the JMS provider
JMSXGroupID	This one of the properties defined by JMS that specifies the group id of the message group to which the message belongs. This is set by the client
JMSXGroupSeq	This one of the properties defined by JMS that specifies the sequence of the message inside the message group.
JMS_IBM_Format	The nature of the application data. For more information, refer to the IBM WebSphere documentation.
JMS_IBM_Report_Exception	Request exception reports. This also specifies how much of the application data from the message must be retained in the report message. For more information, refer to the IBM WebSphere documentation.
JMS_IBM_Report_Expiration	Request expiration reports. This also specifies how much of the application data from the message must be retained in the report message. For more information, refer to the IBM WebSphere documentation.
JMS_IBM_Report_COA	Request a confirm on arrival reports. This also specifies how much of the application data from the message must be retained in the report message. For more information, refer to the IBM WebSphere documentation.
JMS_IBM_Report_COD	Request a confirm on delivery reports. This also specifies how much of the application data from the message must be retained in the report message. For more information, refer to the IBM WebSphere documentation.
JMS_IBM_Report_PAN	Request a positive action notification reports. For more information, refer to the IBM WebSphere documentation.

Table 31-3 (Cont.) JMS Transport Headers

Header	Description
JMS_IBM_Report_NAN	Request a positive action notification reports. For more information, see For more information, refer to the IBM WebSphere documentation.
JMS_IBM_Report_Pass_Msg_ID	Request that the message identifier of any report or reply message is the same as that of the original message. For more information, refer to the IBM WebSphere documentation.
JMS_IBM_Report_Pass_Correl_ID	Request that the correlation identifier of any report or reply message is the same as that of the original message. For more information, refer to the IBM WebSphere documentation.
JMS_IBM_Report_Discard_Msg	Request that the message is discarded if it cannot be delivered to its intended destination. For more information, refer to the IBM WebSphere documentation.
JMS_IBM_MsgType	The type of a message. For more information, refer to the IBM WebSphere documentation.
JMS_IBM_Feedback	An indicator of the nature of a report message. For more information, refer to the IBM WebSphere documentation.
JMS_IBM_Last_Msg_In_Group	An indicator of whether the message is the last message in a message group. For more information, refer to the IBM WebSphere documentation.
JMS_BEA_UnitOfOrder	This header is valid for the request and response.

Configuring Transport Headers

You can configure the transport headers for both inbound and outbound requests in the pipeline. Use a Transport Header action to set the header values in messages. For information about adding transport header actions, see [Adding Transport Header Actions in the Console](#). For information about the transport headers related to the JMS transport, see [JMS Transport Headers](#).

Note:

For information about the limitations for JMS transport headers, see [How the Runtime Uses the Transport Settings in the Test Console](#). Also see [Table 12-8](#).

Configuring Business Services to Use the JMS Transport

When you register a JMS business service, you must manually edit the URI from the WSDL file when adding it to the service definition. The URI format is as follows:

```
jms://host:port/connection_factory/jndi_destination
```

Note:

To configure a JMS request-response application with response correlation by JMS ID for a business service, you must:

- Create uniform distributed queues and local queues targeted to one chosen JMS server.
- Disable the default targeting for the UDQ that deploys the UDQ on the cluster and creates the member queues on all JMS.

The following table describes the properties you use to configure a JMS-based business service. For more information, see [Creating and Configuring Business Services](#)

Table 31-4 JMS Transport Properties for Business Services

Property	Description
Destination Type	Select one of the following JMS destination types: <ul style="list-style-type: none"> • Queue (for a point-to-point destination) • Topic (for a publish/subscribe destination)
Message Type	Select one of the following message types: <ul style="list-style-type: none"> • Bytes (for a stream of uninterpreted bytes) • Text (for text messages) <p>This option is disabled if you select a message type of Java for the response.</p>
Response Queues	Select one of the following options to specify how to handle responses: <ul style="list-style-type: none"> • None: No response is expected. Select this option for one-way operations. • One for all Request URIs: Lets you enter a single URI to handle the response, as well as set other response configuration details such as encoding and timeout, and optionally select a JMS Service Account for passing JMS/JNDI credentials. • One per Request URI: This option provides response failover, letting you enter a response URI or destination for each request URI. You can optionally select a service account for JMS/JNDI credentials on each request/response pairing. <p>This option is available only when the Destination Type is Queue.</p>
Response Pattern	Select one of the following to specify the design pattern for the response: <ul style="list-style-type: none"> • Select JMSCorrelationID for all services other than JAX-RPC services running on WebLogic Server. • Select JMSMessageID for JAX-RPC services running on WebLogic Server. <p>This option is available only when you select a response option in the Response Queue field. For more information, see Message ID and Correlation ID Patterns for JMS Request/Response.</p>

Table 31-4 (Cont.) JMS Transport Properties for Business Services

Property	Description
Dispatch Policy	<p>Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists.</p> <p>For example, if the business service has a JMS transport protocol, the business service endpoint is an MDB (message-driven bean) JAR file that you can associate with the specific dispatch policy.</p> <p>For information about Work Managers, see:</p> <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>
Request Encoding	Enter the character set for encoding requests. The default is UTF-8.
Response Encoding	<p>Enter the character set for encoding responses. The default is UTF-8.</p> <p>This option is available only when one of the response options is selected in the Response Queues field.</p>
Response Timeout	<p>Enter the amount of time, in seconds, to wait for the response before dropping the connection. The default, zero (0), means the response never times out.</p> <p>This option is available only when one of the response options is selected in the Response Queues field.</p>
Client Jar	<p>Select the client JAR to be used for dequeuing messages that contain Java Objects. Selecting the client JAR ensures it is on the classpath. This option is available when the service is a Messaging Service with a response type of Java.</p> <p>For more information, see Sending and Receiving Java Objects in Messages.</p>

Table 31-4 (Cont.) JMS Transport Properties for Business Services

Property	Description
Response URI	<p>Enter a response URI in one of the formats described below. This option is available when you select the One for all Request URIs response option and the JMSCorrelationID response pattern.</p> <pre data-bbox="662 432 1273 459">jms://host:port/connection_factory/jndi_destination</pre> <p>To target multiple servers, use the following format:</p> <pre data-bbox="662 537 1224 590">jms://host1:port,host2:port/connection_factory/jndi_destination</pre> <p>You can also omit the host and port in the response URI. For example:</p> <pre data-bbox="662 699 1166 726">jms:///connection_factory/jndi_destination</pre> <p>When you omit host and port, the connection factory/destination lookup occurs on the local server. This is useful, for example, if the request URI goes to a foreign connection factory/destination, but you want the response sent to the local server.</p> <p>Note: While WebLogic Server allows forward slashes in JNDI names, such as "myqueues/myqueue", JNDI names with forward slashes interfere with the URI format required by Service Bus, and you cannot use those names. To work around this issue, define a JMS foreign server and reference that foreign server in the URI. For more information, see "Configure Foreign Servers" in <i>Oracle WebLogic Server Administration Console Online Help</i>.</p>
Request Responses	<p>Enter a Response URI for each request URI entered on the generic Transport page. Use the formatting and guidelines described for the Response URI field, above.</p> <p>For each URI, you can select an optional service account for JMS/JNDI credentials that the service uses for both the request and response queues.</p> <p>This option is available when you select the One per Request URI response option for the JMSCorrelationID pattern to provide response failover.</p>
Target - Responses	<p>Enter the name of the Target server that is to receive responses, and enter a Response URI, as described in the Response URI field.</p> <p>This option is available when you select the One for all Request URIs response option for the JMSMessageID pattern.</p>
Request Connections	<p>Enter a JMS Connection Factory name for each request URI, identified sequentially by number in the Seq. No field. If you do not enter a name, the JMS transport uses the connection factory from the request URI.</p> <p>You can select an optional service account for JMS/JNDI credentials that the service uses for both the request and response queues.</p> <p>This option is available when you select the One per Request URI response option for the JMSMessageID pattern to provide response failover.</p>

Table 31-4 (Cont.) JMS Transport Properties for Business Services

Property	Description
Target - Destinations	<p>Enter the destination queue on each target server that receives responses for each request URI on each target. Each Target server in the list (determined by the servers in the current domain, such as Managed Servers in a cluster) lists the request URIs by Seq.No, which correspond to those in the Request Connections field.</p> <p>This option is available when you select the One per Request URI response option for the JMSMessageID pattern to provide response failover. Use this field in conjunction with Request Connections.</p> <p>Note: Because the Service Bus development environment supports only a one-server environment, only one Target is listed. To configure this field in a multi-server environment, deploy this service to the runtime environment and complete the service configuration in the Oracle Service Bus Console.</p>
JMS Service Account	<p>Enter a service account to use for the JMS resource managed by the JMS server. A service account is an alias resource for a user ID and password, used for both the request and response. The same service account is used for both JMS and JNDI purposes.</p> <p>This option is available when you use the None or One for all Request URIs option in the Response Queues field. This field is mutually exclusive with the Pass Caller's Subject option. Use one or the other for JMS/JNDI authentication.</p> <p>For more information, see Working with Service Accounts.</p>
Use SSL	<p>Select this check box only if the requests are made over a TLS/SSL connection.</p> <p>TLS/SSL (Secure Sockets Layer) provides secure connections by allowing two applications connecting over a network to authenticate the other's identity and by encrypting the data exchanged between the applications. Authentication allows a server, and optionally a client, to verify the identity of the application on the other end of a network connection.</p> <p>Additionally, if the administrator has restricted access to individual JMS destinations (queues or topics) by setting access control on the JNDI entry for the destination, the service must authenticate when looking up the entry in the JNDI tree. Authenticate using a Service Account or the Pass Caller's Subject option.</p> <p>Note: The JMS transport does not support two-way SSL</p>
Expiration	<p>The time interval in milliseconds after which the message expires. Default value is 0, which means that the message never expires.</p>
Enable Message Persistence	<p>Select this check box for guaranteed message delivery. Clear this check box to improve throughput if an occasional lost message is tolerable. The JMS message delivery mode lets you balance reliability with throughput.</p>

Table 31-4 (Cont.) JMS Transport Properties for Business Services

Property	Description
Unit of Order	<p>Enter a message unit-of-order, which enables message producers to group messages into a single unit with respect to the processing order. This single unit-of-order requires that all messages in that unit be processed sequentially in the order they were created.</p> <p>For more information about using unit-of-order, see "Using Message Unit-of-Order" in <i>Developing JMS Applications for Oracle WebLogic Server</i>.</p>
Pass Caller's Subject	<p>Select this check box to have Service Bus pass the authenticated subject when sending a message.</p> <p>When you enable this field and the business service targets JMS resources in a different domain, enable global trust on both domains. See "Configuring Security for a WebLogic Domain" in <i>Administering Security for Oracle WebLogic Server</i>.</p> <p>This field is mutually exclusive with the Service Account option. Use one or the other for JMS/JNDI authentication.</p>
JNDI Timeout	<p>Enter the period (in seconds) to use when looking up the destination or connection factory in the JNDI tree before the JNDI connection times out.</p> <p>The default, zero (0), means the connection never times out.</p>

Using the Local Transport

This chapter provides an overview of the Local transport and describes how to use and configure it in your proxy services.

This chapter includes the following sections:

- [Introduction to the Local Transport](#)
- [Using Local Transport Proxy Services](#)
- [Propagating SOAP Faults Between Proxy Services](#)
- [Using OWSM Security with Local Proxy Services](#)

Introduction to the Local Transport

In a Service Bus project, proxy service logic is exposed to the client, but there may be cases where you do not want that logic exposed. In this case, you can design the logic behind a local transport proxy service, and then invoke that proxy service from other Service Bus projects. For example, if you have several services that invoke a back-end service, you can create a separate project with a local proxy service to define all the back-end (unexposed) logic. You can then invoke that local proxy service from other Service Bus projects, keeping certain processing logic private from all clients.

Local proxy services are also the only way to call pipelines contained in other projects. When the pipelines are all in the same Service Bus project, one pipeline can call another pipeline directly. However, a pipeline cannot call a pipeline in a different project directly. To achieve this, you need to use a local proxy service between the pipelines. The local transport provides the following capabilities:

- Efficient and secure communication.
- Propagation of transactions and transactional behavior.
- Propagation of security context so the identity can be propagated end-to-end. The security context propagation also allows the client of the first proxy service in a chain of services to be authorized by the proxy services that are subsequently invoked in the chain, supporting fine-grained access control.

Features and Characteristics of Local Transport Proxy Services

Local transport-based proxy services can only be invoked by other proxy services or pipelines, and not by other clients. The invocation is optimized by Service Bus. Local proxy services do not have a URI; however, there are no constraints on the service and interface types supported by local transport proxy services. The one exception is that SAML is only supported in a pass through scenario.

If the quality of service (QoS) for the invoking service is defined as Exactly Once, the transaction of that service is propagated to the local transport proxy service. In other

words, the invoked local transport proxy service inherits the transactional behavior of the invoking service.

A proxy service can authenticate at the transport level or the message level. If it is enabled, the effective client is the message-level authenticated client. If the message-level authenticated client is not enabled, then the transport-level authenticated client is the effective client (if that is enabled). If neither the message-level nor the transport-level authenticated client is enabled, the anonymous client becomes the effective client.

When a proxy service invokes a local transport proxy service, the effective client of the invoking service becomes the transport-level client of the invoked local proxy service. A local transport proxy service can authorize this client for access with an access control policy. In this way, it is possible to propagate the client of the first service to the subsequent proxy services in the overall end-to-end message flow.

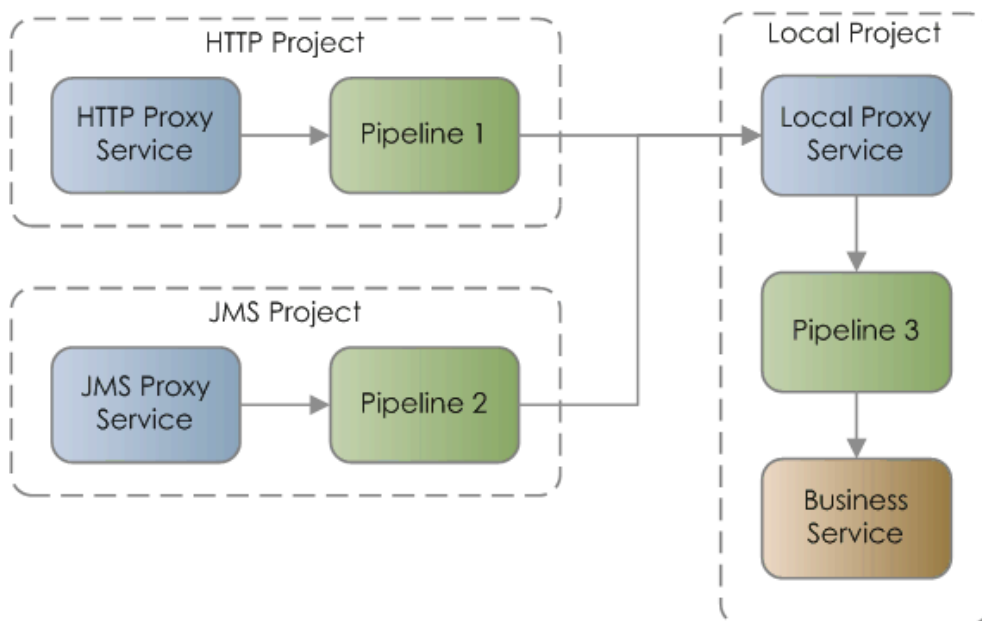
Local transport proxy services support user-defined transport headers. Consider a scenario in which a proxy service uses the HTTP transport. It routes through a pipeline to a local proxy service, and the pipeline passes headers to the local proxy service using a transport header action. In this scenario, if the HTTP proxy service received the Content-Type header, that header is available as a user header in the local transport and is therefore accessible through the standard user header, instead of as a typed transport header.

Using Local Transport Proxy Services

Local transport proxy services are useful when you have portions of the message flow you want to keep private and other portions you want to expose. For example, you could have projects that define back-end processing and call those projects from projects that define front-end logic such as alerting and fault handling. Using local proxy services for the back-end projects keeps them private.

The pipelines in the front-end projects route messages to the local transport proxy service. You can call the same local proxy service from multiple pipelines and from multiple projects. The following figure illustrates a local proxy service invoked by a project with an HTTP proxy service and by a project with a JMS proxy service.

Figure 32-1 Using a Local Transport Proxy Service



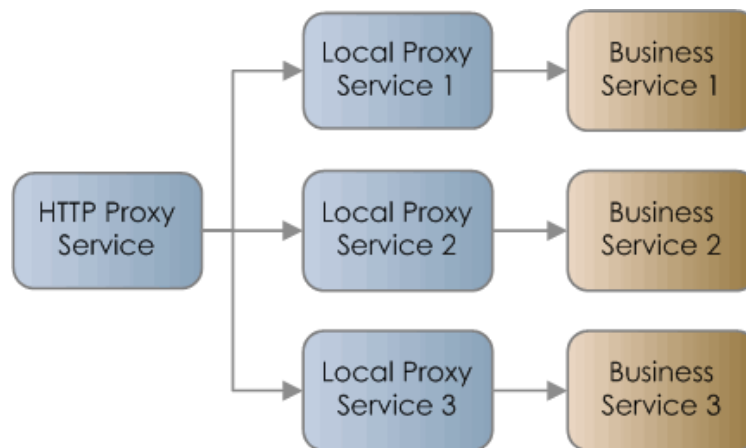
Changes from Previous Usage

In previous versions of Service Bus, the local transport could be used in cases where an Any SOAP or XML type proxy service acted as a front-end to different enterprise systems. This front-end proxy service was a generic router to the appropriate local transport. In the current Service Bus version, with the ability to create pipelines separately from the proxy service, this usage of the local transport is obsolete. You can replace the local proxy services with pipelines, as illustrated in [Figure 32-2](#) and [Figure 32-3](#).

As with the previous version, you can use dynamic routing to abstract the routing rules at runtime and route messages to the local transport proxy services. For an example of how dynamic routing is used, see [Using Dynamic Routing](#).

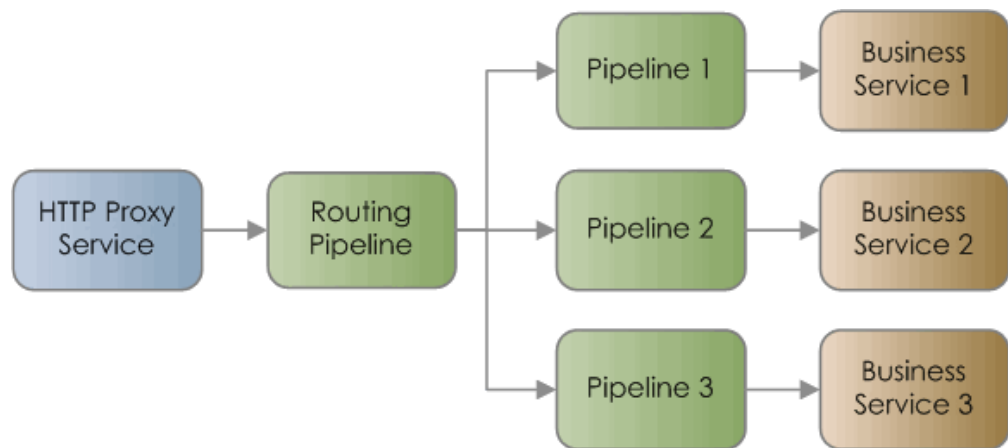
The following figure illustrates how the local transport would be used in this scenario in release 11g.

Figure 32-2 Accessing Multiple Business Services in Previous Versions



In the current release, the above 11g scenario would be updated to include pipelines instead of local proxy services, as shown below.

Figure 32-3 Updated Method to Access Multiple Business Services



Propagating SOAP Faults Between Proxy Services

When chaining local proxy services, SOAP faults in the `$fault` variable are not automatically propagated from one proxy service through the pipeline to another proxy service. Consider the following example:

```
Client > Proxy1 > Pipeline > Proxy2 > Business Service > Back-end Service
```

If a SOAP fault occurs in the back-end service, it is propagated to the `$fault` variable in Proxy2. However, the SOAP fault value in Proxy2 is not automatically propagated to the `$fault` variable in Proxy1 and is therefore not returned to the client.

To propagate the SOAP fault from one proxy to another:

1. In the pipeline, add an error handler that contains a **Reply with Failure** action. This returns the SOAP message with fault information contained in the `$body` variable. For more information, see [Adding Reply Actions in the Console](#).
2. In the pipeline, transform the `$body` variable as necessary to return the desired SOAP error details to the client.

For more information on how Service Bus handles SOAP faults, see [Generating the Error Message, Reporting, and Replying](#).

Using OWSM Security with Local Proxy Services

You can attach Oracle Web Services Manager (OWSM) service policies to local proxy services with a WSDL service type, which lets you apply specific security controls to messages arriving at each local proxy. This section describes how Service Bus processes policies at runtime when a proxy service forwards messages with security headers through a pipeline to local proxy services. Message forwarding occurs through actions such as route, service callout, and publish.

Proxy services do not perform outbound WS-Security processing when forwarding messages to other proxy services. The diagrams in this section illustrate this behavior, showing WS-Security configurations in different proxy-to-proxy scenarios. Use these scenarios to understand the behavior so that you can successfully use OWSM service policies on local proxy services that receive messages from other proxy services.

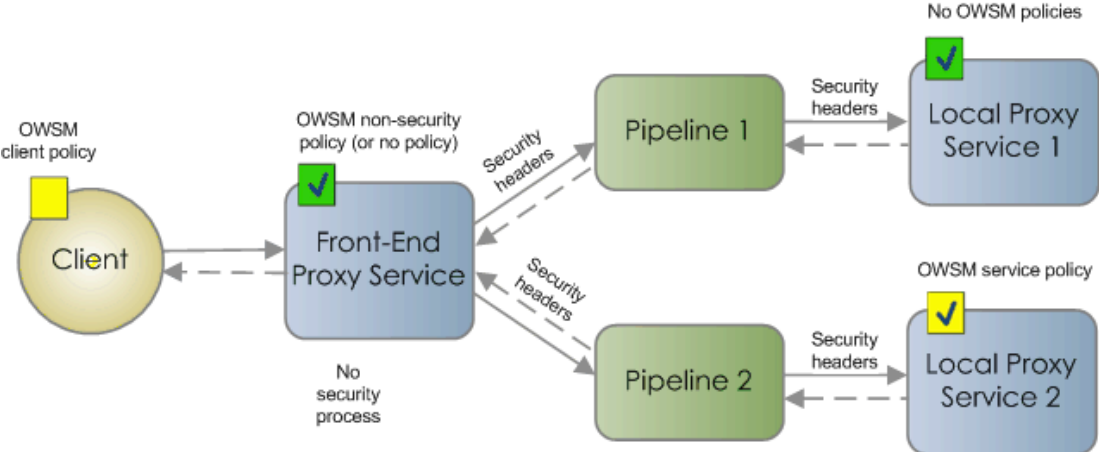
[Figure 32-4](#) shows a client with a client policy sending a message to a front-end proxy that could have any of the following characteristics:

- The front-end proxy could be active and contain an OWSM policy that performs inbound processing on all WS-Security headers in a request or only a subset of those headers. For example, it might process the authentication policy but not the message protection policy when the request contains both authentication and message protection headers. The proxy could also contain a non-security policy such as an OWSM log policy.
- The front-end proxy could be passive and contain an OWSM policy.
- The front-end proxy could contain no OWSM policy.

In each of these cases, the front-end proxy service encounters at least one security header in the message. The proxy service passes this message without performing outbound WS-Security processing to the pipelines, which in turn pass the message to the local proxy services. The local proxy services may or may not contain OWSM policies.

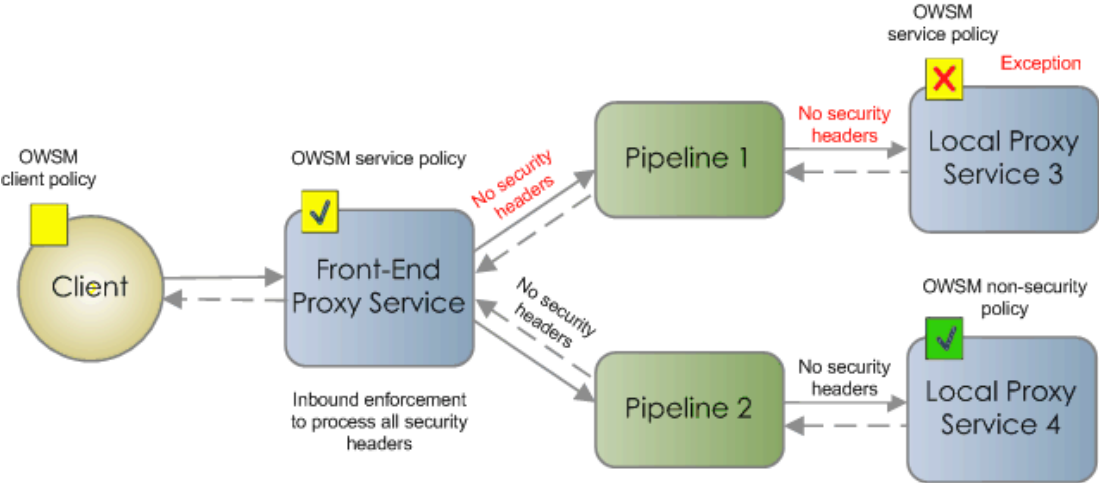
In [Figure 32-4](#), local proxy service 2 receives the message without throwing an exception, because the message contains the expected security headers. Even if the front-end proxy service contains a policy that performs partial security processing (for example, authentication processing but no message protection processing), the forwarded message would still contain security headers.

Figure 32-4 Front-end Proxy as Security Pass-through to Local Proxies



[Figure 32-5](#) shows a client with a client policy sending a message to a front-end proxy service. The front-end proxy service is active and contains an OWSM service policy that processes all WS-Security headers in the message. The inbound service policy is processed, which strips the message of its security headers. Because the front-end proxy service forwards the message to other proxies, no outbound WS-Security processing is performed, and the message without security headers is forwarded to the local proxy services. One local proxy service contains an OWSM service policy that expects security headers, and an exception is thrown when the message arrives without those headers. The other local proxy contains an OWSM non-security policy where no enforcement occurs, so the message without security headers passes through successfully.

Figure 32-5 Front-end Proxy Processes All Security Headers Before Forwarding to Local Proxies



Using the MQ Transport

This chapter provides an overview of the MQ transport and describes how to use and configure it in your services. The MQ transport provides access to IBM WebSphere MQ and supports both inbound and outbound connectivity.

This chapter includes the following sections:

- [Introduction to the MQ Transport](#)
- [Setting Up the Environment for the MQ Transport](#)
- [Working with MQ Connections](#)
- [MQ Transport Error Handling](#)
- [Using the WebSphere JMS MQ Interface](#)
- [MQ Transport Configuration Reference](#)
- [MQ Transport Headers](#)

Introduction to the MQ Transport

The MQ transport provides native connectivity between Service Bus components and IBM WebSphere MQ. MQ proxy services can receive messages from WebSphere MQ and MQ business services can route messages to WebSphere MQ.

To learn more about WebSphere MQ fundamentals, Refer to the IBM WebSphere MQ documentation.

MQ Transport Features

The MQ transport provides Service Bus with the following features:

- Inbound and outbound connectivity. MQ proxy services can receive messages from WebSphere MQ and MQ business services can route messages to WebSphere MQ.
- Access to WebSphere MQ. For more information, see [How to Add MQ Client Libraries to Your Environment](#).
- Sending and receiving messages of Any XML, Binary, XML, Text and MFL types.
- Processing of all the MQ message descriptor (MQMD) headers. A message descriptor is an attribute representing a property of the message that is either being sent or received. For a list of MQ headers that can be configured, see [MQ Transport Headers](#).

- TCP/IP and bindings mode for connecting to queue managers. TCP/IP mode connects to remote WebSphere MQ servers; bindings mode connects to a local WebSphere MQ.
- One-way and two-way SSL on inbound and outbound transport (only when a TCP/IP connection is used).

MQ Transport Advantages

Using the MQ transport has the following advantages over using the WebSphere MQ JMS interface:

- Faster connection to WebSphere MQ through the MQ transport than through the WebSphere MQ JMS interface.
- Ability to read and generate MQ messages. Using the JMS interface, it is not possible to set certain headers.
- Support for sending and receiving MQ receipt messages.
- Explicit binding of MQ Connection Factory and MQ Queue to WebLogic's JNDI is not required.
- Configuration of resources, like a JMS provider, outside of Service Bus is not required.
- Performance improvement because messages are sent directly using the transport instead of channeling them through the JMS transport.

Messaging Patterns

The MQ transport supports one-way and request-response messaging patterns for both inbound and outbound connectivity. The default pattern is one-way messaging. A proxy or business service supports request-response messaging when you set the `IsResponseRequired` option while configuring the service.

The inbound and outbound transports support the asynchronous request-response pattern using `messageID` or `correlationID` for correlation between the request and the response. You can set the response correlation pattern during service configuration. For more information, see "CorrelationID" and "MessageID" in [MQ Transport Headers](#).

The outbound transport provides an option to auto-generate the correlation ID and messageID or to use the one specified by you in the pipeline. Select the `Auto-generate Correlation Value` option to indicate that the correlation ID and message ID should be auto-generated by the transport. When this option is not selected, Service Bus uses the value specified by you in the pipeline. If you use dynamic queues in your MQ implementation, you can use dynamic queues for response correlation in the outbound transport.

If the correlation value (`messageID` / `correlationID`) is not auto-generated and if the Managed Server goes down, the remaining response messages may never get removed when the server is restarted. Oracle recommends that the `Expiry` header on the request is configured to a finite value and that the `Report` header contains the `MQC.MQRO_PASS_DISCARD_AND_EXPIRY` option. The `MQC.MQRO_PASS_DISCARD_AND_EXPIRY` option serves as a directive to the receiving client that the message descriptor of the reply should inherit the `Expiry` header value of the request message. This ensures that the response messages are

removed by the MQ server after the configured expiry period has elapsed. When the correlation value is automatically generated, the Service Bus server is responsible for cleaning up any remaining response messages.

The MQ transport supports local transactions but not remote transactions.

For more information about configuring `Is Response Required`, `Response Correlation Pattern`, `Auto Generate Correlation Value` for a service, see [Configuring Proxy Services to Use the MQ Transport](#) and [Configuring Business Services to Use the MQ Transport](#).

MQ Connection Resources

MQ connections are sharable resources that can be reused across multiple MQ proxy and business services. MQ connection resources provide the connection parameters required for connecting to an MQ queue manager. You must create and configure an MQ connection resource in order to create an MQ-based proxy or business service. For more information, see [Working with MQ Connections](#).

Quality of Service

When the inbound transport is MQ and the Quality of Service (QoS) on the outbound transport is *exactly-once*, the resulting QoS is *at-least-once*. By default, the QoS on the outbound transport is *exactly-once*.

Note:

You must create error handling logic (including any retry logic) in the pipeline error handler. For information about configuring error handling, see [Adding Error Handlers in JDeveloper](#) and [Adding Error Handlers in the Console](#).

When the outbound is request-response, the QoS is *at-least-once* only if the outbound transport is configured to support *exactly-once* QoS. For more information about QoS in Service Bus messaging, see [Quality of Service](#).

MQ Clusters and the MQ Transport

Cluster support in WebSphere MQ is store-and-forward messaging and not load-balancing and fail over. The cluster queues in WebSphere MQ are created locally on one of the queue managers and shared with other cluster members that act as remote forwarders to the shared queue. Requests from the MQ transport are load balanced by sending messages using the load balancing algorithm to the members of the cluster. However, the transport receives messages by accessing only the MQ server node that holds the reference to the local queue.

Limitations of the MQ Transport

The following are the limitations of the MQ transport:

- You cannot call a request-response proxy service based on MQ proxy service:
 - From a proxy service that has been configured with a route action or dynamic routing and routing table actions).
Using the service callout action.

- You cannot call a proxy service with a service callout where the target is a request-response proxy service based on MQ transport.
- You cannot use an indirect call to a request-response MQ proxy service in the Service Bus Test Console.

Setting Up the Environment for the MQ Transport

Service Bus is a client for IBM WebSphere MQ, and although Service Bus supports runtime server compatibility for supported versions of WebSphere MQ, these MQ libraries are not installed with Service Bus. You need to make a supported version of the MQ client library, `com.ibm.mq.jar`, available in your domain.

For WebSphere MQ version support with Service Bus, see "Interoperability Scenarios and Considerations" in *Administering Oracle Service Bus*. For information about the system requirements for WebSphere MQ, see <http://www-306.ibm.com/software/integration/wmq/requirements/index.html>.

How to Add MQ Client Libraries to Your Environment

1. Stop the domain server.
2. From your WebSphere MQ installation, copy the `com.ibm.mq.jar` file to the Service Bus domain at `DOMAIN_HOME/lib` directory.
3. Restart the domain server.

How to Configure Environment Variables

If you use the bindings mode to connect to an MQ Queue Manager located on the same machine as Service Bus, add `<MQ_install_directory>/bin` and `<MQ_install_directory>/java/lib` to the PATH environment variable.

Working with MQ Connections

Before you can configure a proxy or business service to use the MQ transport, you need to create an MQ connection resource that defines the connection to the MQ server. MQ connections are sharable resources that can be reused across multiple MQ proxy and business services. MQ proxy and business services must connect to an MQ queue manager before accessing an MQ queue. MQ connection resources provide the connection parameters required for connecting to an MQ queue manager.

Each MQ Connection resource has a connection pool. Every business or proxy service that uses the same MQ connection resource to get a connection to a queue manager uses the same connection pool that was created for that resource. Thus, multiple business services and proxy services can use the same queue manager and share a connection pool.

How to Create MQ Connections

The option to create an MQ connection in Oracle Service Bus Console is only available after the MQ client library is added to the domain. In JDeveloper, you can create the connection without the libraries, but you need to libraries for the runtime.

Before you begin:

Follow the instructions under [“Setting Up the Environment for the MQ Transport”](#) to make the MQ client library available.

Make sure you have created any of the following resources you will need to create the MQ connection:

- Service account
- Service key provider

To create an MQ connection:

1. Do one of the following:

- For JDeveloper: In the Application Navigator, right-click the project or folder to contain the new MQ connection, point to **New**, and select **MQ Connection**.
- For Oracle Service Bus Console: In the Project Navigator, right-click the project or folder to contain the new MQ connection, point to **Create**, and select **MQ Connection**.

2. Enter a unique name for this MQ connection, and an optional description.

The endpoint URI cannot contain spaces, so do not create MQ Connection resources, projects, or folders with spaces in the names.

3. Click **Create or **Finish**.**

The MQ Connection Definition Editor appears.

4. In the **Connection Type field, select one of the following modes for connecting to the MQ queue manager:**

- **mqTcpModeType**: Use TCP/IP to connect to a queue manager that does not reside on the same machine as Service Bus.
- **mqBindingModeType**: Use the bindings mode to connect to a queue manager that is located on the same machine as Service Bus.

The fields on the editor change based on your selection.

5. Configure the following information about the MQ server:

- **MQ Host Name**: The host name of the MQ queue manager. This is used for TCP mode connections only.
- **MQ Port Number**: The port number of the MQ queue manager listener. This is used for TCP mode connections only.
- **MQ Queue Manager Name**: Enter the name of the MQ queue manager to which to connect.
- **MQ Queue Manager Channel Name**: Enter the name of the queue manager server connection channel. This is used for TCP mode connections only.
- **Queue Manager CCSID**: The coded character set identifier (CCSID) to be used when establishing a connection. This is used for TCP mode connections only, and is mainly for internationalization support.

6. To configure SSL support for TCP mode connections, do the following:

- a. Select the **SSL Required** check box to use SSL for sending messages. This enables service-side SSL authentication.
 - b. In the **Cipher Suite** field, select the cipher suite algorithm to be used by SSL. The Cipher Suite algorithm is used to encrypt and decrypt message communications between the server and client.
 - c. To enable both client-side and server-side SSL authentication, select the **2-way SSL Required** check box.
 - d. If you selected 2-way SSL, for the **Reference to the Service Key Provider** field, click the **Browse** icon to locate and select the service key provider to use.
7. For the **Reference to the Static Service Account** field, click the Browse icon to locate and select the service account to use.
 8. In the **MQ Version** field, select the version of MQ server you are using.
 9. To configure the MQ connection, do the following:
 - a. In the **MQ Connection Pool Size** field, enter the number of connections maintained by the connection pool.
 - b. In the **MQ Connection Timeout** field, enter the time interval in seconds after which unused connections are destroyed. The default is 1800 seconds.
 - c. In the **MQ Connection Max Wait** field, enter the maximum amount of time (in milliseconds) to wait for a connection to become available.

If a connection is not made within that time interval, Service Bus throws an exception. The default is 3000 milliseconds
 - d. If the transactions handled by this connection are distributed (XA) transactions, select **XA Enabled**.

Note that the queue must be configured for persistence. If you are using version 5.3 or 6.0 (both deprecated), add the `com.ibm.mqetclient.jar` file to your classpath.
 10. In the toolbar, click **Save**.
 11. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Edit MQ Connections

Once you create an MQ connection, you can reconfigure its information.

To edit an MQ connection:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the MQ connection to edit.
2. Right-click the MQ connection name, and select **Open**.
3. Modify any of the fields described in [How to Create MQ Connections](#). The online help provides more detailed descriptions.
4. When you are done making changes, click **Save** in the toolbar.

5. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Delete MQ Connections

Before you delete an MQ connection, make sure it is not currently being used by an MQ transport. If it is being used, remove it from the proxy or business service configuration before deleting it. You can delete the MQ connection even if it is referenced by other resources, though this might result in conflicts due to unresolved references to the deleted resource.

You can check for references and dependencies in both the console and JDeveloper. In the Oracle Service Bus Console, open the MQ connection in the MQ Connection Definition Editor and click the **References** icon in the upper right to find out whether any services are using it. In JDeveloper, right-click the MQ connection and select **Explore Dependencies**.

To delete an MQ connection:

1. In the Application Navigator or Project Navigator, expand the project and folders containing the MQ connection to delete.
2. Right-click the name of the MQ connection, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the MQ connection. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

MQ Transport Error Handling

You can configure MQ-transport business services to handle application and communications errors as follows:

- **Application errors:** You can specify whether or not to retry business service endpoint URIs when application errors occur. For more information, see the online help provided for the Business Service Definition Editor for the **Retry Application Errors** field.
- **Communication errors:** You can configure business service URIs to be taken offline when communication errors occur. For more information, see "Managing and Monitoring Endpoint URIs for Business Services" in *Administering Oracle Service Bus*.

Using the WebSphere JMS MQ Interface

The following sections outline how Service Bus connects to WebSphere MQ and presents an overview of some message types used in communication between WebSphere MQ and Service Bus.

- [Using the WebSphere MQ JMS Interface](#)
- [MQ Messaging Types](#)
- [Tuning WebSphere MQ](#)

Using the WebSphere MQ JMS Interface

Service Bus connects to WebSphere MQ through the WebSphere MQ JMS interface. That is, Service Bus is a WebSphere MQ JMS client. The foreign JMS server in Oracle WebLogic Server specifies the initial context factory, connection factory, and queue to the WebSphere MQ server. For more information, see "Configuring Foreign Server Resources to Access Third-Party JMS Providers" in *Administering JMS Resources for Oracle WebLogic Server*.

WebSphere MQ JMS supports two transport types:

- BINDINGS
- CLIENT

If the WebSphere MQ JMS client is running on the same physical machine as the queue manager, set the transport type to BINDINGS. Otherwise, you can use only the CLIENT type.

WebSphere MQ can interface with Service Bus in two ways:

- Service Bus acts as the front-end of WebSphere MQ to accept service requests from other applications and converts them to WebSphere MQ requests. See [Figure 33-1](#).
- WebSphere MQ sends messages to other applications through Service Bus. See [Figure 33-2](#).

Figure 33-1 Service Bus Front End



Figure 33-2 Messages Sent Through Service Bus



MQ Messaging Types

Service Bus supports the following messaging types:

- [Non-Persistent Messaging](#)
- [Non-XA Persistent Messaging](#)
- [XA Messaging](#)

Non-Persistent Messaging

If you choose to accept an unreliable delivery, such as some missing requests, you can use non-persistent messages where appropriate. WebSphere MQ logging and WebLogic JMS message persistence are only performed for persistent messages; therefore, the use of non-persistent messages eliminates any related I/O activity.

Note:

Non-persistent message throughput is usually limited by the processor speed of the machine. However, in case of a shortage of physical memory, the server system may consume CPU cycles on a paging I/O.

Non-XA Persistent Messaging

WebSphere MQ persistent message throughput is usually limited by the queue manager and the I/O latency writing to the log.

XA Messaging

To enable support for transactional (XA) access to queues, use one of the following transport types:

- **BINDINGS** to access the queue manager when Service Bus is co-located with IBM WebSphere MQ
- **CLIENT** when Service Bus and IBM WebSphere MQ are on different machines. However, with **CLIENT**, you need a special version of the IBM WebSphere MQ client that supports XA transactions, called the WebSphere MQ Extended Transaction Client.

Tip:

For the deployment descriptors to be set appropriately for XA capable resources (JMS, TUXEDO, EJB), you must set the XA attribute on the referenced connection factory before creating a proxy service.

Tuning WebSphere MQ

The following guidelines help you tune WebSphere MQ when you are working with Service Bus. For information specific to WebSphere MQ, see the relevant WebSphere MQ documentation.

- Use the **BINDINGS** transport type if Service Bus and the queue manager are deployed on the same machine.
- If you need XA for only a small section of application requests, create a separate connection object and disable XA.
- Distribute active logs across many volumes. If your system is required to handle high persistent message throughput, you must place the log files on a fast Direct Access Storage Device (DASD) with a minimum of contention from other data set usage. Ideally, you can allocate each of the active logs on separate, low-usage volumes.
- To reduce buffer overflow, tune the buffer pools and pagesets. Buffer overflow results in flushing of the hard disk.
- To avoid broken Service Bus JMS connections to MQ queues, increase the number of active channels to more than 100. By default, the number of active channels is 10.

MQ Transport Configuration Reference

Use the MQ transport in proxy services to retrieve messages from WebSphere MQ. Use the MQ transport in business services to send messages to WebSphere MQ. This section describes how to add WebSphere MQ to your Service Bus environment, characteristics of the Service Bus MQ transport, and how to configure MQ proxy and business services.

- [MQ Transport Endpoint URIs](#)
- [Configuring Proxy Services to Use the MQ Transport](#)
- [Configuring Business Services to Use the MQ Transport](#)
- [MQ Transport Environment Values](#)

MQ Transport Endpoint URIs

When you create an MQ proxy or business service, specify the endpoint URI in the in the following format:

```
mq://local-queue-name?conn=mq-connection-resource-ref
```

where:

- *local-queue-name* is the name of the local queue configured on the MQ server.
- *mq-connection-resource-ref* points to the location of the MQ connection resource.

For example, if you create an MQ connection resource named `mqConnection` in the `defaultMQ` folder and the queue name is `testQueue`, the URI would be `mq://testQueue?conn=defaultMQ/mqConnection`.

Note:

The Endpoint URI cannot contain spaces, so do not create MQ Connection resources, projects, or folders with spaces in the names.

Configuring Proxy Services to Use the MQ Transport

Before you can configure a proxy service for the MQ Transport, you need to create the MQ Connection, as noted in [How to Create MQ Connections](#). During service configuration, select either Message or Any XML as the service type. For more information, see [Creating and Configuring Proxy Services](#).

The following table describes the properties you use to configure an MQ transport for a proxy service.

Table 33-1 MQ Transport Properties for Proxy Services

Property	Description
Is Response Required	Select this option to specify that a response is expected after an outbound message is sent.

Table 33-1 (Cont.) MQ Transport Properties for Proxy Services

Property	Description
Response Correlation Pattern	<p>Select either MessageID or CorrelationID to specify which ID the response correlation pattern should be based on.</p> <p>This option is available only when the Is Response Required check box is selected.</p>
MQ Response URI	<p>Enter the destination to which the response should be published. Enter a response URI in the same format as the endpoint URI:</p> <pre>mq://local-queue-name?conn=mq-connection-resource-ref</pre> <p>Where:</p> <ul style="list-style-type: none"> local-queue-name is the name of the MQ queue where responses will be sent. mq-connection-resource-ref is the path (project/folder) and name of the MQ connection resource; for example, default/my_MQconnection. <p>This option is available only when the Is Response Required check box is selected.</p>
Response Message Type	<p>Select the message type for the response from one of the following:</p> <ul style="list-style-type: none"> Bytes (for a stream of uninterpreted bytes) Text (for text messages) <p>This option is available only when the Is Response Required check box is selected.</p>
Transaction Timeout	<p>Enter the amount of time in seconds to wait before timing out an XA transaction initiated by the proxy service. This property only applies to services using an MQ connection resource that is configured for XA transactions (that is, the XA Enabled option is selected for the MQ connection resource).</p> <p>The default is 300 seconds.</p>
Polling Interval	<p>Enter an interval in milliseconds to wait between polling for messages. The default is 1000.</p>
Poller Dispatch Policy	<p>Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for the poller threads. The default Work Manager is used if no other Work Manager exists. For information about Work Managers, see:</p> <ul style="list-style-type: none"> Using Work Managers with Service Bus "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>
Backout Threshold	<p>Enter a value representing the number of times the pipeline should retry a message before redirecting the message to the queue specified in the Dead Letter URI field.</p> <p>If you do not specify a value for this field, the message is redirected to the dead letter queue without attempting any retries.</p>

Table 33-1 (Cont.) MQ Transport Properties for Proxy Services

Property	Description
MQ Dead Letter URI	<p>Enter the URI of the dead letter queue to which request messages should be redirected after attempting the number of retries specified in the Backout Threshold field.</p> <p>If you do not specify a value for this field, the message is returned to the queue and ignored by the MQ transport for each poll after retrying the number of times specified in the Backout Threshold field. The dead letter URI uses the same format as the endpoint URI.</p>
Endpoint URI 'GET' options	<p>Enter the MQ GET message options from among the following:</p> <ul style="list-style-type: none"> • MQC.MQGMO_ACCEPT_TRUNCATED_MSG • MQC.MQGMO_ALL_MSGS_AVAILABLE • MQC.MQGMO_BROWSE_FIRST • MQC.MQGMO_BROWSE_NEXT • MQC.MQGMO_COMPLETE_MSG • MQC.MQGMO_CONVERT • MQC.MQGMO_FAIL_IF QUIESCING • MQC.MQGMO_LOCK • MQC.MQGMO_LOGICAL_ORDER • MQC.MQGMO_MARK_BROWSE_CO_OP • MQC.MQGMO_MARK_SKIP_BACKOUT • MQC.MQGMO_NO_SYNCPOINT • MQC.MQGMO_NONE • MQC.MQGMO_NO_WAIT • MQC.MQGMO_SYNCPOINT • MQC.MQGMO_SYNCPOINT_IF_PERSISTENT • MQC.MQGMO_UNLOCK • MQC.MQGMO_UNMARK_BROWSE_CO_OP • MQC.MQGMO_UNMARK_BROWSE_HANDLE • MQC.MQGMO_UNMARKED_BROWSE_MSG • MQC.MQGMO_VERSION_1 • MQC.MQGMO_VERSION_2 • MQC.MQGMO_VERSION_3 • MQC.MQGMO_WAIT <p>You can use either " " or "+" to separate multiple options. For example, you can specify the following: MQC.MQGMO_ACCEPT_TRUNCATED_MSG MQC.MQGMO_LOCK</p> <p>The MQ GET message options are applied when reading a message from the inbound queue.</p> <p>For information about how the MQ transport handles RFH2 headers, see About RFH2 Headers.</p>
Process RFH2 Headers	<p>Select this option to parse WebSphere MQ RFH2 headers from a message payload and automatically generate an RFH2Headers transport header containing the RFH2 data. If you do not select this option, the payload is passed through as received.</p> <p>For information about how the MQ transport handles RFH2 headers, see About RFH2 Headers.</p>

Table 33-1 (Cont.) MQ Transport Properties for Proxy Services

Property	Description
Ignore Reply-To Headers	Select this option to ignore the reply-to headers that specify the queue manager.
Dynamic Reply-To Headers	Select this option to use the reply-to headers from the request message in the response message as well. When this property is set to true and the Ignore Reply-To Headers property is set to false, if the static reply-to queue manager (defined in proxy service's response URI) is not the same as the queue manager specified in the request message, the message is sent to the queue manager specified in the request message.
Worker Thread Dispatch Policy	Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists. For information about Work Managers, see: <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>

Configuring Business Services to Use the MQ Transport

Before you can configure a business service for the MQ Transport, you need to create the MQ Connection, as noted in [How to Create MQ Connections](#). During service configuration, select either Message or Any XML as the service type. For more information, see [Creating and Configuring Business Services](#).

The following table describes the properties you use to configure an MQ transport for a proxy service.

Table 33-2 MQ Transport Properties for Business Services

Option	To create or edit...
Message Type	Select one of the following message types for the messages processed by the service: <ul style="list-style-type: none"> • Bytes (for a stream of uninterpreted bytes) • Text (for text messages)
Is Response Required	Select this option to specify that a response is expected after an outbound message is sent.
Response Correlation Pattern	Specify one of the following options to base the response correlation pattern should on: <ul style="list-style-type: none"> • MessageID • CorrelationID • Dynamic Queue – Select this option if your WebSphere MQ implementation uses dynamic queues for response correlation. The MQ transport supports only temporary dynamic queues. This option is available only when the Is Response Required check box is selected.

Table 33-2 (Cont.) MQ Transport Properties for Business Services

Option	To create or edit...
Auto-generate Correlation Value	<p>Select this check box to automatically generate a CorrelationID or MessageID.</p> <p>This option is available only when the Is Response Required check box is selected.</p>
Model Queue	<p>Enter the name of the model queue used to generate the dynamic queue (for Dynamic Queue Response Correlation Pattern only).</p>
MQ Response URI	<p>Enter the destination to which the response should be published. Enter the response URI in the same format as the endpoint URI:</p> <pre data-bbox="662 638 1252 663">mq://local-queue-name?conn=mq-connection-resource</pre> <p>If you are using dynamic queues, enter the URI in the following format:</p> <pre data-bbox="662 772 1300 798">mq://dynamic_queue_prefix?conn=mq-connection-resource</pre> <p>The <i>dynamic_queue_prefix</i>, which is limited to 32 characters, is used to create the dynamic queue on the MQ server. The queue name becomes the prefix plus a unique ID. For example, if the <i>dynamic_queue_prefix</i> is <i>example</i>, the dynamic queue would be named something like <i>example123129083821</i>.</p> <p>You can also use an asterisk (*) as a wildcard in the dynamic queue response URI. For example:</p> <pre data-bbox="662 1058 976 1108">mq://dynamic_queue_prefix* mq://*</pre> <p>If you do not provide a <i>dynamic_queue_name</i> in the URI, the transport uses the dynamic queue name generated by the MQ server. If you do not provide an explicit <i>mq_connection_resource</i> in the URI (best practice), the transport uses the <i>mq_connection_resource</i> from the endpoint URI.</p> <p>This option is available only when the Is Response Required option is selected.</p>
Response Timeout	<p>Enter the number of seconds to wait for a response before dropping the connection. The default is 300.</p> <p>This option is available only when the Is Response Required check box is selected.</p>
Polling Interval	<p>Enter an interval, in milliseconds, to wait between polling for messages. The default is 1000.</p> <p>This option is available only when the Is Response Required check box is selected.</p>
Poller Dispatch Policy	<p>Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for the poller threads. The default Work Manager is used if no other Work Manager exists.</p> <p>For information about Work Managers, see:</p> <ul data-bbox="667 1835 1333 1923" style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>

Table 33-2 (Cont.) MQ Transport Properties for Business Services

Option	To create or edit...
Dynamic Queue Pooling	<p>Select this option to specify that the server use pooled connections to dynamic queues (for the dynamic queue response correlation pattern only).If you want to use a separate connection pool for dynamic queues, consider configuring a dedicated MQ connection resource for the dynamic queues.</p> <p>Do not select this option if you want to create a new dynamic queue instance on each request (and destroy the queue after the response).</p>
Endpoint URI 'PUT' options	<p>Enter the MQ PUT message options from among the following:</p> <ul style="list-style-type: none"> • MQC.MQPMO_ALTERNATE_USER_AUTHORITY • MQC.MQPMO_DEFAULT_CONTEXT • MQC.MQPMO_FAIL_IF QUIESCING • MQC.MQPMO_LOGICAL_ORDER • MQC.MQPMO_NEW_CORREL_ID • MQC.MQPMO_NEW_MSG_ID • MQC.MQPMO_NO_CONTEXT • MQC.MQPMO_NO_SYNCPOINT • MQC.MQPMO_NONE • MQC.MQPMO_PASS_ALL_CONTEXT • MQC.MQPMO_PASS_IDENTITY_CONTEXT • MQC.MQPMO_RESOLVE_LOCAL_Q • MQC.MQPMO_SET_ALL_CONTEXT • MQC.MQPMO_SET_IDENTITY_CONTEXT • MQC.MQPMO_SYNCPOINT • MQC.MQPMO_VERSION_1 • MQC.MQPMO_VERSION_2 <p>You can use either " " or "+" to separate multiple options. For example, you can specify the following: MQC.MQPMO_LOGICAL_ORDER MQC.MQPMO_NEW_MSG_ID</p> <p>The MQ PUT message options are applied when the message is placed in the outbound queue.</p>
MQ Unrecognized Response URI	<p>Enter the URI representing the queue to which unrecognized response messages should be sent. Note that this setting is enabled only when the Auto-generate Correlation Value check box is selected.</p> <p>If you do not specify a value for this field, unrecognized response messages are deleted.</p>
Process RFH2 Headers	<p>Select this option to parse WebSphere MQ RFH2 headers from a message payload and automatically generate an RFH2Headers transport header containing the RFH2 data. If you do not select this option, the payload is passed through as received.</p> <p>For information about how the MQ transport handles RFH2 headers, see About RFH2 Headers.</p>

Table 33-2 (Cont.) MQ Transport Properties for Business Services

Option	To create or edit...
Worker Thread Dispatch Policy	<p>Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for the worker threads. The default Work Manager is used if no other Work Manager exists.</p> <p>For information about Work Managers, see:</p> <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>

MQ Transport Environment Values

Environment values are certain predefined fields in the configuration data whose values are very likely to change when you move your configuration from one domain to another (for example, from test to production). For information about updating environment values, see "Customizing Oracle Service Bus Environments" in *Administering Oracle Service Bus*.

The following table describes the environment values supported by the MQ transport and MQ connection resources. The values you specify for these variables override the properties configured for specific MQ proxy and business services. Services based on the MQ transport also support the `Work Manager` environment value.

Table 33-3 MQ Transport and Connection Resource Environment Values

Value	Description
MQ Connection Pool Size	The number of connections maintained by the MQ connection pool.
MQ Connection Timeout	The time interval in seconds after which unused connections are destroyed.
MQ Dead Letter URI	The URI of the dead letter queue to which request messages should be redirected after attempting the specified number of retries.
MQ Host Name	The host name of the MQ queue manager, for TCP mode connections only.
MQ Port Number	The port number of the MQ queue manager, for TCP mode connections only.
MQ Queue Manager Channel Name	The name of the queue manager server connection channel, for TCP mode connections only.
MQ Queue Manager Name	The name of the MQ queue manager to which to connect.
MQ Response URI	The URI of the destination to which the response should be published when a response is required.
MQ Unrecognized Response URI	The URI representing the queue to which unrecognized response messages should be sent when the correlation value is automatically generated.
MQ Version	The version of WebSphere MQ being used.

Table 33-3 (Cont.) MQ Transport and Connection Resource Environment Values

Value	Description
MQ XA Enabled	An indicator of whether transactions handled by this connection are distributed (XA) transactions.

MQ Transport Headers

The following table lists the headers used by the MQ transport. Most of the headers are common to both outbound requests and inbound response. The Reply To Queue Name, Reply To Queue Manager Name, User ID and Version headers can be edited only for the inbound response.

When you configure a pipeline, you can use a Transport Header action to set the header values in messages.

Table 33-4 MQ Transport Headers

Header	Description	Inbound Response / Outbound Request
Accounting Token	Accounting token is part of the identity context of the message. Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline. Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.	Both
Application ID Data	Application ID data is part of the identity context of the message. This value can be used to provide additional information about the message or its originator. Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline. Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.	Both
Application Origin Data	Data about the originating application. This value can be used by the application to provide additional information about the origin of the message. Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline. Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.	Both
Backout Count	The number of times the message was returned by the MQ Queue, as part of a unit of work, and subsequently backed out. Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline. Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.	Both

Table 33-4 (Cont.) MQ Transport Headers

Header	Description	Inbound Response / Outbound Request
Character Set	<p>The coded character set identifier of character data in the application message data.</p> <p>Inbound Transport Action: This field is used by the inbound transport to convert data in a specific representation. For request-response messaging, the characterSet header from the request message is copied to the response. When this header is not configured on the incoming request, default value of the MQC.MQCCSI_Q_MGR field is assumed.</p> <p>Outbound Transport Action: This header can be set in the pipeline for the outbound transport. If this header value is not set, the default MQC.MQCCSI_Q_MGR value is assumed.</p>	Both
Correlation ID	<p>The correlation-id of the message that should be retrieved.</p> <p>Inbound Transport Action: For correlationID-based response correlation pattern, the correlationID from the request is echoed on the response. The user can override the correlationID in the response pipeline.</p> <p>Outbound Transport Action: When the Auto-generate correlationID option is selected during service configuration, the outbound transport will automatically generate a correlationID and overwrite the correlationID from the transport header. If this value is not specified, the correlationID specified in the pipeline is used.</p> <p>For one-way messaging, the correlationID specified in the pipeline is used in the (outbound) request.</p>	Both
Encoding	<p>The representation used for numeric values in the application message data.</p> <p>Inbound Transport Action: The inbound transport uses this header to interpret the incoming message data. If this header is not configured in the response pipeline, the default value of MQC.MQENC_NATIVE is used.</p> <p>Outbound Transport Action: If this header is not set in the pipeline for the outbound transport, the default value of MQC.MQENC_NATIVE is used.</p>	Both
Expiry	<p>The expiry time (in tenths of a second) is set by the application that puts the message. After a message's expiry time has elapsed, it is eligible to be discarded by the queue manager.</p> <p>Inbound Transport Action: For request-response messaging, the inbound transport copies the expiry header of the request to the response.</p> <p>Outbound Transport Action: If the corresponding transport header is set in the pipeline, it is copied to the outbound request message.</p> <p>Note: The report header will always contain the MQC.MQRO_PASS_DISCARD_AND_EXPIRY option (in addition to others). This option is a directive to the receiving client that the expiry time of the original message should be copied to the report or reply message.</p>	Both

Table 33-4 (Cont.) MQ Transport Headers

Header	Description	Inbound Response / Outbound Request
Feedback	<p>The nature of the feedback report. This value is used with a message of type <code>MQC.MQMT_REPORT</code> to indicate the nature of the report.</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Both
Format	<p>Format name of the message data. The format name is used by the sender of the message to indicate the nature of the data in the message to the receiver.</p> <p>Inbound Transport Action: When the field is set to <code>MQC.MQFMT_MD_EXTENSION</code>, the inbound transport will read the extended MQMD object.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Both
Group ID	<p>The value that identifies the message group to which the physical message belongs.</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Both
Offset	<p>In a segmented message, offset of data in the physical message from the start of the logical message.</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Both
Original Length	<p>Original length of a segmented message.</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Both
Message Flags	<p>Flags that control the segmentation and status of a message.</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Both

Table 33-4 (Cont.) MQ Transport Headers

Header	Description	Inbound Response / Outbound Request
Message ID	<p>ID of the message to be retrieved.</p> <p>Inbound Transport Action: If messageID is not specified in the response pipeline, the messageID header is set to MQC.MQMI_NONE. For messageID-based correlation, the inbound transport copies the messageID from the request to the correlationID header of the response. MessageID-based correlation is assumed when the report header contains the MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID option.</p> <p>Outbound Transport Action: When the Auto-generate messageID option is specified during service configuration, the outbound transport automatically generates the messageID and overwrites the messageID from the transport header. If this value is not specified, the messageID transport header is used.</p> <p>For one-way messaging, the messageID specified in the pipeline is used in the outbound request. If this value is not specified, the messageID is automatically generated by the transport.</p>	Both
Message Sequence Number	<p>Sequence number of a logical message within group.</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Both
Message Type	<p>Message type of the message.</p> <p>Inbound Transport Action: The inbound transport reads and processes messages of any type including MQC.MQMT_REQUEST, MQC.MQMT_DATAGRAM, MQC.MQMT_REPLY and MQC.MQMT_REPORT. The inbound transport does not generate report messages.</p> <p>Outbound Transport Action: The outbound transport generates messages of any type including MQC.MQMT_DATAGRAM, MQC.MQMT_REQUEST, MQC.MQMT_REPLY and MQC.MQMT_REPORT. When the messageType header is not configured in the pipeline, the transport generates messages of type MQC.MQMT_DATAGRAM when the messaging pattern is one-way and MQC.MQMT_REQUEST when the messaging pattern is request-reply.</p>	Both
Persistence	<p>The message persistence.</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Both
Priority	<p>Priority of the message</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Both

Table 33-4 (Cont.) MQ Transport Headers

Header	Description	Inbound Response / Outbound Request
Put Application Name	<p>The name of the application that put the message.</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Both
Put Application Type	<p>The type of the application that put the message.</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Both
Put Date Time	<p>The time and date when the message was put. This is specified in simple date format (yyyy-MM-dd HH:mm:ss.SSS). For example: 2014-03-18 05:17:20.123.</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Both
Reply To Queue Name	<p>The name of the queue to which a reply should be sent.</p> <p>The application that issued the get request for the message can send MQC.MQFMT_REPLY and MQC.MQFMT_REPORT messages to this queue.</p> <p>Inbound Transport Action: The inbound transport uses the replyToQueueName as the response queue name when this field is set. If this values is not set, the queue name is derived from the default destination URI.</p> <p>Outbound Transport Action: In request/response message pattern, replyToQueueName set in the message flow is ignored. In one way message pattern, replyToQueueName set in the message flow is used in the outbound messages.</p>	Inbound Response
Reply To Queue Manager Name	<p>The name of the queue manager to which reply or report messages can be sent.</p> <p>Inbound Transport Action: In request/response message pattern, if the inbound message replyToQueueManager header value does not match the configured value for the queue manager in the response URI, the response message is dropped and a transport error is logged. To override this logic, set the Dynamic Reply-To Headers property to true and the Ignore Reply-To Headers property to false in the proxy service. Then, if the headers values do not match, the message is sent to the queue manager specified in the request message.</p> <p>Outbound Transport Action: In request/response message pattern, replyToQueueManager set in the message flow is ignored. In one way message pattern, replyToQueueManager set in the message flow is used in the outbound messages.</p>	Inbound Response

Table 33-4 (Cont.) MQ Transport Headers

Header	Description	Inbound Response / Outbound Request
Report	<p>A report is a message about another message. This field enables the application sending the original message to specify which report messages are required, whether the application message data is to be included in them, and also how the message and correlation ID in the report or reply are to be set. It comprises one or more constants from the MQC class combined by means of the '+' or ' ' operators.</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline. For request-response messaging, this header can be configured in the response pipeline.</p> <p>Outbound Transport Action: The transport always sets a combination of the following options in the report field.</p> <p>Set <code>MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID</code> if messageID-based correlation pattern is used and <code>MQC.MQRO_PASS_CORREL_ID</code> if correlationID-based correlation pattern is used. Always set <code>MQC.MQRO_PASS_DISCARD_AND_EXPIRY</code>.</p> <p>Note: These options are set in addition to the options specified on the corresponding transport header in the pipeline.</p>	Both
User ID	<p>It is part of the identity of the message and identifies the user who originated the message.</p> <p>Inbound Transport Action: No explicit processing is done by the transport. The header is copied to the transport header in the pipeline.</p> <p>Outbound Transport Action: No explicit processing is done by the transport. If the corresponding transport header is set in the pipeline, it is copied to the request message.</p>	Inbound Response
Version	<p>The version number of the message descriptor.</p> <p>Inbound Transport Action: The inbound transport supports both version 1 and version 2 message descriptors.</p> <p>Outbound Transport Action: By default, the outbound transport generates version 2 headers. However, this field can be overridden in the pipeline.</p>	Inbound Response
RFH2Headers	<p>The RFH2 headers in the payload when the Process RFH2 Headers option is set in the transport configuration. The RFH2Headers header is a String.</p> <p>Inbound Transport Action: RFH2 headers are extracted from the MQ payload to construct the corresponding transport metadata header.</p> <p>Outbound Transport Action: RFH2Headers data are parsed to extract the RFH2 headers, which are inserted (along with the content length for each header) into the outbound MQ payload.</p>	Both

Configuring Transport Headers

You can configure the transport headers for both inbound and outbound requests in the pipeline using a Transport Header action to set the header values in messages. For more information, see [Adding Transport Header Actions in the Console](#).

When the transport header is explicitly set in the pipeline, this value overrides the header value except in the following scenarios:

- For the outbound request-response pattern, when the `Auto-generate Correlation Value` option is selected for a outbound request with a request-response message pattern, the correlation ID is always generated even if this value is set in the message flow.
- When the report header is set in the message flow, the combination of multiple directives associated with the report header are merged with the default directives.
- When the `replyToQueueManagerName` or `replyToQueueName` headers are set in the message flow for an outbound request with a request/response message pattern, these values are ignored. Instead, these transport header values are derived from the response URI configured for the business service.
- For the inbound response, when the value in the `replyToQueueManagerName` header does not match the queue manager name specified in the response URI, an error message is generated and the response message is not sent.
- The `replyToQueueName` set in the inbound request header overrides the value configured in the reply to URI for the proxy service.
- For a one-way business service, when the message type header is configured to be of type request in the message flow, the `replyToQueueName` header must be specified. If this value is not specified, an error is generated on the MQ server and the message is rolled back.

About RFH2 Headers

RFH2Header headers can contain multiple `<RFH2Header>` blocks, each of which can contain multiple folders. The MQ transport consolidates the blocks into a single RFH2 header containing a linear list of folders.

For example, the following blocks are consolidated into a single RFH2 header:

```
<RFH2Header>
  <mcd><Msd>jms_bytes</Msd></mcd>
</RFH2Header>
<RFH2Header>
  <usr><clientId>DASHBOARD</clientId></usr>
</RFH2Header>
```

Using the Oracle BPEL Process Manager Transport

This chapter provides an overview of the BPEL transport and describes how to use and configure it in your services. The BPEL transport lets you bring Oracle BPEL Process Manager (Oracle BPEL PM) into your service oriented architecture (SOA) environment using Service Bus.

Note:

The BPEL transport (bpel-10g in the user interface) is for messaging with only Oracle SOA Suite 10g Release 3. Service Bus provides the SOA-DIRECT transport for use with SOA Suite 11g and later. For more information, see [Using the SOA-DIRECT Transport](#).

This chapter contains the following sections:

- [Introduction to the BPEL Transport](#)
- [BPEL Transport Simple Use Cases \(Synchronous\)](#)
- [Advanced Use Cases \(Asynchronous\)](#)
- [BPEL Transport Configuration Reference](#)
- [BPEL Transport Security](#)
- [BPEL Transport Error Handling](#)
- [WS-Addressing Reference](#)
- [Examples of XML Messaging with the BPEL Transport](#)

Introduction to the BPEL Transport

Service Bus provides support for communication with Oracle BPEL Process Manager, letting you include BPEL processes in your service oriented architecture (SOA). With Service Bus's native BPEL transport for Oracle BPEL PM, you can expose BPEL processes as web services in the service bus layer, letting other services invoke BPEL processes. Likewise, Oracle BPEL PM can call services in the service bus layer for use in its own processes.

The built-in integration between Service Bus and Oracle BPEL Process Manager is enabled at the Service Bus API level. This chapter focuses on this type of integration, but you are not limited to only these solutions. You can also use other standard communication protocols provided with Service Bus, such as HTTP, JMS, and File.

For detailed information on Oracle BPEL Process Manager, go to <http://www.oracle.com/technetwork/middleware/bpel/overview/index.html>.

SOAP Support with the BPEL Transport

Communication between Oracle BPEL Process Manager and Service Bus is done over SOAP only. Service Bus and Oracle BPEL PM do not provide full support for SOAP RPC encoding. The BPEL transport accepts SOAP RPC encoding bindings, but some encoding mechanisms, such as multiRef, might cause runtime failures.

The BPEL transport supports the following features:

- SOAP 1.1. SOAP 1.2 is supported only from Service Bus to Oracle BPEL PM using synchronous communication.
- SOAP headers

The BPEL transport has the following restrictions:

- No attachments
- No WS-Security or WS-RM

Transaction Propagation in the BPEL Transport

Oracle BPEL PM supports transaction propagation through its API, and the BPEL transport is transactional to support transaction propagation when Oracle BPEL PM is deployed on Oracle WebLogic Server. For example, if a process begins in a service outside of Oracle BPEL PM, Service Bus can propagate the transaction to Oracle BPEL PM through the BPEL transport to complete the transaction.

Transaction propagation is also supported from Oracle BPEL PM to Service Bus using an SB transport-based proxy service.

Note:

Transaction propagation is not yet supported for Oracle servers OC4J and Oracle AS and not yet certified on IBM WebSphere.

SSL Support in the BPEL Transport

Calls from Service Bus to Oracle BPEL PM are made using RMI, so the BPEL transport supports security at the call level through one-way SSL. For more information, see [BPEL Transport Endpoint URI](#).

BPEL Transport Environment Values

The BPEL transport declares the environment variables listed in the following table, each of which corresponds to a property you specify when you configure a BPEL transport. You can update the values for environment variables when moving Service Bus projects among different deployment environments without having to update the transport configuration itself. For more information about environment values, see "Customizing Oracle Service Bus Environments" in *Administering Oracle Service Bus*.

Table 34-1 BPEL Transport Environment Variables

Environment Variable	Description
Dispatch Policy	Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists. For information about Work Managers, see: <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>
Service Account	A Service Bus service account resource to use for JNDI context security to access the Oracle BPEL Process Manager delivery service.
Endpoint URI	The URI of the service.

BPEL Transport Simple Use Cases (Synchronous)

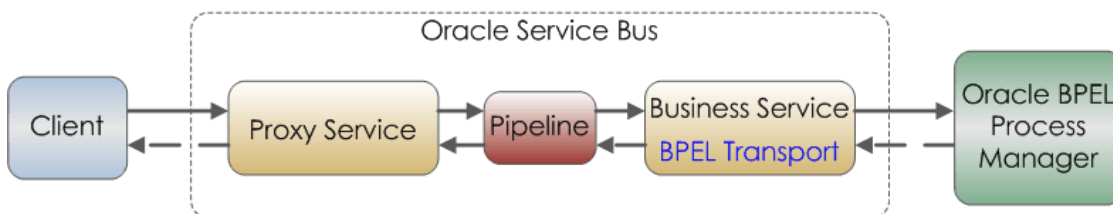
This section describes the most common use cases for communicating to and from Oracle BPEL Process Manager through Service Bus. These represent simple synchronous one-way or request/response communication.

- [Synchronous: Invoking Processes in Oracle BPEL Process Manager](#)
- [Synchronous: Calling External Services from Oracle BPEL Process Manager](#)
- [Associating Messages with the Correct Conversation](#)

Synchronous: Invoking Processes in Oracle BPEL Process Manager

Figure 34-1 illustrates a synchronous communication pattern between a client and Oracle BPEL Process Manager through Service Bus.

Figure 34-1 Invoking Oracle BPEL Processes Synchronously Through Service Bus



Creating and Configuring the Services

Use the following guidelines to invoke Oracle BPEL Process Manager processes from a client:

- Create a Service Bus business service that represents the BPEL process you want to invoke.
 - Create a WSDL-based business service. Generate the WSDL file from Oracle BPEL Process Manager.

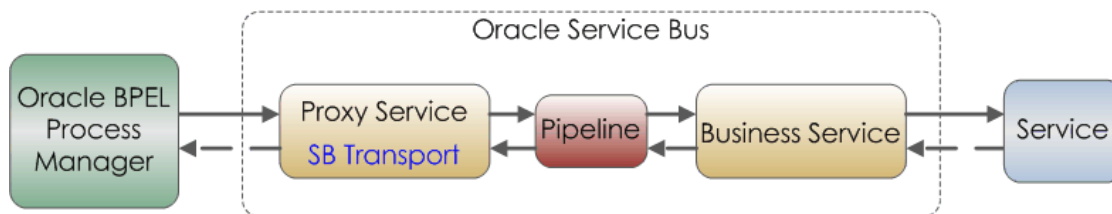
- Select the bpel-10g transport in the business service configuration.
- Set the endpoint URI as described in [Table 34-2](#).
- Configure the remainder of the business service. See [BPEL Transport Configuration Reference](#).
- Create a proxy service in Service Bus that accepts messages from the client.
- Create a pipeline in Service Bus that invokes the business service from the proxy service.

To ensure that messages are associated with the correct conversation, see [Associating Messages with the Correct Conversation](#).

Synchronous: Calling External Services from Oracle BPEL Process Manager

[Figure 34-2](#) illustrates a synchronous communication pattern between Oracle BPEL Process Manager and a service provider through Service Bus.

Figure 34-2 Oracle BPEL Processes Invoking a Service Synchronously Through Service Bus



Creating and Configuring the Services

Use the following guidelines to invoke an external service from Oracle BPEL Process Manager:

- Create a business service in Service Bus that represents the external service you want to invoke.
- Create a proxy service and its associated pipeline in Service Bus that invokes the business service.
 - You must create the proxy service with a SOAP WSDL file to invoke the business service. When defining your proxy service, select the WSDL service type, and select the desired port or binding.
 - Select the SB transport in the proxy service configuration.
 - To invoke the proxy service from Oracle BPEL Process Manager, export the proxy service's effective WSDL file and import it into your Oracle BPEL Process Manager development environment. Invoke the proxy service from Oracle BPEL Process Manager as you normally would.

For configuration information, see the online help provided with Service Bus.

To ensure that messages are associated with the correct conversation, see [Associating Messages with the Correct Conversation](#).

Associating Messages with the Correct Conversation

When using stateful services, the messages sent synchronously between Service Bus and Oracle BPEL Process Manager are known as a conversation. Oracle BPEL Process Manager supports the following mechanisms for ensuring that messages are correctly associated with each other as part of a conversation. These mechanisms are independent of each other, and you may choose to use both to ensure correct association.

- **BPEL Correlation:** BPEL correlation is part of the BPEL specification. When a WSDL-based business service in Service Bus sends a message to a BPEL process, the BPEL engine examines the message to find the target BPEL process instance.
- **Opaque Correlation using WS-Addressing:** When a conversation is initiated by a client through Service Bus to a BPEL process, the BPEL engine looks in the WS-Addressing SOAP header for the "messageID" value to use as the ID for the new conversation. The conversation ID is carried through the conversation as the "RelatesTo" value.

For more information on WS-Addressing, see [MessageID / RelatesTo](#) in the WS-Addressing reference. For an example of conversation ID setting, see [Conversation ID Examples](#).

Advanced Use Cases (Asynchronous)

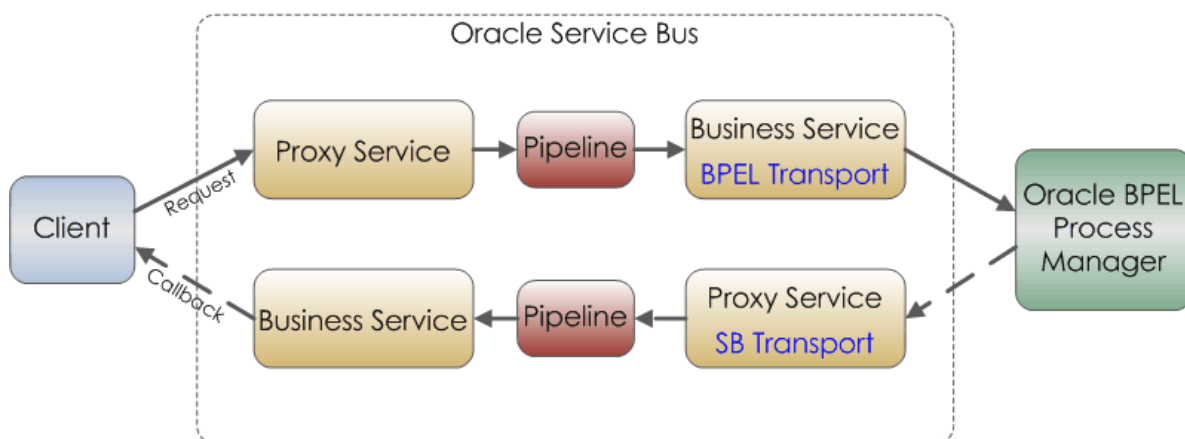
This section describes more advanced use cases for communicating to and from Oracle BPEL Process Manager through Service Bus using asynchronous communication.

- [Asynchronous: Invoking Processes in Oracle BPEL Process Manager](#)
- [Asynchronous: Calling Service Providers from Oracle BPEL Process Manager](#)

Asynchronous: Invoking Processes in Oracle BPEL Process Manager

[Figure 34-3](#) illustrates an asynchronous communication pattern between Oracle BPEL Process Manager and a service provider through Service Bus.

Figure 34-3 *Invoking Oracle BPEL Processes Asynchronously Through Service Bus*



In an asynchronous message exchange, a callback is sent on a different connection than the request.

Creating and Configuring the Services

Use the following guidelines to invoke Oracle BPEL Process Manager processes asynchronously from a client through Service Bus:

Create two proxy services in Service Bus with their associated pipelines. One that invokes the business service and another that handles the callback.

- **Request Proxy Service and Pipeline:** Since the callback will be sent on a different connection in asynchronous communication, you must establish the callback address in the request proxy. This callback address will be passed to the callback proxy and callback business services so that the message is sent back to the correct client.

As part of the business service configuration, you select a callback proxy. At runtime, the BPEL transport uses this proxy as the callback proxy. For approaches to setting a callback address if you do not select a callback proxy in the business service, see [WS-Addressing Reference](#) and [Asynchronous BPEL to BPEL Through Service Bus Example](#).

- **Callback Proxy Service and Pipeline:** Configure the proxy to use the WSDL SOAP or Any SOAP service type and the SB or HTTP transport. Use the SB transport if you want transaction propagation from Oracle BPEL Process Manager to Service Bus.

If you select this proxy service as the business service's callback proxy, the BPEL transport provides the correct callback URI at runtime.

Create two business services in Service Bus: one that makes the request to the Oracle BPEL Process Manager process you want to interact with and another that handles the callback.

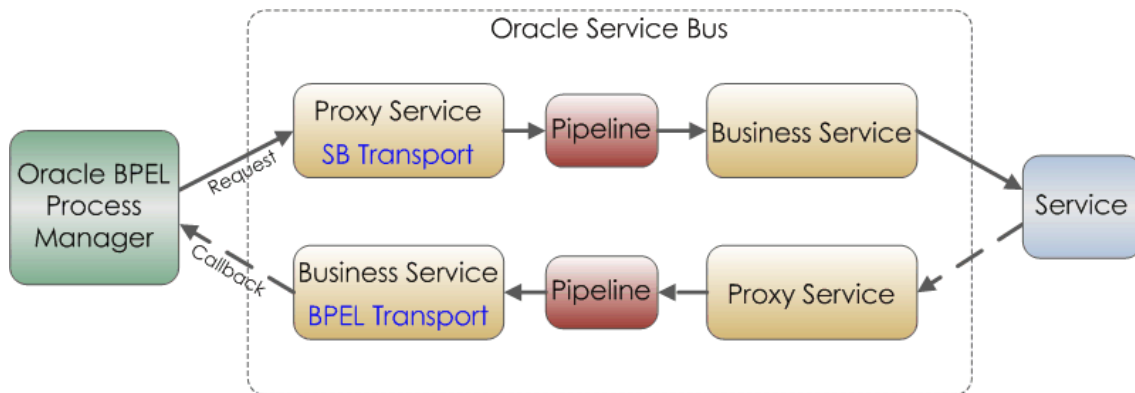
- **Request Business Service:** Create a WSDL-based business service. Generate the WSDL file from Oracle BPEL Process Manager. Select the WSDL service type, and select the appropriate binding or port in the WSDL file.
 - Select the `bpel-10g` transport in the business service configuration.
 - Set the role to Asynchronous Client.
 - Specify the endpoint URI, described in [Table 34-2](#).
 - Use the Callback Proxy field on the BPEL transport configuration page to select the callback proxy you created.
- **Callback Business Service:** Configure the business process you need to handle the callback.

See the online help provided with Service Bus for configuration information not covered in this guide.

Asynchronous: Calling Service Providers from Oracle BPEL Process Manager

This section describes the steps and configuration needed for Oracle BPEL Process Manager to make service calls through Service Bus.

[Figure 34-4](#) illustrates an asynchronous communication pattern between Oracle BPEL Process Manager and a service provider through Service Bus.

Figure 34-4 Oracle BPEL Processes Invoking a Service Asynchronously Through Service Bus

In an asynchronous message exchange, a callback is sent on a different connection than the request.

Creating and Configuring the Services

Use the following guidelines to invoke an external service asynchronously from Oracle BPEL Process Manager through Service Bus.

Create two proxy services and their associated pipelines in Service Bus: one for the request that invokes the business service and another that handles the callback.

- **Request Proxy Service and Pipeline:** Configure the proxy service to use the SB transport. Since the callback will be sent on a different connection in asynchronous communication, you must establish a callback address so the message is sent back to the correct client.

For information on setting a callback address, see [ReplyTo](#) of the WS-Addressing reference and [Asynchronous BPEL to BPEL Through Service Bus Example](#).

- **Callback Proxy Service and Pipeline:** Configure the proxy service to pass the callback address to the business service. The callback URI is provided in the request. Use URI rewriting to extract the callback URI and forward it to the business service.

For proxy configuration details, see [Creating and Configuring Proxy Services](#) and [Using the SB Transport](#).

Create two business services in Service Bus: a request business service that invokes the external service and a callback business service.

- **Request Business Service:** Configure the business service to invoke the external service.
- **Callback Business Service:** The callback business service receives the callback address from the callback proxy. The URI rewriting performed by the callback proxy service determines which BPEL process to send the response to.
 - Create a WSDL-based business service. Generate the WSDL file from Oracle BPEL Process Manager. Select the WSDL service type and select the appropriate binding or port in the WSDL file.
 - Select the `bpel-10g` transport in the business service configuration.
 - Set the Endpoint URI to `bpel://callback`, as described in [Table 34-2](#). The callback URI is provided by the callback proxy service.

Note:

If the callback address is always known, for example when the client and BPEL service are linked together because of a trading partner agreement, you can provide the exact callback address in the callback business service instead of using `bpel://callback`.

- Set the role to Service Callback on the `bpel-10g` transport configuration page, as described in [Table 34-3](#).
- Configure the remainder of the business service. See the online help provided with Service Bus for configuration information not covered in this document.

BPEL Transport Security

Security in Oracle BPEL Process Manager is handled at the EJB level. The BPEL transport supports a JNDI service account option used for the creation of the JNDI context and the EJB lookup, as described in [Table 34-3](#). The BPEL transport gets the user name and password from the service account. If the service account is not configured, an anonymous subject is assumed.

The BPEL transport also supports security at the call level by letting you indicate one-way SSL in the protocol portion of the URI from Service Bus to Oracle BPEL Process Manager. For more information, see [BPEL Transport Endpoint URI](#).

Using SSL from Oracle Service Bus to Oracle Servers

To use SSL from Service Bus to OC4J and Oracle AS servers (using the `ormis` and `opmns` protocols), you must configure SSL on your Service Bus server by adding the following properties to your domain's `setDomainEnv` script:

```
-Djavax.net.ssl.trustStorePassword=passphrase  
-Djavax.net.ssl.trustStore=file path to a keystore of trust certificate
```

BPEL Transport Error Handling

The BPEL transport handles Oracle BPEL Process Manager exceptions in different ways.

- [Application Errors](#)
- [Connection Errors](#)
- [Other Errors](#)

Application Errors

If a BPEL process replies with a fault, the BPEL transport intercepts the fault message at the API and translates it into a SOAP fault. The transport's automated application error-handling functionality lets you decide whether or not to automatically retry application errors—in this case BPEL faults—when they occur. For example, you may determine that application errors will always fail until the problem is fixed. Use the **Retry Application Errors** option on the transport configuration page to turn application retries on and off.

Connection Errors

Oracle BPEL Process Manager can throw the following non-fault exceptions, which the BPEL transport automatically categorizes as `TransportException` connection errors:

- `NamingException`
- `RemoteException`

The transport's automated connection error-handling functionality lets you determine if and how often connection errors are retried. Use the **Retry Count** and the **Retry Iteration Interval** options on the transport configuration page to control the number of retries and the number of seconds to wait between retries on connection errors. Setting `Retry Iteration Interval` to zero (0) means connection errors are not retried.

Other Errors

Other non-application and non-connection exceptions are re-thrown as generic `TransportException` errors.

WS-Addressing Reference

This section describes specific WS-Addressing properties that the BPEL transport uses to communicate with Oracle BPEL Process Manager. This section also describes ways to provide callback addresses in asynchronous communication, as described in the [Advanced Use Cases \(Asynchronous\)](#) guidelines.

See [Examples of XML Messaging with the BPEL Transport](#) for WS-Addressing examples.

- [ReplyTo](#)
- [MessageID / RelatesTo](#)

ReplyTo

In asynchronous communication, a service callback is sent on a different connection than the request. As a service developer, you must supply the correct callback address in an asynchronous exchange so that the callback is sent to the correct client.

Calling a BPEL Process Asynchronously Through Service Bus

The BPEL transport provides a built-in mechanism for providing the correct callback address. When you create a BPEL business service in Service Bus, you can select a callback proxy to handle the callback, and the BPEL transport automatically sets the correct callback address. You do not need to manually use "ReplyTo."

For more information, see [Asynchronous: Invoking Processes in Oracle BPEL Process Manager](#).

BPEL Processes Calling External Services Through Service Bus

When calling an external service from Oracle BPEL Process Manager through Service Bus, you must manually set a callback address. To do this using WS-Addressing, in the request proxy service set the callback address as the "ReplyTo" value. The BPEL transport in the business service uses that URI as the callback address.

For more information, see [Asynchronous: Calling Service Providers from Oracle BPEL Process Manager](#).

MessageID / RelatesTo

"MessageID" and "RelatesTo" are used to store the conversation ID in conversations between Service Bus and Oracle BPEL Process Manager, making sure related messages remain in the same conversation. The BPEL transport does not let you specify whether a given operation is a start or continue operation. Instead, the BPEL transport looks for the "MessageID" and "RelatesTo" properties and sets them accordingly.

The following describes how the BPEL transport uses "MessageID" and "RelatesTo" in synchronous and asynchronous conversations:

- **Synchronous conversation:** In the initial request, the "MessageID" determines the conversation ID. In the remaining communication, the BPEL transport provides the conversation ID as the RelatesTo value.

If there is no value assigned to "MessageID" or "RelatesTo," the transport assumes either no conversation is occurring or that Oracle BPEL Process Manager is handling the correlation.

- **Asynchronous callbacks:** In the initial request, the "MessageID" determines the conversation ID. In the remaining communication, the BPEL transport provides the conversation ID as the "RelatesTo" value in the callback.

If there is no value assigned to "MessageID" or "RelatesTo," the transport assumes either no conversation is occurring or that Oracle BPEL Process Manager is handling the correlation.

For more implementation on establishing a conversation ID to make sure messages participate in the correct conversation, see [Associating Messages with the Correct Conversation](#) and the [Conversation ID Examples](#).

Examples of XML Messaging with the BPEL Transport

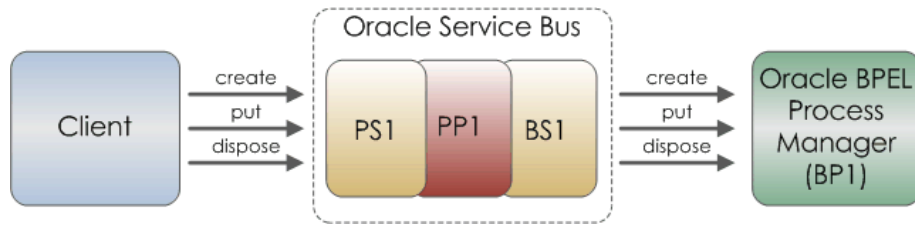
Following are examples of XML messaging issues between Service Bus and Oracle BPEL Process Manager.

- [Conversation ID Examples](#)
- [Asynchronous BPEL to BPEL Through Service Bus Example](#)

Conversation ID Examples

This section provides different examples of establishing a conversation ID among messages in a conversation between Service Bus and Oracle BPEL Process Manager.

In [Figure 34-5](#), a client is synchronously invoking a process in Oracle BPEL Process Manager. The business service (BS1) uses the BPEL transport to invoke a process. The pipeline (PP1) handles any necessary conversation ID mapping and passes the messages from the proxy service (PS1) to the business service.

Figure 34-5 Operations in a Synchronous Exchange Through Service Bus

Port and Message Definitions

The examples in this section use the following port and message definitions defined in the WSDL file.

```

<wsdl:types>
  <xsd:schema
    targetNamespace="http://www.sample.org/spec/samples/types"
    elementFormDefault="qualified">
    <xsd:complexType name="ValueHolder">
      <xsd:all>
        <xsd:any minOccurs="1"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:schema>
</wsdl:types>
<message name="create"/>
<message name="putRequest">
  <part name="key" type="xsd:string"/>
  <part name="value" type="types:ValueHolder"/>
</message>
<message name="putResponse">
  <part name="value" type="types:ValueHolder"/>
</message>
...
<message name="dispose"/>
<portType name="ServiceMap">
  <operation name="create">
    <input message="tns:create"/>
  </operation>
  <operation name="put">
    <input message="tns:putRequest"/>
    <output message="tns:putResponse"/>
  </operation>
  ...
  <operation name="dispose">
    <input message="tns:dispose"/>
  </operation>
</portType>
  
```

WS-Addressing that Sets the Conversation ID

This example shows how WS-Addressing is used to set the conversation ID among messages in a conversation. [Figure 34-5](#) shows communication pattern.

Create Operation

```

<soap:Envelope>
  <soap:Header>
    <wsa03:MessageID>uuid:123456789</wsa03:MessageID>
  </soap:Header>
  <soap:Body>
  
```

```
        <create/>
    </soap:Body>
</soap:Envelope>
```

Put Operation

```
<soap:Envelope>
  <soap:Header>
    <wsa03:MessageID>uuid:111111111</wsa03:MessageID>
    <wsa03:RelatesTo>uuid:123456789</wsa03:RelatesTo>
  </soap:Header>
  <soap:Body>
    <put>
      <key>key</key>
      <value>
        <PO/>
      </value>
    </put>
  </soap:Body>
</soap:Envelope>
<soap:Envelope>
  <soap:Body>
    <putResponse>
      <value/>
    </putResponse>
  </soap:Body>
</soap:Envelope>
```

The <put> operation also has a MessageID, but it is ignored because the RelatesTo has a value that provides the conversation ID.

Message Payload Data that Sets the Conversation ID

This example shows how message payload data can be used to set the conversation ID among messages in a conversation. In these examples, the proxy service maps the ID to the MessageID / RelatesTo SOAP headers. [Figure 34-5](#) shows communication pattern.

Create Operation

Client to proxy service

```
<soap:Envelope>
  <soap:Body>
    <create/>
  </soap:Body>
</soap:Envelope>
<soap:Envelope>
  <soap:Body>
    <createResponse>
      <mapID>uuid:123456789</mapID>
    </createResponse>
  </soap:Body>
</soap:Envelope>
```

Proxy service to BPEL process (using a business service)

```
<soap:Envelope>
  <soap:Header>
    <wsa03:MessageID>uuid:123456789</wsa03:MessageID>
  </soap:Header>
  <soap:Body>
```

```

    <create/>
  </soap:Body>
</soap:Envelope>

```

Not shown: The ID was generated in the request of the proxy service pipeline and inserted as a `<wsa03:MessageID>` before invoking the process. On the process side, the `create` operation is one-way, so a SOAP response must be created before replying to the client. The response sends back the ID that was generated by the proxy service.

Put Operation

Client to proxy service

```

<soap:Envelope>
  <soap:Body>
    <put>
      <mapID>uuid:123456789</mapID>
      <key>key</key>
      <value>
        <PO/>
      </value>
    </put>
  </soap:Body>
</soap:Envelope>
<soap:Envelope>
  <soap:Body>
    <putResponse>
      <value/>
    </putResponse>
  </soap:Body>
</soap:Envelope>

```

Proxy service to BPEL process (using a business service)

```

<soap:Envelope>
  <soap:Header>
    <wsa03:RelatesTo>uuid:123456789</wsa03:RelatesTo>
  </soap:Header>
  <soap:Body>
    <put>
      <key>key</key>
      <value>
        <PO/>
      </value>
    </put>
  </soap:Body>
</soap:Envelope>
<soap:Envelope>
  <soap:Body>
    <putResponse>
      <value/>
    </putResponse>
  </soap:Body>
</soap:Envelope>

```

Dispose Operation

Client to proxy service

```

<soap:Envelope>
  <soap:Body>

```

```
        <dispose>
          <mapID>uuid:123456789</mapID>
        </dispose>
      </soap:Body>
    </soap:Envelope>
```

Proxy service to BPEL process (using a business service)

```
<soap:Envelope>
  <soap:Header>
    <wsa03:RelatesTo>uuid:123456789</wsa03:RelatesTo>
  </soap:Header>
  <soap:Body>
    <dispose/>
  </soap:Body>
</soap:Envelope>
```

Transformation Examples

In these examples, the client uses a more recent version of the WS-Addressing spec (*wsa04* prefix). The proxy service is responsible for transforming the SOAP headers to use the *wsa03* prefix. The proxy service developer configures the transformation. [Figure 34-5](#) shows communication pattern.

Create Operation

Client to proxy service

```
<soap:Envelope>
  <soap:Header>
    <wsa04:MessageID>uuid:123456789</wsa04:MessageID>
  </soap:Header>
  <soap:Body>
    <create/>
  </soap:Body>
</soap:Envelope>
```

Proxy service to BPEL process (using a business service)

```
<soap:Envelope>
  <soap:Header>
    <wsa03:MessageID>uuid:123456789</wsa03:MessageID>
  </soap:Header>
  <soap:Body>
    <create/>
  </soap:Body>
</soap:Envelope>
```

Put Operation

Client to proxy service

```
<soap:Envelope>
  <soap:Header>
    <wsa04:MessageID>uuid:111111111</wsa04:MessageID>
    <wsa04:RelatesTo>uuid:123456789</wsa04:RelatesTo>
  </soap:Header>
  <soap:Body>
    <put>
      <key>key</key>
      <value>
        <PO/>
      </value>
    </put>
  </soap:Body>
</soap:Envelope>
```

```

    </put>
  </soap:Body>
</soap:Envelope>
<soap:Envelope>
  <soap:Body>
    <putResponse>
      <value/>
    </putResponse>
  </soap:Body>
</soap:Envelope>

```

Proxy service to BPEL process (using a business service)

```

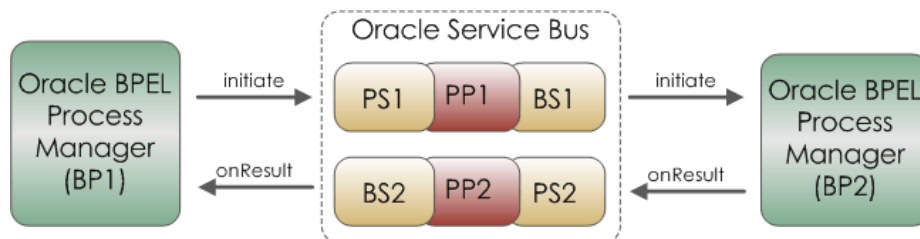
<soap:Envelope>
  <soap:Header>
    <wsa03:MessageID>uuid:11111111</wsa03:MessageID>
    <wsa03:RelatesTo>uuid:123456789</wsa03:RelatesTo>
  </soap:Header>
  <soap:Body>
    <put>
      <key>key</key>
      <value>
        <PO/>
      </value>
    </put>
  </soap:Body>
</soap:Envelope>
<soap:Envelope>
  <soap:Body>
    <putResponse>
      <value/>
    </putResponse>
  </soap:Body>
</soap:Envelope>

```

Asynchronous BPEL to BPEL Through Service Bus Example

The following example shows the SOAP headers involved in one BPEL process invoking another BPEL process asynchronously through Service Bus. In [Figure 34-6](#), PP1 and PP2 are pipelines that perform transformations and pass messages from PS1 and PS2 proxy services to BS1 and BS2 business services. The business services are required to make calls to BPEL processes using the BPEL transport.

Figure 34-6 One BPEL Process Invoking Another BPEL Process Through Service Bus



Refer to [Figure 34-6](#) for the following SOAP header examples.

Port and Message Definitions

```

<message name="LoanServiceRequestMessage">
  <part name="payload" element="types:loanApplication"/>

```

```

</message>
<message name="LoanServiceResultMessage">
  <part name="payload" element="types:loanOffer"/>
</message>
<portType name="LoanService">
  <operation name="initiate">
    <input message="tns:LoanServiceRequestMessage"/>
  </operation>
</portType>
<portType name="LoanServiceCallback">
  <operation name="onResult">
    <input message="tns:LoanServiceResultMessage"/>
  </operation>
</portType>

```

BP1 to P1 – Initiate operation

```

<soap:Envelope>
  <soap:Header>
    <wsa03:ReplyTo>
      <wsa03:Address>ormi://serverB:7001/default/AmericanLoanClient/1.0/
service/LoanServiceRequester
      </wsa03:Address>
    </wsa03:ReplyTo>
    <MessageID>AmericanLoanClient~1.0/60007</MessageID>
  </soap:Header>
  <soap:Body >
    <loanApplication>
      ...
    </loanApplication>
  </soap:Body>
</soap:Envelope>

```

P1/B1 to BP2

```

<soap:Envelope>
  <soap:Header>
    <wsa03:ReplyTo>
      <wsa03:Address>http://serverB:7001/P2</wsa03:Address>
      <wsa03:ReferenceProperties>
        <osb:Callback>
          <osb:Address>
ormi://localhost/default/AmericanLoanClient/1.0/service/LoanServiceRequester
          </osb:Address>
        </osb:Callback>
      </wsa03:ReferenceProperties>
    </wsa03:ReplyTo>
    <MessageID>AmericanLoanClient~1.0/60007</MessageID>
  </soap:Header>
  <soap:Body >
    <loanApplication>
      ...
    </loanApplication>
  </soap:Body>
</soap:Envelope>

```

The ReplyTo callback address is set by B1, which gets the value from the Callback Proxy field in the BPEL transport configuration, as described in [Table 34-2](#). B1's callback proxy is P2.

You must wrap the original replyTo information and send it as reference properties so that it is echoed back in the onResult callback message (to follow).

Note:

This sample uses osb:Callback and osb:Address for illustration purpose only. There is no standard or Service Bus standard elements defined for WS-Addressing support.

BP2 to P2 – onResult operation

```
<soap:Envelope>
  <soap:Header>
    <wsa03:RelatesTo>AmericanLoanClient~1.0/60007</wsa03:RelatesTo>
    <osb:Callback>
      <osb:Address>ormi://localhost/default/AmericanLoanClient/1.0/service/
LoanServiceRequester
      </osb:Address>
    </osb:Callback>
  </soap:Header>
  <soap:Body >
    <loanOffer>
      ...
    </loanOffer>
  </soap:Body>
</soap:Envelope>
```

The reference property osb:Callback is sent back as a SOAP header by the Oracle BPEL Process Manager engine.

P2/B2 to BP1 – onResult operation

```
<soap:Envelope>
  <soap:Header>
    <wsa03:RelatesTo>AmericanLoanClient~1.0/60007</wsa03:RelatesTo>
  </soap:Header>
  <soap:Body >
    <loanOffer>
      ...
    </loanOffer>
  </soap:Body>
</soap:Envelope>
```

Proxy service P2 removes the temporary osb:Callback header; but prior to deleting this header, the replyTo address value is copied to the \$outbound variable so that the BPEL transport in business service B2 can send the callback message to the correct BPEL process.

BPEL Transport Configuration Reference

This section provides descriptions for BPEL transport-specific properties for business and proxy services.

- [BPEL Transport Endpoint URI](#)
- [Configuring Business Services to Use the BPEL Transport](#)

BPEL Transport Endpoint URI

[Table 34-2](#) describes the URI formats for the BPEL transport, which you configure on the main Transport page of the business service in either JDeveloper or the Oracle Service Bus Console.

Table 34-2 Specifying an Endpoint URI

Transport Role	Endpoint URI Format
Synchronous Client or Asynchronous Client role	<p>For the following endpoint URI format, optional elements are shown in square brackets.</p> <pre>protocol://host[:port][/protocol-path]/domain/process[/version[/partnerlink/role]</pre> <p>Following are descriptions of the other endpoint URI elements:</p> <ul style="list-style-type: none"> • protocol: Use one of the following RMI / JNDI protocols. <ul style="list-style-type: none"> iiop / iiops: For generic, server-agnostic use. t3 / t3s: For use with Oracle WebLogic Server. http / https: For tunneling and use with Oracle WebLogic Server. ormi / ormis: For stand-alone deployment on OC4J (Oracle Container for JEE). • port: Optional. For the ormi and opmn protocols only, if the server is configured to use the default port. • protocol-path: Optional. For use only with the opmn and opmns protocols. The <code>protocol-path</code> is the server instance in a cluster. • version: Optional. The version of the process to invoke. • partnerlink/role: Optional. Include this option for a full path description when you specify version. <p>For a cluster, you can also use the following URI format for targeting specific nodes in the cluster:</p> <pre>protocol://host1:port1,host2:port2/<remainder_of_URI></pre>
Service Callback	<p>If the callback address is always known, for example when the client and BPEL service are linked together because of a trading partner agreement, you can provide the exact callback address for the Endpoint URI instead of using the following format:</p> <pre>bpel://callback</pre>

Configuring Business Services to Use the BPEL Transport

[Table 34-3](#) describes the options available on the BPEL transport configuration page of the business service in both JDeveloper and the Oracle Service Bus Console. For more information, see [Creating and Configuring Business Services](#).

Table 34-3 BPEL Transport Properties for Business Services

Property	Description
Role	<p>Select one of the following roles for the business service. The BPEL transport is used to send request messages from Service Bus to Oracle BPEL PM. Each role requires specific configuration. The business service can serve one of the following roles:</p> <ul style="list-style-type: none"> • Synchronous Client: Select this role for synchronous communication with a Service Bus client. The only required location information is the BPEL address. This address is captured statically as the endpoint URI or dynamically through URI rewriting. • Asynchronous Client: Select this role for asynchronous communication with a Service Bus client. In this case, a callback from Oracle BPEL Process Manager to Service Bus is sent on a different connection than the request, and you must configure Service Bus to provide the correct callback address in the Callback Proxy field. For more information, see the guidelines in Advanced Use Cases (Asynchronous). • Service Callback: Select this role if the business service is designed to be a service callback to Oracle BPEL Process Manager (where Oracle BPEL Process Manager is calling an external service asynchronously through Service Bus). The callback address is known only at runtime. Use an Endpoint URI of <code>bpel://callback</code>. <p>If you configure the business service with the marker URI, configure your pipeline logic to dynamically set the URI on <code>\$outbound</code>. For example, you could use the <code>TransportHeader</code> action to do this.</p> <p>Note: A Service Callback business service does not support load balancing or failover.</p> <p>For instructions on using Service Callback, see "Service Callback" in Table 34-2 and Asynchronous: Calling Service Providers from Oracle BPEL Process Manager.</p>
Callback Proxy	<p>For asynchronous clients only, enter the proxy service to use to route callbacks to the Service Bus client that made the request. The proxy service must be either an SB or HTTP proxy service with a service type of Any SOAP. For more information, see the guidelines in Advanced Use Cases (Asynchronous).</p> <p>This field is available only for the Asynchronous Client role.</p>
Service Account	<p>Enter a service account that will be used for JNDI context security to access the Oracle BPEL Process Manager delivery service. If no service account is specified, an anonymous subject is used. There is no restriction on the type of service account that can be configured, such as static or pass-through, but the runtime must be able to access a user name and password.</p> <p>For more information, see Working with Service Accounts.</p>

Table 34-3 (Cont.) BPEL Transport Properties for Business Services

Property	Description
Suspend Transaction	<p>Select this option to make the business service non-transactional even if the business service is invoked by a transaction.</p> <p>If you do not select Suspend Transaction, the following rules apply:</p> <ul style="list-style-type: none"> • If the protocol indicates a protocol supported by WebLogic Server (t3, iiop, http), the transaction is propagated. • If the protocol indicates an OC4J server (ormi, opmn), the BPEL transport throws an exception, since OC4J does not support transaction propagation. <p>The BPEL transport ignores the Suspend Transaction setting in the following situations:</p> <ul style="list-style-type: none"> • The business service is called with quality of service (QoS) "best-effort." The BPEL transport automatically suspends any transaction that does not support QoS. • The business service is called with QoS set to "exactly-once" and there is no transaction. <p>For a description of transaction propagation, see Transaction Propagation in the BPEL Transport.</p>
Dispatch Policy	<p>Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists.</p> <p>For information about Work Managers, see:</p> <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>

Using the SB Transport

This chapter provides an overview of the SB transport and describes how to use and configure it in your services. The SB transports integrates Oracle products with Service Bus using RMI

This chapter includes the following sections:

- [Introduction to the SB Transport](#)
- [SB Transport Error Handling](#)
- [UDDI and the SB Transport](#)
- [SB Transport Configuration Reference](#)

Introduction to the SB Transport

The SB transport allows Oracle products to synchronously invoke an Service Bus proxy service using RMI. The inbound transport allows clients to access SB proxy services using RMI. The outbound transport allows the invocation of SB proxy services using RMI. By default, accessing all services using T3 protocol, IIOP, HTTP, T3s, IIOPS, or HTTPS depends on the configuration of the target server. For more information, see "Configure Default Network Connections" in the *Oracle WebLogic Server Administration Console Online Help*.

SB Transport Features

The SB transport supports the following:

- Propagation of the transaction context. The transaction that originated in the client Service Bus servers can optionally be propagated to the SB proxy service.
- Propagation of the security context. By default, the security context associated with the SB client thread is used to invoke the SB proxy services. This may require enabling domain trust between domains. See "Important Information Regarding Cross-Domain Security Support" in *Administering Security for Oracle WebLogic Server*.
- Invocation of SB proxy services, with custom identities, by the outbound endpoint using a service account.
- Specification of time out value for non-transactional invocations. The client request returns when Service Bus does not respond to the request within the specified interval.
- Association of a dispatch-policy for both request and response connections. For more information, see "Using Work Managers to Optimize Scheduled Work" in *Administering Server Environments for Oracle WebLogic Server*.

- Optimization of RMI call and call-by-reference when routing to a SB business service without a JNDI provider.
- The following service types:
 - WSDL service
 - Any SOAP service
 - Any XML service
- The following messaging patterns:
 - Request (one-way) and request-response for the inbound transport.
For a Service Bus client, the messaging pattern is inherited from the pipeline of the SB outbound transport by default.
For a non-Service Bus client the default messaging pattern is request-response.
 - Request and request-response for the outbound transport environment values.
For more information on the environment values the SB supports, see [SB Transport Environment Values](#).
- The following default values for the Quality of Service (QoS):
 - Exactly-Once for non-Service Bus clients.
 - Best-Effort for Service Bus clients.

You can also set the QoS of a service using routing options in the pipeline. For more information, see [Quality of Service](#).

SB Transport Error Handling

You can configure business service URIs to be taken offline when communication errors occur. When you configure the operational settings for the business service, you can enable the business service endpoint URIs to be taken offline after the specified retry interval. For more information, see "Monitoring and Managing Endpoint URIs for Business Services" in *Administering Oracle Service Bus*.

When a connection error occurs while invoking a SB proxy service, the SB transport provider generates the BEA-380002 error code. A connection error can occur due to any of the following reasons:

- The target proxy service does not exist.
- The JNDI provider settings are incorrect.
- Any remote or naming exception occurs during RMI invocation.
A naming exception of the type `javax.naming.NamingSecurityException` typically occurs when the identity used during the invocation is not recognized by the target server. When this occurs, the request returns a generic runtime error, which is not treated as a connection error.
- An external service returns a SOAP payload with a fault code.

When there is an error handler in place to handle the fault code, the transaction is not marked for rollback. If there is no error handler, the transaction is marked for rollback.

When an error occurs and there is an error handler in place that catches the fault and sends back a fault response (using the `With Error` option for the Reply action), the error is considered an application error and the transaction is not marked for rollback. In all other error cases, the transaction is marked for rollback.

SOAP faults returned by SB proxy servers are treated as application errors.

UDDI and the SB Transport

You can import and publish services to the UDDI registry. For more information, see [Working with UDDI Registries..](#)

- [Publishing a Service](#)
- [Importing a Service](#)

Publishing a Service

When you publish a proxy service to a UDDI registry, the URI associated with the published service has the following format:

```
sb://host:port/service_name
```

where `host:port` refers to the host name and listening port of the server hosting the proxy service that is being published.

If the `Use SSL` option is enabled for the proxy service that is being published, the URI associated with the published service has the following format:

```
sbs://host:port/service_name
```

If the proxy service that is being published is running on a cluster, `host:port` is the `Cluster Address` setting in the Cluster section of the `config.xml` file. This value can either be a single host name and port number that is used to connect to any WebLogic Server in the cluster or it can be a comma-separated list of the host name and listener ports of the Managed Servers in the Service Bus cluster. For more information, see "WebLogic JNDI" in *Developing JNDI Applications for Oracle WebLogic Server*.

For more information, see [Publishing Proxy Services to a UDDI Registry..](#)

Importing a Service

When you import a service from the UDDI registry, the SB transport provider matches the `sbscheme` and `host:port` information from the service URI property with a JNDI provider registered on the Service Bus server using the appropriate protocol based on `sbscheme`. `Sbscheme` is the URI scheme of the SB transport-based service and can be either `sb` or `sbs`.

If `sbscheme` is `sb`, the transport provider looks for the JNDI provider using T3, T3S, IIOP, IIOPS, HTTP, or HTTPS protocol (in this order). If `sbscheme` is `sbs`, the transport provider looks for the JNDI provider using T3S protocol, IIOPS, then HTTPS (in this order). The JNDI provider that matches the service URI property is used to generate the endpoint URI of the business service that is imported to Service Bus.

If there is no matching JNDI provider, the import fails unless the imported URI is a local URI and the scheme is not `sb`, the default context is used. This implies that there is no JNDI provider specified for the service and it is considered co-located with the server.

For example, if the service URI property value is:

```
sbs://remote_oracle_service_bus_host:7002/myService
```

the generated URI of the business service imported to Service Bus would be:

```
sb://my_jndi_provider/myService
```

where, `my_jndi_provider` is a JNDI provider resource registered on the Service Bus server with a `t3s://remote_oracle_service_bus_host:7002` URL.

For more information, see [Working with UDDI Registries](#).

SB Transport Configuration Reference

This section describes the environment values and properties you can configure for business and proxy services using the SB transport. It also describes the JNDI provider and error handling for the SB transport.

- [SB Transport Environment Values](#)
- [Configuring Proxy Services to Use the SB Transport](#)
- [Configuring Business Services to Use the SB Transport](#)

SB Transport Environment Values

[Table 35-1](#) describes the environment values supported by the SB transport. The values you specify for these variables override the properties configured for specific SB-based business or proxy services.

Table 35-1 SB Transport Environment Variables

Environment Value	Use this value to ...
Timeout (category: operational)	Specifies the time out value (in seconds) for the business service.
Service account (category: security)	A Service Bus service account resource to use for JNDI context security to access the Oracle BPEL Process Manager delivery service. Use this to update the user credentials associated with an SB-based business service.
Use SSL (category: security)	Specifies whether to expose the service through a secure protocol for a proxy server. For more information, see Configuring Proxy Services to Use the SB Transport .

Configuring Proxy Services to Use the SB Transport

A Service Bus client connects with the Service Bus server using the JNDI context and the proxy service URI. The security context of the client is used to invoke the proxy service. The default QoS is `Exactly-once`. Optionally, the client can change the QoS, set a request time out value, and specify a desired messaging pattern. The message is received by the inbound SB transport and processed through the pipeline.

[Table 35-2](#) describes the properties you use to configure an SB-based proxy service. Specify the endpoint URL for the endpoint as the proxy service name. For instructions on creating a proxy service, see [Creating and Configuring Proxy Services](#).

Table 35-2 SB Transport Properties for Proxy Services

Property	Description
Dispatch Policy	<p>Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists.</p> <p>For information about Work Managers, see:</p> <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>
Use SSL	<p>Select this option to expose the service through a secure protocol. Although this implies that the client should use the SSL protocol to access the SB proxy service, this does not prevent the client from accessing the service through unsecure protocols. In addition, the endpoint URI associated with the service would be <code>sbs</code> instead of <code>sb</code> for the following:</p> <ul style="list-style-type: none"> • When you export a secure service through UDDI and preserve security and policy configuration details during import. • In the effective WSDL file. <p>Note: A proxy service is not bound to any particular protocol. It is the responsibility of the Oracle WebLogic Server administrator to enable SSL, IIOP, or HTTP tunneling whenever it is necessary.</p> <p>This flag only affects the URI scheme of the service when it is exported or the JNDI provider selection for the business service URI when it is imported from UDDI. It does not prevent a client from accessing the service using a non-secured protocol.</p>

Configuring Business Services to Use the SB Transport

SB business services can send messages only to SB proxy services. A JNDI provider, which is specified in the endpoint URI of the business service, performs a JNDI lookup on the remote Service Bus server. The client user credentials or the user credentials defined in the service account associated with the business service are used to invoke the proxy service. Optionally, a time out value and a custom dispatch policy can be associated with the business service. The QoS of the service can also be set by using the routing options.

Specify the endpoint URI for the service using the following format:

```
sb://jndi_provider_name/service_name
```

Where:

- `jndi_provider_name` is the name of the JNDI provider, which points to the corresponding Service Bus JNDI provider resource. This is optional. When omitted, the default context is used. This implies that the service and the Service Bus server are located on the same machine. When the call is co-located, serialization is skipped during service invocation. For more information, see [JNDI Providers](#).
- `service_name` is a target service and corresponds to the remote proxy service URI.

The following table describes the properties you use to configure an SB-based business service. For more information, see [Creating and Configuring Business Services](#).

Table 35-3 SB Transport Properties for Business Services

Property	Description
Dispatch Policy	<p>Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists.</p> <p>For information about Work Managers, see:</p> <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>
Time out	<p>Enter the duration, in seconds, after which the business service times out and returns a runtime error.</p> <p>The specified time out value is <i>not</i> applied when:</p> <ul style="list-style-type: none"> • QoS of the service endpoint is <code>Exactly-Once</code>. • The specified value is a negative value. • The time out value is overridden in the optional Timeout custom header of the outbound request in the message flow. For information, see Adding Transport Header Actions in the Console.
Service Account	<p>Specify the user credentials that should be used for invoking the remote proxy service. If no service account is specified, the user credentials of the inbound proxy service (the inbound client) of this business service are used for security context propagation. For more information, see Working with Service Accounts.</p>

JNDI Providers

A JNDI provider points to the Service Bus server where the service is deployed to retrieve the RMI stubs corresponding to the SB proxy service. The JNDI provider has a high performance caching mechanism for remote connections and EJB stubs. T3, IIOP, HTTP, T3s, IIOPS, or HTTPS transport protocols can be used by JNDI provider. The preferred communication protocol from Service Bus to an Oracle WebLogic Server domain is T3 or T3S. If messages need to go through a fire wall, you can use HTTP tunneling by using an HTTP provider url in the context and by enabling HTTP tunneling on the Oracle WebLogic Server server.

Note:

It is the responsibility of the administrator to ensure that the protocol supported by the JNDI provider is on the remote Service Bus server.

When you create a business service, you can associate it with a JNDI provider. For more information, see [Working with JNDI Provider Resources](#).

Using the SOA-DIRECT Transport

This chapter provides an overview of the SOA-DIRECT transport and describes how to use and configure it in your services. The SOA-DIRECT transport lets you invoke Oracle SOA Suite service components, such as BPEL processes, human tasks, rules, and Oracle Mediator components.

Note:

The SOA-DIRECT transport is for communicating with Oracle SOA Suite 11g and later service components. Service Bus also provides a `bpel-10g` transport to communicate with Oracle SOA Suite 10g Release 3. For information on that transport, see [Using the Oracle BPEL Process Manager Transport](#).

This chapter includes the following sections:

- [Introduction to the SOA-DIRECT Transport](#)
- [Using SOA Suite Services with Service Bus](#)
- [SOA-DIRECT Transport Configuration Reference](#)
- [WS-Addressing Reference](#)
- [XML Messaging Examples](#)

Introduction to the SOA-DIRECT Transport

The SOA-DIRECT transport provides native connectivity between Service Bus and Oracle SOA Suite service components. Oracle SOA Suite provides a "direct binding" framework that lets you expose Oracle SOA Suite service components in a composite application. The SOA-DIRECT transport interacts with those exposed services through the SOA direct binding framework, letting those service components interact in the service bus layer and leverage the capabilities and features of Service Bus.

For more information on SOA binding components, see "Getting Started with Binding Components" and "Using the Direct Binding Invocation API" in the *Developing SOA Applications with Oracle SOA Suite*.

SOA-DIRECT Transport Features

The SOA-DIRECT transport supports the following features:

- Invocation of any SOA binding component services through Java remote method invocation (RMI)

- WS-Addressing, including optional auto-generation of ReplyTo properties for asynchronous callbacks
- Identity propagation
- Transaction propagation
- Attachments
- Optimized RMI transport for invoking SOA services
- High availability and clustering support
- Failover and load balancing (not available for services in the Service Callback role)
- Connection and application retries on errors

Service Binding Types

The SOA-Direct transport supports WSDL type services with SOAP 1.1, SOAP 1.2, or, alternatively, XML bindings. The SOA direct binding framework only exposes direct binding services as WSDL with SOAP 1.1 and SOAP 1.2 bindings, not XML. However, if you want to use an XML binding, you must manually customize the imported SOA service WSDL files for the direct binding services. An XML binding has no effect on the message payload, since messages between the SOA-DIRECT transport and SOA binding components are always abstract (no binding).

WS-Addressing for the SOA-DIRECT Transport

The SOA-DIRECT transport uses only WS-Addressing for message correlation in synchronous and asynchronous communications. The transport automatically generates the following WS-Addressing properties in the SOAP header when you configure a callback proxy in the business service configuration:

- `ReplyTo`: For setting the callback address and connection information in asynchronous callbacks.
- `ReferenceParameters`: Contains the callback properties for ReplyTo, including JNDI and connection factory properties, for the following supported WS-Addressing versions:
 - `http://www.w3.org/2005/08/addressing`
 - `http://schemas.xmlsoap.org/ws/2004/08/addressing`
- `ReferenceProperties`: Contains the callback properties for ReplyTo, including JNDI and connection factory properties, for the following supported WS-Addressing version: `http://schemas.xmlsoap.org/ws/2003/03/addressing`.

For ReplyTo and ReferenceParameters examples, see [WS-Addressing Reference](#). For all other WS-Addressing properties, you must add or transform them in Service Bus pipelines if they are not available or suitable for pass-through to the SOA-DIRECT business service. If you use correlation and callback mechanisms other than WS-Addressing, you must transform messages in pipelines to support WS-Addressing between Service Bus and SOA framework service components.

For WS-Addressing examples with the SOA-DIRECT transport, see [WS-Addressing Reference](#) and [XML Messaging Examples](#).

SOA-DIRECT Transport Security

The SOA-DIRECT transport supports one-way SSL. To use SSL, enable SSL in the domain, use the secure protocol in the endpoint URI, such as HTTPS, IIOPS, or T3S, and reference the secure port in the URI. For more information on the SOA-DIRECT URI, see [SOA-DIRECT Endpoint URIs](#).

You can provide identity propagation with the SOA-DIRECT transport by passing the caller's subject through the service or with a service account bound to the service. Because the SOA-DIRECT transport deals with only normalized, abstract messages, the transport does not support WS-Security. For more information on security settings, see [Configuring Business Services to Use the SOA-DIRECT Transport](#).

SOA-DIRECT Transport Error Handling

The SOA-DIRECT transport recognizes connection and application errors, letting you configure the appropriate retry settings in the transport configuration. The transport throws generic errors for errors that are neither connection nor application related.

- [Connection Errors](#)
- [Application Errors](#)
- [Generic Errors](#)

Connection Errors

The SOA-DIRECT transport raises connection errors in the following situations:

- The target service does not exist.
- A naming exception occurs during the RMI lookup or invocation (with the exception of `javax.naming.NamingSecurityException`, which is a generic error).
- A remote exception occurs during the RMI lookup or invocation.

Application Errors

The SOA-DIRECT transport raises application errors when the outbound business service receives a SOAP fault. You can deselect Retry Application Errors on the service configuration page to prevent retries on application errors—errors that are likely to keep failing despite retries.

Generic Errors

The SOA-DIRECT transport raises a generic error in the following situations:

- All errors other than connection and application errors.
- A `javax.naming.NamingSecurityException`, which is thrown during the JNDI lookup, is not considered a connection error as are other naming exceptions.

Using SOA Suite Services with Service Bus

This section describes synchronous and asynchronous communication patterns between Service Bus and Oracle SOA Suite composites.

- [Simple Use Cases – Synchronous](#)

- [Advanced Use Cases – Asynchronous](#)

Simple Use Cases – Synchronous

This section describes the simple, most common use cases for communicating natively to and from SOA composites through Service Bus: synchronous communication.

- [Transactional Boundaries](#)
- [Synchronous Invocation of a SOA Composite](#)
- [Synchronous Invocation from a SOA Composite](#)
- [Associating Messages with the Correct Conversation](#)

Transactional Boundaries

When synchronous BPEL components use the direct binding to interact with proxy services, the Service Bus and BPEL components share the same transactional context by participating in the same global transaction. The pipeline can perform any back-end activity within the same transactional context initiated by the BPEL component. In order to guarantee data consistency, everything that was done in that transaction must be rolled back if something fails to maintain state during processing. Note that the direct binding is typically used because transaction or security propagation is needed.

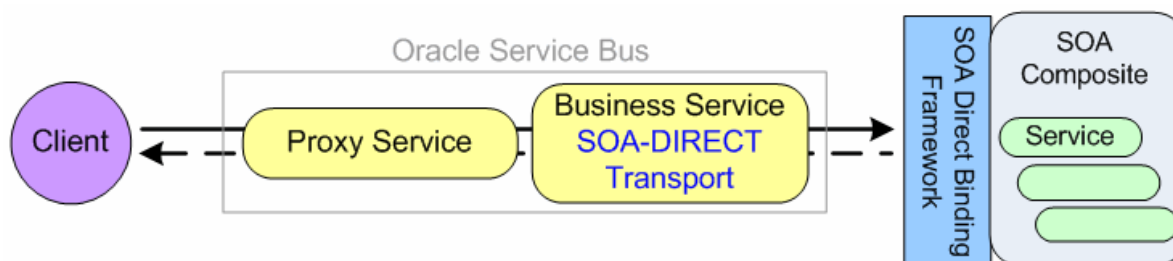
Service Bus direct binding failures are thrown back to the BPEL component as system faults, because Service Bus always marks the transaction for rollback in case of processing failure within the request pipeline. Therefore, any fault thrown from the Service Bus direct binding is a rollback fault and is interpreted as system fault on the SOA Suite side.

For more information about this behavior, see "BPELCaller Calls BPELCallee That Has `bpel.config.transaction` Set to required" in *Developing SOA Applications with Oracle SOA Suite*. This section explains transactional behavior between BPEL components with synchronous interfaces.

Synchronous Invocation of a SOA Composite

The SOA-DIRECT transport can invoke any component in a SOA composite that is exposed as a direct binding service. [Figure 36-1](#) illustrates a synchronous communication pattern between a client and an Oracle SOA service component through Service Bus using a SOA-DIRECT business service and direct binding service.

Figure 36-1 Client Invoking a SOA Binding Service Synchronously



Use the following guidelines to invoke an Oracle SOA direct binding service from a client through Service Bus:

- Create a SOA-DIRECT business service in Service Bus that represents the SOA service component you want to invoke.

- In Service Bus, create a WSDL resource based on the corresponding Oracle SOA direct binding service WSDL file.

You can locate the SOA direct binding service WSDL file in JDeveloper using the SOA Resource Browser, as described in "Developing SOA Composite Applications with Oracle SOA Suite" in the *Developing SOA Applications with Oracle SOA Suite*.

- Create a new business service with a **soa-direct** transport type and a **WSDL** service type.
- Select the WSDL resource you created, and choose the appropriate port or binding.

Note:

If you select the port, the transport type and URI will be automatically propagated in the next configuration page.

- Set the endpoint URI as described in [SOA-DIRECT Endpoint URIs](#).
- Configure the remainder of the business service, described in [Configuring Business Services to Use the SOA-DIRECT Transport](#).
- Create a proxy service in Service Bus that invokes the business service. Choose the transport type that is used by the client. For proxy configuration information, see the online help provided with Service Bus.

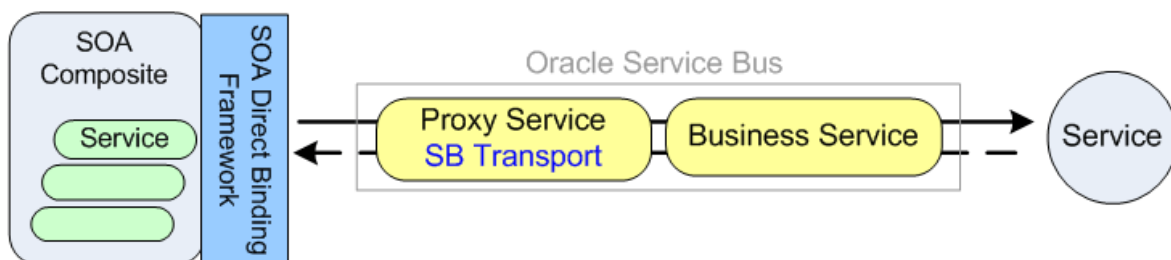
If you are using stateful services to ensure that messages are associated with the correct conversation, see [Associating Messages with the Correct Conversation](#).

Synchronous Invocation from a SOA Composite

A SOA composite can invoke any Service Bus SB WSDL-based proxy service. To invoke an SB proxy service, the SOA service component must use a direct binding reference of target type Oracle Service Bus. For more information on target types, see How to Create an Outbound Direct Binding Reference in *Developing SOA Applications with Oracle SOA Suite*.

Figure 36-2 illustrates a synchronous communication pattern between an Oracle SOA service component and an external service through Service Bus.

Figure 36-2 SOA Binding Service Invoking an External Service Synchronously



Use the following guidelines to invoke an external service from a SOA composite using direct binding references:

- Create a business service in Service Bus that represents the external service you want to invoke. Choose the transport type that is supported by this service. For

business service configuration information, see the online help provided with Service Bus.

- Create an SB proxy service in Service Bus that invokes the business service.
 - Create a WSDL resource to be used by the proxy service that invokes the business service.
 - Create a new proxy service with an **sb** transport type and a **WSDL** service type.
 - Select the WSDL file for the proxy service, and select the desired port or binding.
 - Configure the remainder of the proxy service. For more information, see [Using the SB Transport](#).

Note:

Use the SB proxy service effective WSDL file and port type to define the direct binding reference that invokes Service Bus. You can import this WSDL file into an Oracle SOA Suite project.

If you are using stateful services, ensure that messages are associated with the correct conversation. See [Associating Messages with the Correct Conversation](#).

Associating Messages with the Correct Conversation

When using stateful services, the messages sent synchronously between Service Bus and Oracle SOA composites are known as a conversation. To ensure that messages are correctly associated with each other as part of a conversation, the SOA-DIRECT transport provides built-in support for WS-Addressing.

For more information on WS-Addressing, see [MessageID / RelatesTo Headers](#). For an example of conversation ID setting, see [Conversation ID Examples](#).

Advanced Use Cases – Asynchronous

This section describes asynchronous communications between a SOA composite and Service Bus using the SOA-DIRECT transport.

Note:

Only the following SOA service components currently support asynchronous conversations using WS-Addressing: BPEL Process, Mediator, and Human Task.

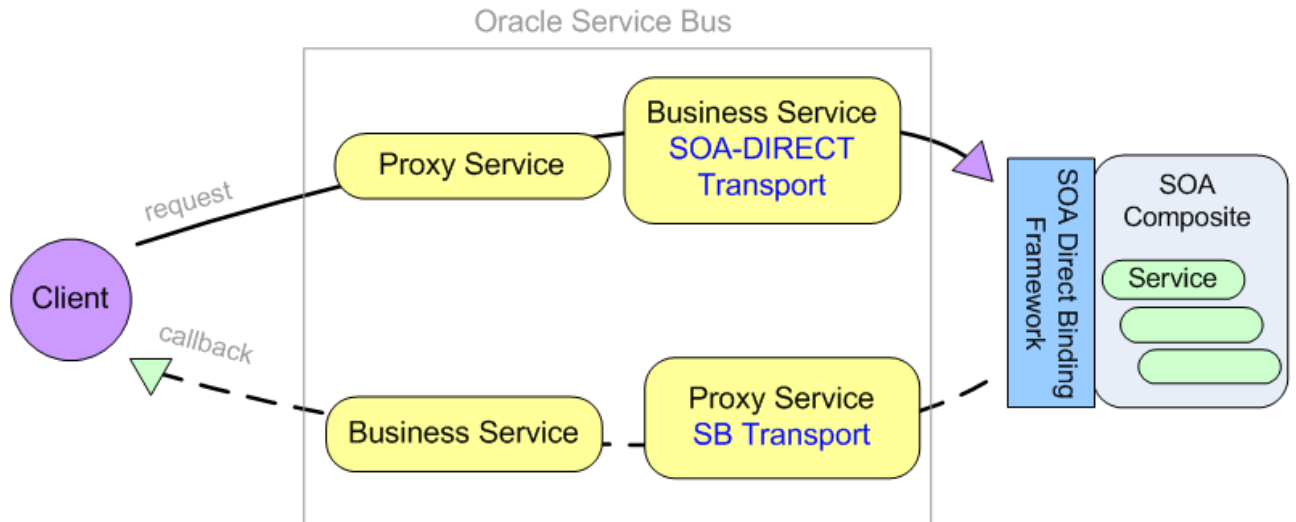
- [Asynchronous Invocation of a SOA Composite](#)
- [Asynchronous Invocation from a SOA Composite](#)

Asynchronous Invocation of a SOA Composite

The SOA-DIRECT transport can invoke asynchronous SOA service components that are exposed as direct binding services. [Figure 36-3](#) illustrates an asynchronous communication pattern between a client and an Oracle SOA composite through

Service Bus using a direct binding service, the SOA-DIRECT transport, and the SB transport.

Figure 36-3 Client Invoking a SOA Binding Service Asynchronously



Use the following guidelines to invoke the SOA direct binding service asynchronously from a client through Service Bus:

- On the inbound client side, create the Service Bus artifacts to interact with the client: a request proxy service that invokes the outbound SOA-DIRECT business service, and a callback business service that handles the callback to the client. Use the transport type used by the client.

– Request Proxy Service

Configure the proxy service that receives the client request. This proxy service invokes the outbound request SOA-DIRECT business service.

Since the callback is sent to a different connection, Service Bus must be able to remember the original callback location when calling back the client. When using WS-Addressing, the callback address is sent to the request proxy service in the ReplyTo address header. Before invoking the SOA-DIRECT business service, the request proxy service can pass this address as a referenceParameter property inside the ReplyTo header. Following the WS-Addressing specification, the referenceParameter property is inserted in the SOAP header block of the callback. The callback SB proxy can then extract this callback address and set the callback business service URI.

For information on setting a callback address, see [ReplyTo Header](#) and [Asynchronous Composite to Composite Communication Through Service Bus](#).

– Callback Business Service

Configure the business service you need to handle the callback. This business service is invoked by the outbound callback SB proxy service.

For service and transport configuration guidance, see the online help provided with Service Bus.

- On the Service Bus outbound side, create the artifacts to interact with the SOA composite. This includes a request SOA-DIRECT business service that makes the request to the Oracle SOA direct binding service exposing the asynchronous

service component you want to invoke, and a callback SB proxy service that handles the callback from the direct binding service and invokes the inbound callback business service.

– **Request SOA-DIRECT Business Service**

- ◆ In Service Bus, create a WSDL resource based on the corresponding Oracle SOA direct binding service WSDL file.

You can locate the SOA direct binding service WSDL file in JDeveloper using the SOA Resource Browser, as described in "Developing SOA Composite Applications with Oracle SOA Suite" in the *Operation*.

- ◆ Create a new business service with a soa-direct transport type and a WSDL service type.
- ◆ For the WSDL file, browse to the WSDL resource you created and select the appropriate port or binding for the direct binding service.

If you select the port, the transport type and URI are automatically propagated in the next configuration page.

- ◆ Set the endpoint URI, described in [SOA-DIRECT Endpoint URIs](#).
- ◆ On the transport configuration page, set the Role to **Asynchronous Client**.
- ◆ Optionally use the **Callback Proxy** option to select the SB callback proxy service you created.

When you select a callback proxy, the SOA-DIRECT transport automatically generates the WS-Addressing headers to tell the SOA direct binding service that it expects the asynchronous callback response to be sent to the specified callback proxy.

For approaches to setting a callback address if you do not select a callback proxy in the SOA-DIRECT business service, see [WS-Addressing Reference](#) and [Asynchronous Composite to Composite Communication Through Service Bus](#).

- ◆ Configure the remainder of the business service. For more information, see [SOA-DIRECT Transport Configuration Reference](#).
- ◆ Invoke this business service from the request proxy service.

• **Callback SB Proxy Service**

- Create a new proxy service with an **sb** transport type and a **WSDL** service type.
- Browse to the WSDL file corresponding to direct binding service's WSDL file, and select the appropriate port or binding.
- Complete the proxy service configuration. For more information, see [Configuring Proxy Services to Use the SB Transport](#).

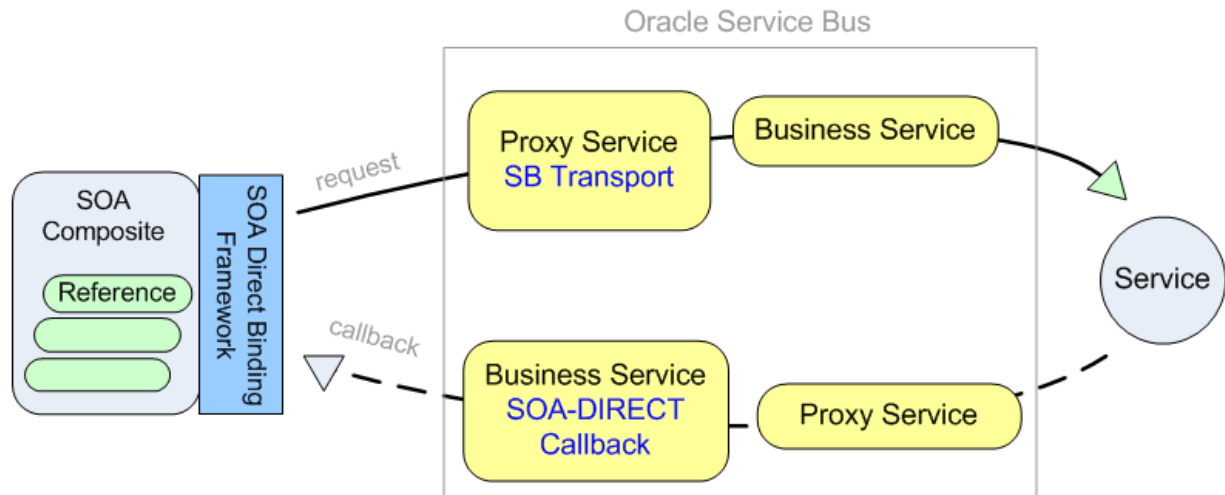
Asynchronous Invocation from a SOA Composite

An asynchronous SOA service component in a SOA composite can invoke external services through Service Bus. To do so, the SOA service component must use a direct binding reference of Target Type of "Oracle Service Bus." (For more information on

target types, see *How to Create an Outbound Direct Binding Reference in Developing SOA Applications with Oracle SOA Suite*.

Figure 36-4 illustrates an asynchronous communication pattern between an Oracle SOA service component and an external service through Service Bus using a direct binding reference, the SB transport, and the SOA-DIRECT transport.

Figure 36-4 SOA Binding Service Invoking an External Service Asynchronously



Use the following guidelines to invoke an external service asynchronously from an Oracle SOA direct binding reference through Service Bus.

- In Service Bus, on the inbound side, create the artifacts to interact with the SOA composite: a request SB proxy service that receives the SOA direct binding reference request and a callback SOA-DIRECT business service that handles the callback to the SOA direct binding reference.
 - **Request SB Proxy Service**
 - ◆ Create a WSDL resource representing the interface used to interact with the direct binding reference.
 - ◆ Create a new proxy service with an **sb** transport type and a **WSDL** service type.
 - ◆ For the WSDL file, browse to the WSDL file you created and select the appropriate port or binding.
 - ◆ Complete proxy service configuration. For more information, see [Configuring Proxy Services to Use the SB Transport](#).

Since the callback is sent to a different connection, Service Bus must be able to remember the original callback location when calling back the client. When using WS-Addressing, the callback address is sent to the request proxy service in the ReplyTo address header. Before invoking the external service, the request proxy service passes this address as a `referenceParameter` property inside the ReplyTo header. Following the WS-Addressing specification, the `referenceParameter` property is inserted in the SOAP header block of the callback. The callback proxy service can then extract this callback address and set the callback business service URI.

For information on setting a callback address, see [ReplyTo Header](#) and [Asynchronous Composite to Composite Communication Through Service Bus](#).

– **Callback Business Service**

- ◆ Create a new business service with a **soa-direct** transport type and a **WSDL** service type.
- ◆ For the WSDL file, browse to the WSDL file representing the callback interface with the direct binding reference, and select the appropriate port or binding.
- ◆ Set the service URI to "callback," as described in [SOA-DIRECT Endpoint URIs](#).

In general, the callback URI is dynamically set in the invoking proxy using URI rewriting. However, if the callback address is always known, you can provide the exact callback address instead of "callback."

- ◆ Set the role to **Service Callback** on the SOA-DIRECT transport configuration page.
 - ◆ Configure the remainder of the business service, as described in [Configuring Business Services to Use the SOA-DIRECT Transport](#).
- On the Service Bus outbound side, create the artifacts to interact with the external service: a request business service that makes the request to the external service and a callback proxy service that handles the callback from this service.

– **Request Business Service**

Configure the business service to invoke the external service. This business service will be invoked by the request SB proxy service. Choose the transport type that is supported by this service. For business service configuration information, see the online help provided with Service Bus.

– **Callback Proxy Service**

Configure the proxy service to pass the callback address to the business service. The callback URI is provided in the request. Use URI rewriting to extract the callback URI and forward it to the SOA-DIRECT business service. Choose the transport type that is supported by this service. For proxy service configuration information, see the online help provided with Service Bus.

For information on setting the callback addresses using WS-Addressing, see [WS-Addressing Reference](#).

SOA-DIRECT Transport Configuration Reference

This section describes the endpoint URL format and configuration options for the SOA-DIRECT transport.

- [SOA-DIRECT Endpoint URIs](#)
- [Configuring Business Services to Use the SOA-DIRECT Transport](#)
- [SOA-DIRECT Transport Environment Values](#)

SOA-DIRECT Endpoint URIs

When specifying the endpoint URI for a SOA-DIRECT server, you need to follow a specific formatting pattern, depending on the type of role the service plays.

For SOA-DIRECT business services in the Service Callback role handling the inbound request, the actual URI is specified dynamically at runtime in the pipeline. Enter the following for the endpoint URI:

```
callback
```

Alternatively, if the callback address is always known, you can provide the exact callback address.

For all other SOA-DIRECT business service roles, use the following format. Optional elements are in brackets [].

```
protocol://authority]/default/compositeName[!versionNumber[*label]]/serviceName
```

where:

- **protocol** is the RMI or JNDI protocol to use. Use one of the following:
 - **iiop / iiops**: For generic, server-agnostic use.
 - **t3 / t3s**: For use with WebLogic Server.
 - **http / https**: For tunneling and use with WebLogic Server.
For HTTP(S) protocols, enable HTTP tunneling on the server. For SSL protocols, enable SSL on the server.

The protocol and authority are optional when the SOA services are co-located on the same server as Service Bus.

- **authority**: The IP address or host name and the port of the SOA server or cluster hosting the SOA service components.

The protocol and authority are optional when the SOA services are co-located on the same server as Service Bus.

- **default**: This domain name value is always "default."
- **compositeName**: The name of the composite application where the binding component service is defined.
- **!versionNumber**: The composite application version number. This is optional. If you do not specify a version, the current version is used.
- ***label**: Used with **!versionNumber**, this is the label hash used in the SOA service WSDL file. This is optional.
- **serviceName**: The name of the SOA binding component service.

While you can specify more than one URI on a service for load balancing and failover, only one URI is allowed for services in the Service Callback role, described in [Table 36-1](#). Therefore, load balancing and failover are not available for services in the Service Callback role.

Endpoint URI Format in a Cluster

When operating in a cluster, the SOA-DIRECT transport uses a different format for the endpoint URI. Use the following format for the endpoint URI in a cluster:

```
t3://example_managed1:port1,example_managed2:port2/service_path
```

where `t3://example_managed1:port1,example_managed2:port2` is the JNDI provider URL.

Endpoint URI Examples

Following are some endpoint URI examples for the SOA-DIRECT transport:

- `t3s://example:7002/default/compositeApp/1.0/myService`
Points to a service deployed on a single node.
- `/default/compositeApp!1.0/myService`
Points to a service co-located on the same server as Service Bus.
- `t3://soaserver.example.com:7001/default/VacationRequest!1.0/directclient`
Points to a service deployed on a single node using a version number. This is the format in SOA binding component service WSDL files.
- `t3://example_managed1:8001,example_managed2:8002/default/myComposite/myService`
Points to a clustered SOA framework environment identified by "myService." Because no specific version is specified, the most recent version of the service is used.

Configuring Business Services to Use the SOA-DIRECT Transport

[Table 36-1](#) describes the properties you use to configure a SOA-DIRECT transport for a business service. For more information, see [Creating and Configuring Business Services](#).

Table 36-1 SOA-DIRECT Transport Properties for Business Services

Property	Description
JNDI Service Account	Enter the <i>static</i> service account that defines security credentials for the JNDI lookup of the target SOA service. If you do not specify a service account, an anonymous subject is used. You can select from a list of defined service accounts. For more information, see Working with Service Accounts .

Table 36-1 (Cont.) SOA-DIRECT Transport Properties for Business Services

Property	Description
Role	<p>Select one of the following options to identify the communication pattern the service uses:</p> <ul style="list-style-type: none"> • Synchronous Client (default): In this role the Callback Proxy option is disabled because there is no asynchronous callback. The WS-Addressing Version option is also disabled. • Asynchronous Client: In this role you can identify a Callback Proxy, and you must select a WS-Addressing Version. Asynchronous callback is usually required. The asynchronous option is enabled only when the WSDL service is of type SOAP. • Service Callback: This role is for returning the asynchronous callback to an SOA service after an external service invocation. <p>There is no load balancing or failover for Callback services.</p>
Callback Proxy	<p>Specify the proxy service that receives callbacks. This option is enabled only for the Asynchronous Client role.</p> <p>When you select a callback proxy, if no WS-Addressing is provided by the request or the proxy service pipeline, Service Bus automatically populates the ReplyTo property in the SOAP header. You must select a WSDL proxy service that uses the SB transport (for RMI), and the callback proxy service must understand WS-Addressing.</p> <p>WS-Addressing properties that are sent in the request or set in the proxy service pipeline are used instead of the callback proxy you set in this option.</p> <p>If you do not specify a Callback Proxy, and the request does not contain ReplyTo properties, you must provide ReplyTo properties in the SOAP header through the proxy service pipeline.</p>
Fault Proxy	<p>This option is not currently supported. You must configure your callback services to handle faults in an asynchronous pattern.</p>
WS-Addressing Version	<p>Specify the default WS-Addressing version to use when no WS-Addressing is provided in the request or the proxy service pipeline. This option is enabled only for the Asynchronous Client role.</p> <p>WS-Addressing properties that are sent in the request or set in the pipeline are used instead of the WS-Addressing version you set in this option. For WS-Addressing version mismatches between environments, perform any necessary transformations in the pipeline. For more information, see Transformation Examples.</p>

Table 36-1 (Cont.) SOA-DIRECT Transport Properties for Business Services

Property	Description
Dispatch Policy	<p>Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists.</p> <p>For information about Work Managers, see:</p> <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>
Pass Caller's Subject	<p>Select this option to have Service Bus pass the authenticated subject from the proxy service when invoking the SOA service. The Invocation Service Account option, an alternative to Pass Caller's Subject, is disabled when you select this option.</p> <p>Note: Make sure that domain trust is enabled between client and target server if they are in different domains. For more information, see "Important Information Regarding Cross-Domain Security Support" in <i>Administering Security for Oracle WebLogic Server</i>.</p>
Invocation Service Account	<p>Specify custom security credentials by selecting a service account for RMI invocation. You can specify any type of service account (pass through, static, or mapping). If you do not specify a service account, an anonymous subject is used. This is an alternative to the Pass caller's subject option.</p> <p>For more information, see Working with Service Accounts.</p>

SOA-DIRECT Transport Environment Values

[Table 36-2](#) describes the environment values supported by the SOA-DIRECT transport. The values you specify for these variables override the properties configured for specific SOA-DIRECT business services.

Table 36-2 SOA-DIRECT Transport Environment Values

Environment Value	Description
JNDI Service Account (security category)	<p>The static service account that defines security credentials for the JNDI lookup of the target SOA service. If you do not specify a service account, an anonymous subject is used.</p> <p>For more information about service accounts, see Working with Service Accounts.</p>
Pass Caller's Subject (security category)	<p>When this is enabled, Service Bus passes the authenticated subject from the proxy service when invoking the SOA service. The Invocation Service Account variable below is an alternative to Pass Caller's Subject.</p>
Invocation Service Account (security category)	<p>The service account for RMI invocation using custom security credentials. You can specify any type of service account (pass through, static, or mapping). If you do not specify a service account, an anonymous subject is used. This is an alternative to Pass Caller's Subject.</p>

Table 36-2 (Cont.) SOA-DIRECT Transport Environment Values

Environment Value	Description
Work Manager (environment category)	<p>The instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists.</p> <p>For information about Work Managers, see:</p> <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>

For information on these values, see [Configuring Business Services to Use the SOA-DIRECT Transport](#).

WS-Addressing Reference

This section describes specific WS-Addressing properties that the SOA-DIRECT transport uses to communicate natively with an Oracle SOA composite. It also describes ways to provide callback addresses in asynchronous communication, as described in [Advanced Use Cases – Asynchronous](#).

See [XML Messaging Examples](#) for WS-Addressing examples.

- [ReplyTo Header](#)
- [MessageID / RelatesTo Headers](#)

ReplyTo Header

In an asynchronous communication, a service callback is sent on a different connection than the request. As a service developer, you must supply the correct callback address in an asynchronous exchange so that the callback is sent to the correct client. When using the SOA-DIRECT transport with WS-Addressing correlation, the callback address is specified in the "ReplyTo" WS-Addressing header.

Calling a SOA Composite Asynchronously

The SOA-DIRECT business service can optionally generate the ReplyTo header. In the business service configuration, if you select a Callback Proxy to handle the callback, the SOA-DIRECT transport sets the correct callback address corresponding to this callback proxy in the ReplyTo header. Note that this header is generated only if the incoming message does not already contain a ReplyTo header.

For more information, see [Asynchronous Invocation of a SOA Composite](#).

Calling Back to a SOA Composite Asynchronously

When calling an external service from an Oracle SOA composite through Service Bus, you must manually set a callback address. To do this, set the callback address as the ReplyTo value in the proxy service that invokes the callback SOA-DIRECT business service.

For more information, see [Asynchronous Invocation from a SOA Composite](#).

MessageID / RelatesTo Headers

MessageID and RelatesTo WS-Addressing headers are used to store the conversation ID in conversations between Service Bus and Oracle SOA service components, ensuring related messages remain in the same conversation.

Unlike ReplyTo, the SOA-DIRECT transport does not provide built-in support for the MessageID or RelatesTo headers. Instead, you must set the correct values for those headers in the pipeline that invokes a SOA-DIRECT business service.

The requirements for using MessageID and RelatesTo headers are slightly different in synchronous and asynchronous conversations, as described below:

- **Synchronous conversation:** The MessageID header value determines the conversation ID in the initial request. Then, for subsequent requests within the same conversation, the conversation ID must be provided in the RelatesTo header.
- **Asynchronous callbacks** - The MessageID header value determines the conversation ID in the initial request. Then, for the callback, the conversation ID must be provided in the RelatesTo header.

For more implementation on establishing a conversation ID to make sure messages participate in the correct conversation, see [Associating Messages with the Correct Conversation](#) and the [Conversation ID Examples](#).

XML Messaging Examples

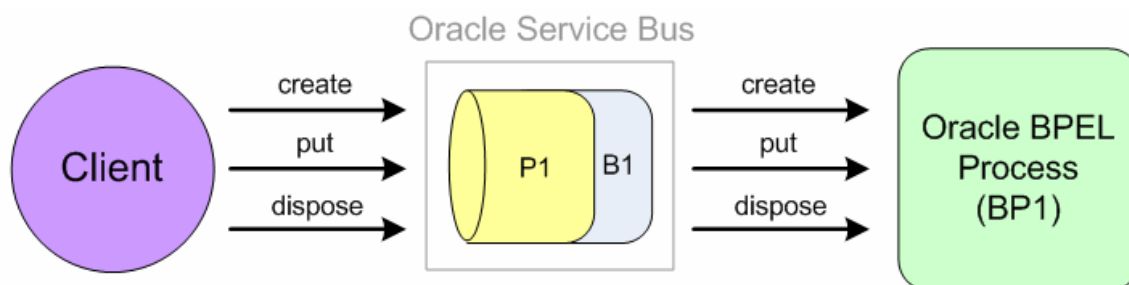
Below are examples of XML messaging between Service Bus and Oracle SOA service Components.

- [Conversation ID Examples](#)
- [Asynchronous Composite to Composite Communication Through Service Bus](#)

Conversation ID Examples

This section provides examples of establishing a conversation ID among messages in a conversation between Service Bus and Oracle SOA composites. In [Figure 36-5](#), a client synchronously invokes a BPEL Process component in an Oracle SOA composite. The business service (B1) uses the SOA-DIRECT transport to invoke a process. The pipeline called by the proxy service (P1) handles any necessary conversation ID mapping. The SOA composite exposes the BPEL Process as a direct binding service.

Figure 36-5 Operations in a Synchronous Exchange Through Service Bus



Port and Message Definitions

The examples in this section use the following port and message definitions defined in the WSDL file.

```
<wsdl:types>
  <xsd:schema
    targetNamespace="http://www.sample.org/spec/samples/types"
    elementFormDefault="qualified">
    <xsd:complexType name="ValueHolder">
      <xsd:all>
        <xsd:any minOccurs="1"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:schema>
</wsdl:types>
<message name="create"/>
<message name="putRequest">
  <part name="key" type="xsd:string"/>
  <part name="value" type="types:ValueHolder"/>
</message>
<message name="putResponse">
  <part name="value" type="types:ValueHolder"/>
</message>
...
<message name="dispose"/>
<portType name="ServiceMap">
  <operation name="create">
    <input message="tns:create"/>
  </operation>
  <operation name="put">
    <input message="tns:putRequest"/>
    <output message="tns:putResponse"/>
  </operation>
  ...
  <operation name="dispose">
    <input message="tns:dispose"/>
  </operation>
</portType>
```

WS-Addressing that Sets the Conversation ID

This example shows how WS-Addressing is used to set the conversation ID among messages in a conversation.

[Figure 36-5](#) shows the communication pattern.

Create Operation

```
<soap:Envelope>
  <soap:Header>
    <wsa03:MessageID>uuid:123456789</wsa03:MessageID>
  </soap:Header>
  <soap:Body>
    <create/>
  </soap:Body>
</soap:Envelope>
```

Put Operation

```
<soap:Envelope>
  <soap:Header>
```

```
<wsa03:MessageID>uuid:111111111</wsa03:MessageID>
  <wsa03:RelatesTo>uuid:123456789</wsa03:RelatesTo>
</soap:Header>
<soap:Body>
  <put>
    <key>key</key>
    <value>
      <PO/>
    </value>
  </put>
</soap:Body>
</soap:Envelope>
<soap:Envelope>
  <soap:Body>
    <putResponse>
      <value/>
    </putResponse>
  </soap:Body>
</soap:Envelope>
```

The <put> operation also has a MessageID, but it is ignored because the RelatesTo has a value that provides the conversation ID.

Message Payload Data that Sets the Conversation ID

This example shows how message payload data can be used to set the conversation ID among messages in a conversation.

In these examples, the proxy service maps the ID to the MessageID / RelatesTo SOAP headers.

[Figure 36-5](#) shows the communication pattern.

Create Operation

Client to proxy service

```
<soap:Envelope>
  <soap:Body>
    <create/>
  </soap:Body>
</soap:Envelope>
<soap:Envelope>
  <soap:Body>
    <createResponse>
      <mapID>uuid:123456789</mapID>
    </createResponse>
  </soap:Body>
</soap:Envelope>
```

Proxy service to SOA composite (using a SOA-DIRECT business service)

```
<soap:Envelope>
  <soap:Header>
    <wsa03:MessageID>uuid:123456789</wsa03:MessageID>
  </soap:Header>
  <soap:Body>
    <create/>
  </soap:Body>
</soap:Envelope>
```

Not shown: The ID was generated in the request of the pipeline and inserted as a <wsa03:MessageID> before invoking the process. On the process side, the Create

operation is one-way, so a SOAP response must be created before replying to the client. The response sends back the ID that was generated by the proxy service.

Put Operation

Client to proxy service

```
<soap:Envelope>
  <soap:Body>
    <put>
      <mapID>uuid:123456789</mapID>
      <key>key</key>
      <value>
        <PO/>
      </value>
    </put>
  </soap:Body>
</soap:Envelope>
<soap:Envelope>
  <soap:Body>
    <putResponse>
      <value/>
    </putResponse>
  </soap:Body>
</soap:Envelope>
```

Proxy service to SOA composite (using a SOA-DIRECT business service)

```
<soap:Envelope>
  <soap:Header>
    <wsa03:RelatesTo>uuid:123456789</wsa03:RelatesTo>
  </soap:Header>
  <soap:Body>
    <put>
      <key>key</key>
      <value>
        <PO/>
      </value>
    </put>
  </soap:Body>
</soap:Envelope>
<soap:Envelope>
  <soap:Body>
    <putResponse>
      <value/>
    </putResponse>
  </soap:Body>
</soap:Envelope>
```

Dispose Operation

Client to proxy service

```
<soap:Envelope>
  <soap:Body>
    <dispose>
      <mapID>uuid:123456789</mapID>
    </dispose>
  </soap:Body>
</soap:Envelope>
```

Proxy service to SOA composite (using a SOA-DIRECT business service)

```
<soap:Envelope>
  <soap:Header>
    <wsa03:RelatesTo>uuid:123456789</wsa03:RelatesTo>
  </soap:Header>
  <soap:Body>
    <dispose/>
  </soap:Body>
</soap:Envelope>
```

Transformation Examples

In these examples, the client uses a more recent version of the WS-Addressing spec (*wsa04* prefix). The proxy service is responsible for transforming the SOAP headers to use the *wsa03* prefix. The proxy service developer configures the transformation.

[Figure 36-5](#) shows communication pattern.

Create Operation

Client to proxy service

```
<soap:Envelope>
  <soap:Header>
    <wsa04:MessageID>uuid:123456789</wsa04:MessageID>
  </soap:Header>
  <soap:Body>
    <create/>
  </soap:Body>
</soap:Envelope>
```

Proxy service to SOA composite (using a SOA-DIRECT business service)

```
<soap:Envelope>
  <soap:Header>
    <wsa03:MessageID>uuid:123456789</wsa03:MessageID>
  </soap:Header>
  <soap:Body>
    <create/>
  </soap:Body>
</soap:Envelope>
```

Put Operation

Client to proxy service

```
<soap:Envelope>
  <soap:Header>
    <wsa04:MessageID>uuid:11111111</wsa04:MessageID>
    <wsa04:RelatesTo>uuid:123456789</wsa04:RelatesTo>
  </soap:Header>
  <soap:Body>
    <put>
      <key>key</key>
      <value>
        <PO/>
      </value>
    </put>
  </soap:Body>
</soap:Envelope>
<soap:Envelope>
  <soap:Body>
    <putResponse>
      <value/>
    </putResponse>
  </soap:Body>
</soap:Envelope>
```

```

    </putResponse>
  </soap:Body>
</soap:Envelope>

```

Proxy service to SOA composite (using a SOA-DIRECT business service)

```

<soap:Envelope>
  <soap:Header>
    <wsa03:MessageID>uuid:111111111</wsa03:MessageID>
    <wsa03:RelatesTo>uuid:123456789</wsa03:RelatesTo>
  </soap:Header>
  <soap:Body>
    <put>
      <key>key</key>
      <value>
        <PO/>
      </value>
    </put>
  </soap:Body>
</soap:Envelope>
<soap:Envelope>
  <soap:Body>
    <putResponse>
      <value/>
    </putResponse>
  </soap:Body>
</soap:Envelope>

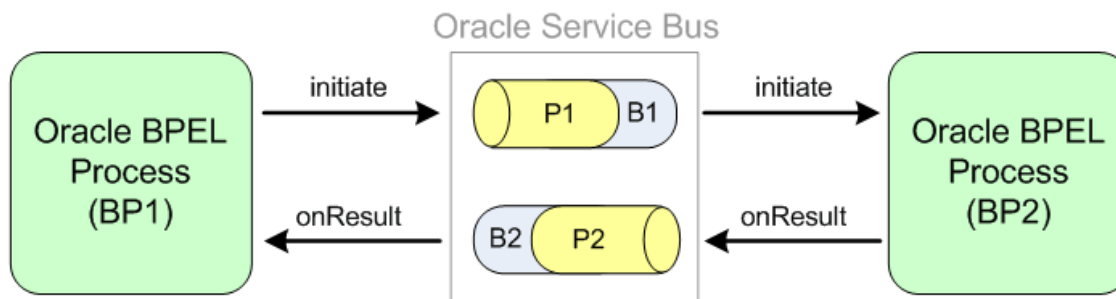
```

Asynchronous Composite to Composite Communication Through Service Bus

The following example shows the SOAP headers involved in a SOA composite invoking another SOA composite asynchronously through Service Bus. The first SOA composite uses a BPEL Process exposed as a direct binding reference to invoke Service Bus. The second SOA composite uses a BPEL process exposed as a direct binding service to receive requests from Service Bus.

In [Figure 36-6](#), P1 and P2 are proxy services with pipelines that pass messages (and perform transformations) to B1 and B2 business services, which are required to make calls to SOA composites using the SOA-DIRECT transport.

Figure 36-6 SOA Composite Invoking an SOA Composite Through Service Bus



Refer to [Figure 36-6](#) for the following SOAP header examples.

Port and Message Definitions

```

<message name="LoanServiceRequestMessage">
  <part name="payload" element="types:loanApplication"/>
</message>
<message name="LoanServiceResultMessage">

```

```

    <part name="payload" element="types:loanOffer" />
</message>
<portType name="LoanService">
  <operation name="initiate">
    <input message="tns:LoanServiceRequestMessage" />
  </operation>
</portType>
<portType name="LoanServiceCallback">
  <operation name="onResult">
    <input message="tns:LoanServiceResultMessage" />
  </operation>
</portType>

```

BP1 to P1 – Initiate operation

```

<soap:Envelope>
  <soap:Header>
    <wsa03:ReplyTo>
      <wsa03:Address>
        t3://soaserver:8001/default/AmericanLoanClient/LoanserviceRequester
      </wsa03:Address>
    </wsa03:ReplyTo>
    <MessageID>AmericanLoanClient~1.0/60007</MessageID>
  </soap:Header>
  <soap:Body >
    <loanApplication>
      ...
    </loanApplication>
  </soap:Body>
</soap:Envelope>

```

P1/B1 to BP2

```

<soap:Envelope>
  <soap:Header>
    <wsa03:ReplyTo>
      <wsa03:Address>http://serverB:7001/P2</wsa03:Address>
      <wsa03:referenceParameters>
        <osb:Callback>
          <osb:Address>
            t3://soaserver:8001/default/AmericanLoanClient/
LoanserviceRequesterRef#LoanserviceRequesterBpel
          </osb:Address>
        </osb:Callback>
      </wsa03:referenceParameters>
    </wsa03:ReplyTo>
    <MessageID>AmericanLoanClient~1.0/60007</MessageID>
  </soap:Header>
  <soap:Body >
    <loanApplication>
      ...
    </loanApplication>
  </soap:Body>
</soap:Envelope>

```

The ReplyTo callback address is set by B1, which gets the value from the Callback Proxy field in the SOA-DIRECT transport configuration, as described in [Configuring Business Services to Use the SOA-DIRECT Transport](#). B1's callback proxy is P2.

You must wrap the original replyTo information and send it as reference properties so that it is echoed back in the onResult callback message (to follow).

Note:

This sample uses `osb:Callback` and `osb:Address` for illustration purpose only. There is no standard or Service Bus standard elements defined for WS-Addressing support.

BP2 to P2 – onResult operation

```
<soap:Envelope>
  <soap:Header>
    <wsa03:RelatesTo>AmericanLoanClient~1.0/60007</wsa03:RelatesTo>
    <osb:Callback>
      <osb:Address>
        t3://soaserver:8001/default/AmericanLoanClient/
LoanserviceRequesterRef#LoanserviceRequesterBpel
      </osb:Address>
    </osb:Callback>
  </soap:Header>
  <soap:Body >
    <loanOffer>
      ...
    </loanOffer>
  </soap:Body>
</soap:Envelope>
```

The reference property `osb:Callback` is sent back as a SOAP header by the Oracle BPEL Process Manager engine.

P2/B2 to BP1 – onResult operation

```
<soap:Envelope>
  <soap:Header>
    <wsa03:RelatesTo>AmericanLoanClient~1.0/60007</wsa03:RelatesTo>
  </soap:Header>
  <soap:Body >
    <loanOffer>
      ...
    </loanOffer>
  </soap:Body>
</soap:Envelope>
```

The pipeline in P2 removes the temporary `osb:Callback` header; but prior to deleting this header, the `replyTo` address value is copied to the `$outbound` variable so the SOA-DIRECT transport in business service B2 can send the callback message to the correct SOA service component.

Using the Tuxedo Transport

This chapter provides an overview of the Tuxedo transport and describes how to use and configure it in your services. The Tuxedo transport lets you bring Tuxedo services into the Service Bus environment.

This chapter includes the following sections:

- [Introduction to the Tuxedo Transport](#)
- [Configuring Oracle Tuxedo Connector](#)
- [Using Tuxedo Services from Service Bus](#)
- [Using Service Bus from Tuxedo](#)
- [Tuxedo Transport Buffer Transformation](#)
- [Tuxedo Transport Transaction Processing](#)

Introduction to the Tuxedo Transport

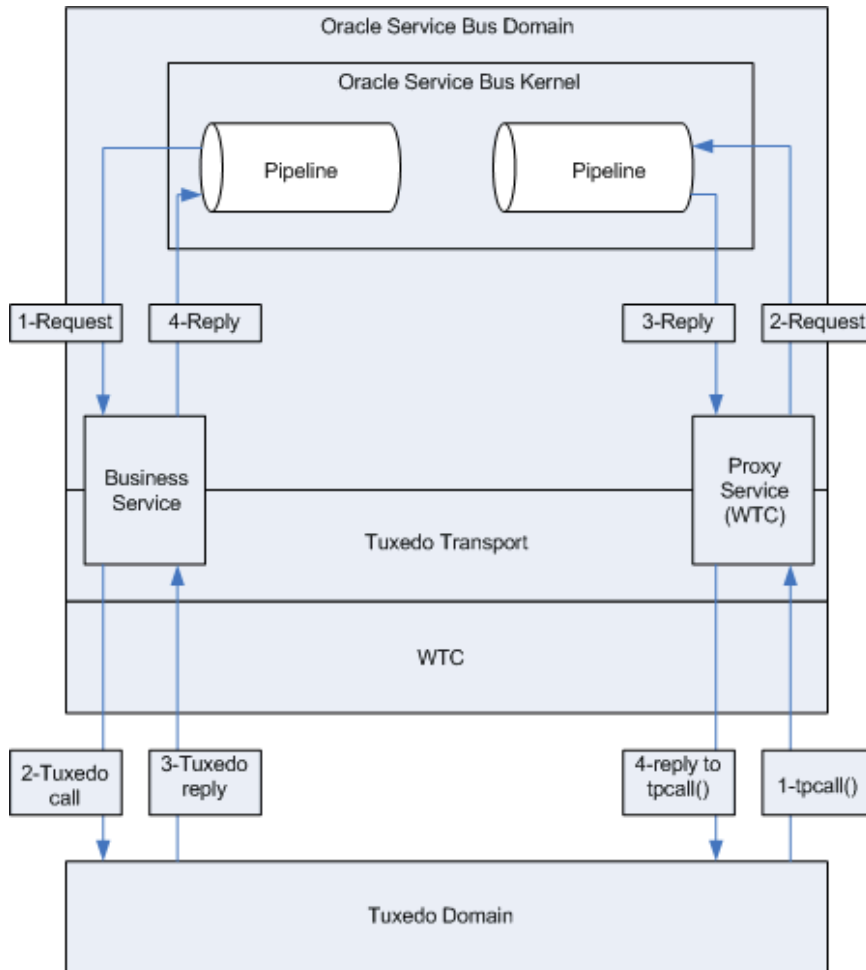
Service Bus and Oracle Tuxedo can work together to use the services that each product offers. The Tuxedo transport provides secure, guaranteed, high performance, bi-directional access to a Tuxedo domain from Service Bus. The Tuxedo transport lets Tuxedo domains call services, as well as have services called, in a Tuxedo domain.

Services can either be outbound or inbound.

- When Service Bus uses services offered by Tuxedo, the Tuxedo transport facilitates access to those Tuxedo services. The term *outbound* refers to this business service scenario.
- When Tuxedo uses services offered by Service Bus, Tuxedo services can call Service Bus services as though they were another Tuxedo application. The term *inbound* refers to this proxy service scenario.

You configure the Tuxedo transport in either JDeveloper or the Oracle Service Bus Console. Specific parameters provide definitions for both proxy and business services. A basic WebLogic Tuxedo Connector (WTC) configuration with one local access point and one remote access point is required to enable configuration of the Tuxedo transport. Support for transactional and security contexts are available as well.

The following diagram summarizes the message handling processes.

Figure 37-1 WTC Message Handling

Capabilities of the Tuxedo Transport

The following capabilities are available in the native Tuxedo transport in Service Bus.

- **First-class tier transport**
The native Tuxedo transport is fully integrated into Service Bus. You can configure, manage, and monitor both Tuxedo proxy services and Tuxedo businesses services.
- **Bi-directional access**
Service Bus is an intermediary between SOAP, JMS, or other services and Tuxedo. The Tuxedo transport provides access to Tuxedo ATMI services as business services in Service Bus and allows Service Bus proxy services to be seen by Tuxedo as another ATMI service.
- **Buffer transformation**
You can transform XML messages to Tuxedo buffer types and Tuxedo buffer types to XML. All standard Tuxedo buffer types are supported; transformation is automatic and transparent. For more information, see [Tuxedo Transport Buffer Transformation](#).
- **Transactional integrity**

The Tuxedo transport provides transactional integrity for inbound and outbound messages. You can call Tuxedo services in the context of a global transaction allowing Exactly Once quality of service (QoS). A Tuxedo application can start a transaction and call a Service Bus service, and the XA transaction context is carried through to Service Bus through the pipeline and finally to the destination transport. For more information, see [Tuxedo Transport Transaction Processing](#).

- Security propagation

The security context established at the beginning of the pipeline, from either a Tuxedo client or a Service Bus client, is propagated to the other system. This means that an incoming SOAP over HTTP request to Service Bus that requires authentication is authenticated by Service Bus. As with transactions, this support is fully bi-directional, so a client authenticated to Tuxedo can make requests to Service Bus services without requiring authentication a second time.

- Encrypted network links

You can encrypt the connections between Service Bus and Tuxedo through WTC configuration to ensure the security and privacy of communications between the two systems.

- Load balancing

A single network connection is the only requirement to connect Service Bus to a Tuxedo domain. However, it might be necessary to support multiple connections in case of a machine or network failure. You can make multiple connections to a single domain or multiple domains for purposes of load balancing.

Configuring Oracle Tuxedo Connector

The Tuxedo transport enables access to Tuxedo services using WebLogic Tuxedo Connector (WTC). To use the Tuxedo transport, you must configure a basic WTC server including one local access point and one remote access point.

The following sections describe how to configure WTC:

- [Before You Begin](#)
- [Configuring Oracle Tuxedo Connector](#)

For information about WTC security, see "How to Configure Oracle WebLogic Tuxedo Connector to Provide Security between Oracle Tuxedo and Oracle WebLogic Server" in the *Administering WebLogic Tuxedo Connector for Oracle WebLogic Server*.

Before You Begin

Gather the following information about the Tuxedo application that Service Bus will use:

- The ID of the Tuxedo local access point.
- The network address of the Tuxedo local access point.
- The name of the exported Tuxedo service.
- Whether the service needs XML-to-FML and FML-to-XML conversion or VIEW-to-XML or XML-to-VIEW conversion.

The example described in the following sections assumes the use of FML/FML32 buffer types.

- The ID of the access point that the Tuxedo domain gateway will use to refer to this Oracle Tuxedo Connector instance. This is referred to as the remote access point ID.)
- The network address that the Tuxedo domain gateway has defined for this Oracle Tuxedo Connector local access point. This is referred to as the remote network address.)

Configuring Oracle Tuxedo Connector

When you create or import Tuxedo business and proxy services in Service Bus, the service configuration includes WebLogic Tuxedo Connector configurations that appear as WebLogic Tuxedo Connector resources in the Oracle WebLogic Server Administration Console. Service Bus needs to keep the WebLogic Tuxedo Connector resources it uses in sync. Modifications to those WebLogic Tuxedo Connector resources in the Oracle WebLogic Server Administration Console can cause those resources to become out of sync with Service Bus, and a re-import of those services into Service Bus results in service activation failure.

Use the following guidelines for using and configuring WebLogic Tuxedo Connector resources in Service Bus:

- Do not modify WebLogic Tuxedo Connector resources in the Oracle WebLogic Server Console that you use in Service Bus proxy and business services. Modify the WebLogic Tuxedo Connector configuration in your Service Bus service configurations.
- If the WebLogic Tuxedo Connector configurations do become out of sync between your Tuxedo services in Service Bus and the Oracle WebLogic Server Console, the easiest way to get back in sync is to delete the WebLogic Tuxedo Connector resources in the Oracle WebLogic Server Console and re-configure or re-import the Tuxedo services in Service Bus.

Using Tuxedo Services from Service Bus

The following sections describe how to use Tuxedo services from Service Bus:

- [Configuring a Tuxedo-Based Business Service](#)
- [Load Balancing and Failover for Tuxedo-Based Business Services](#)
- [Error Handling for Tuxedo-Based Business Services](#)
- [Testing Your Configuration](#)

Configuring a Tuxedo-Based Business Service

To use Tuxedo services from Service Bus, create a new business service that uses the Tuxedo transport in either JDeveloper or the Oracle Service Bus Console. For more information about creating and configuring business services, see [Creating and Configuring Business Services](#). For information about the business service properties specific to the Tuxedo transport, see [Configuring Business Services to Use the Tuxedo Transport](#).

When you create the business service, select **tuxedo** for the transport and select either **Any XML Service** or **Messaging Service** as the service type.

Note:

When editing the Transport tab of a Tuxedo transport business service in the Service Bus console, you may have to save and close the tab to propagate the changes you have made. For example, when adding an endpoint URI, you should save and close the tab, and then reopen the tab, to display the added endpoint URI on the Transport Details tab.

Business Service Endpoint URIs for Tuxedo Transports

When you create a Tuxedo-based business service, you specify the endpoint URIs for the service. Use the following URI format for outbound calls to Tuxedo services:

```
tuxedo:resourcename[/remotename]
```

Where:

- `resourcename` corresponds to a WTC Import service name.
- `remotename` corresponds to the service name exported by the remote Tuxedo domain. This is optional.

Use the following URI format for outbound calls to Tuxedo resources of type /Q:

```
tuxedo-queue:sendQspace/sendQname[/[rcvQspace:]/rcvQname][failureQname]
```

Where:

- `tuxedo-queue` indicates that /Q calls will be made.
- `sendQspace` corresponds to the unique name of the queue space in the Tuxedo domain.
- `sendQname` corresponds to the queue name in the queue space in which requests will be stored.
- `rcvQspace` corresponds to the unique name of the queue space in the Tuxedo domain from which replies will be received. This is optional. If it is not specified, the `sendQspace` value is used.
- `rcvQname` corresponds to the name of the queue in the Tuxedo domain from which replies will be received. This is optional.

Note:

The `rcvQspace` and `rcvQname` properties are optional. If you specify neither value, the runtime returns immediately and does not expect a response. In this case, the **Response Required** option on the Tuxedo Transport Detail page is unavailable.

If you specify either value and do not select the **Response Required** option, the values you specified are ignored.

- `failureQname` corresponds to the name of the queue in the Tuxedo domain where error messages will be stored. This value is also optional. If you specify neither `rcvQspace` nor `rcvQname`, but specify `failureQname`, the URI format is `tuxedo-queue:sendQspace/sendQname//failureQname`.

Note:

When a response is expected, it occurs in the same thread that sends the request.

The Tuxedo transport uses the resource name and remote name from the URI to dynamically create a WTC Import service. If you specify more than one URI, you must have unique resource names for each endpoint. If no remote name is specified, its value is the value of the resource name. If no remote name is entered or if the remote and resource name are the same, only one URI is allowed. This allows already defined WTC Import services to use WTC load balancing and failover.

Note:

If you configure two identical URIs, an error appears notifying you that the service name already exists.

Load Balancing and Failover for Tuxedo-Based Business Services

When specifying a business service and defining the endpoint URIs, you can use the Service Bus load balancing and failover capabilities by entering a remote name that is different from the resource name. In this case, you can define multiple service names and associate them to a service that is replicated across multiple remote domains. The resource name must be unique, but remote names do not have the same restriction.

Error Handling for Tuxedo-Based Business Services

You can configure Tuxedo-based business services to handle application and communication errors as follows:

- **Application Errors:** Specify whether to retry business service endpoint URIs when application errors occur. For more information, see [Business Service Transport Protocol Configuration](#).
- **Communication Errors:** Specify whether business service URIs are taken offline when communication errors occur. For more information, see "Configuring Service Bus to Take Unresponsive Endpoint URIs Offline" in *Administering Oracle Service Bus*.

[Table 37-1](#) describes the following Tuxedo exceptions and the type of Service Bus error they denote.

Table 37-1 Tuxedo Exceptions

Exception	Description
TPSVCFAIL	The service failed—an application error
TPENOENT	The requested entity does not exist—a communication error

Table 37-1 (Cont.) Tuxedo Exceptions

Exception	Description
TPEPERM	A permissions error has occurred—a communication error

Testing Your Configuration

Once you have configured Service Bus to work with Tuxedo, you can test the configuration using the Test Console in the Oracle Service Bus Console.

The following list of tasks summarizes the process of testing outbound usage of Tuxedo by Service Bus.

1. Build and start the Tuxedo servers.
2. Set up a Tuxedo service to call the Service Bus proxy service associated with the business service you just created.
3. In the Oracle Service Bus Console, click **Activate** to enable the Test Console.
4. Open the business service in its editor and click the **Launch Test Console** icon in the upper right corner.
5. Enter a payload in the Test Console. For more information, see [Business Service Testing](#).
6. Click **Execute**.

A response page displays the results of the service request.

Using Service Bus from Tuxedo

The following sections describe how to use Service Bus services from Tuxedo:

- [Configuring a Tuxedo-Based Proxy Service](#)
- [Testing Your Configuration](#)

Configuring a Tuxedo-Based Proxy Service

To use Service Bus services from Tuxedo, configure a new proxy service in JDeveloper or the Oracle Service Bus Console. For more information about creating and configuring proxy services, see [Creating and Configuring Proxy Services](#). For information about the proxy service properties specific to the Tuxedo transport, see [Configuring Proxy Services to Use the Tuxedo Transport](#).

When you create the proxy service, select **tuxedo** for the transport and select either **Any XML Service** or **Messaging Service** as the service type. The endpoint URI is a service name that corresponds to the endpoint URI on the Tuxedo server where the service was deployed.

Testing Your Configuration

Once you have configured Tuxedo to work with Service Bus, you can test the service to verify that it is working correctly. If you are using XML-to-FML32 and FML32-to-XML conversions, test this configuration using the ud32 Tuxedo client program

included with Tuxedo. If you are using FML conversions, you can use the `ud client`. `ud32` reads input consisting of the text representation of FML buffers.

If you are not using XML-to-FML and FML-to-XML conversions, you must develop a test client program in Tuxedo to test this configuration.

Tuxedo Transport Buffer Transformation

Service Bus and Tuxedo can interoperate to use the services that each product offers, which includes buffer transformation. The Service Bus service type and the Tuxedo buffer type determine how transformation occurs.

The Tuxedo transport supports Any XML Service and Messaging Service service types in Service Bus.

- **Any XML Services (Non SOAP):** The messages to XML-based services are XML, but can be of any Tuxedo buffer type allowed by the service configuration.
- **Messaging Services:** Messaging services are those that can receive messages of one data type and respond with messages of a different data type. The supported data types include XML, MFL, text, and untyped binary.

The following sections explain how the Tuxedo transport handles buffer transformation.

- [Buffer Transformation with the Any XML Service Type](#)
- [Buffer Transformation with the Messaging Service Type](#)

Buffer Transformation with the Any XML Service Type

Table 37-2 shows the behavior of the Tuxedo transport depending on the Tuxedo buffer type when the service type is Any XML Service. For Any XML Service, the payload must be a well-formed XML document.

Table 37-2 Buffer Transformation for Any XML Service

Tuxedo Buffer Type	Tuxedo Transport Behavior
STRING	Passes the buffer as is.
CARRAY X_OCTET	Passes the buffer as is.
FML/FML32	Converts the buffer to and from XML using the configured field classes.
XML	Passes the buffer as is. Note: The Tuxedo application <i>should not</i> send NULL bytes when the Tuxedo buffer is XML.
VIEW/VIEW32 X_C_TYPE X_COMMON	Converts the buffer to and from XML using the configured view classes.
MBSTRING	Passes the buffer as is. Note: Encoding is determined by the <code>encoding</code> element of the <code>MetaData</code> of the message sent or received.

Buffer Transformation with the Messaging Service Type

Table 37-3 shows the behavior of the Tuxedo transport depending on the Tuxedo buffer type when the service type is **Messaging Service**. If **None** is specified in the subtype, the Tuxedo transport should not receive any buffer.

Table 37-3 Buffer Transformation for Messaging Service

Tuxedo Buffer Type	Binary Messaging Type	Text Messaging Type	MFL Messaging Type	XML Messaging Type
STRING	Passed as is	Passed as is	Passed as is, assuming the buffer is in a suitable format. If not, the transport returns an error.	XML
CARRAY	Passed as is	Not supported	Passed as is, assuming the buffer is in a suitable format. If not, the transport returns an error.	XML
FML/FML32	Passed as is	Not supported	Not supported	XML
XML	Passed as is	Passed as is	Not supported	XML
VIEW/VIEW32	Passed as is	Not supported	Not supported	XML
MBSTRING	Passed as is	Passed as is	Passed as is, assuming the buffer is in a suitable format. If not, the transport returns an error.	XML

The Tuxedo transport handles the buffer manipulation the same way as if the service was Any XML service type.

Tuxedo Transport Transaction Processing

Service Bus and Tuxedo can interoperate to use the services that each product offers, which often includes transaction processing. Tuxedo transport takes advantage of transactions or starts transactions in Service Bus. The exception to this transaction support is when the inbound transport is Tuxedo with a transactional message and the outbound is request/response XA-JMS. In this case, Service Bus detects this exception and it results in a TPESYSTEM error.

The Tuxedo transport transactional behavior is driven by the Quality of Service (QoS) setting available at the message context level. For more information, see [Quality of Service](#).

The following sections explain how the Tuxedo transport handles transactions.

- [Buffer Transformation with the Any XML Service Type](#)

- [Outbound Tuxedo Service Transaction Processing](#)

Inbound Tuxedo Service Transaction Processing

When a transactional context is received, the message going into the pipeline sets the QoS to **Exactly Once**, otherwise QoS is set to **Best Effort**. When a `TransportException` is caught before the reply is sent back to the client, the request aborts by throwing a `TPESYSTEM` exception and a transaction rollback results.

Outbound Tuxedo Service Transaction Processing

When the thread calling the business service has a transactional context, the Tuxedo transport behaves in the following manner:

- If QoS is set to **Exactly Once**, the Tuxedo transport automatically forwards the transactional context to the remote domain unless the endpoint is configured to suspend the transaction.
- If QoS is set to **Best Effort**, the Tuxedo transport suspends the transaction before making the call and resumes it after the call. This is equivalent to making an ATMI call with `TPNOTRAN` flag set.

When the thread calling the business service has no transaction associated, the Tuxedo call occurs non-transactionally, regardless of the QoS setting. In this case, it will correspond to a `tpcall()` or `tpacall()` with the `TPNOTRAN` flag set.

Tuxedo Transport Configuration Reference

This section lists and describes the properties you can configure when using the Tuxedo transport with proxy and business services.

- [Configuring Proxy Services to Use the Tuxedo Transport](#)
- [Configuring Business Services to Use the Tuxedo Transport](#)

Configuring Proxy Services to Use the Tuxedo Transport

The Transport Detail page of the Proxy Service Definition Editor provides the properties listed in the following table for you to configure the transport.

Table 37-4 Tuxedo Transport Properties for Proxy Services

Option	To create or edit...
Field Table Classes	Enter the name of the class or classes describing the FML/FML32 buffer received. These are used for the FML/FML32-to-XML conversion routines to map field names to element names. This is a space separated list of fully qualified class names.
View Classes	Enter the name of the class or classes describing the VIEW/VIEW32 buffer received or sent. These are used for the VIEW-to-XML or VIEW32-to-XML conversion routines to map field names to element names. This is a list of fully qualified class names separated by spaces. X_C_TYPE and X_COMMON Tuxedo buffer types are handled in the same manner as VIEW/VIEW32 buffers. If an incoming request contains a VIEW, then the corresponding VIEW class should be specified in the Service Bus CLASSPATH.

Table 37-4 (Cont.) Tuxedo Transport Properties for Proxy Services

Option	To create or edit...
Classes Jar	Select a JAR resource that contains a JAR file with the <code>FML/FML32</code> or <code>VIEW/VIEW32</code> classes necessary for this endpoint operation. If you are working in the Oracle Service Bus Console, you must create the JAR resource in the console before you can select it. For more information, see How to Add JAR Files .
Local Access Point	<p>Select a local access point from the list that is associated with the export. The list contains local access points configured in WebLogic Tuxedo Connector (WTC). A proxy service cannot be created if there is not an associated local access point.</p> <p>If no local access points exist or to create a new one, select New. Enter the corresponding Local Access Point Name and Local Network Address in the adjacent fields. Upon validation of the endpoint, the access point is added to the WTC configuration for each WTC server. If no WTC server exists, one is created.</p> <p>You can enter an existing access point name after selecting the New option. This causes the existing information to be updated with the new parameters. You can change only the host name and port number.</p>
Remote Access Point	<p>Select a remote access point from the list to be associated with the newly created local access point. To create a new access point, select New. Enter the corresponding Access Point Name and Network Address in the adjacent fields. This field appears only when you select New in the Local Access Point field.</p> <p>You can enter an existing access point name after selecting the New option. This causes the existing information to be updated with the new parameters. You can change only the host name and port number.</p> <p>The remote access point is also the authentication principal for the WTC connection for inbound requests. Optionally, you can create a user with the same access point ID in the default security realm to allow incoming calls. To do so, select Yes from the Create User? list. The password is randomly generated using a temporary variable to avoid security issues.</p>
Reply Buffer Type	Select the type of buffer that the remote Tuxedo client will receive. This option is available only if the Response Required? field is selected.
Reply Buffer Subtype	Enter the buffer subtype with which to associate the reply buffer. This option is available only when the Response Required? option is selected and the Reply Buffer Type value is VIEW or VIEW32 .
Response Required?	<p>Select this check box if this service is expected to send a response. By default, this option is selected.</p> <p>This option is cleared and the unavailable if the service type is Messaging Service and the response message type is None.</p>
Request Encoding	Specify a character set encoding for requests in Tuxedo transports.
Response Encoding	Specify a character set encoding for responses in Tuxedo transports.

Table 37-4 (Cont.) Tuxedo Transport Properties for Proxy Services

Option	To create or edit...
Transformation Style	<p>Specify the way you want FML/FML32 buffers to be represented in an XML document. Select one of the following:</p> <ul style="list-style-type: none"> • None: The order of fields may not be respected. This is the default selection. • Ordered: The fields are presented with all their occurrences in the correct order. • Ordered and Grouped: If the fields are logically structured as records, the fields are ordered by occurrence and grouped by record.

Configuring Business Services to Use the Tuxedo Transport

The Transport Detail page of the Business Service Definition Editor provides the properties listed in the following table for you to configure the transport.

Table 37-5 Tuxedo Transport Properties for Business Services

Property	Description
Field table Classes	Enter the name of the classes describing the FML/FML32 buffer received. These are used for the FML/FML32-to-XML conversion routines to map field names to element names. This is a space separated list of fully qualified class names.
View Classes	Enter the name of the class or classes describing the VIEW/VIEW32 buffer received or sent. These are used for the VIEW-to-XML or VIEW32-to-XML conversion routines to map field names to element names. This is a space separated list of fully qualified class names.
Classes Jar	<p>Select a JAR Resource that contains a JAR file with the FML/FML32 or VIEW/VIEW32 classes necessary for this endpoint operation.</p> <p>If you are working in the Oracle Service Bus Console, you must create the JAR resource in the console before you can select it. For more information, see How to Add JAR Files.</p>
Remote Access Point(s)	<p>Select a remote access point from the list of available options. The list contains remote access points configured in WTC. A business service cannot be created if there is no associated remote access point.</p> <p>If no remote access points exist or to create a new one, select New. Enter the corresponding Access Point Name and Network Address in the adjacent fields. Upon validation of the endpoint, the access point is added to the WTC configuration for each WTC server. If no WTC server exists, one is created.</p> <p>You can enter an existing access point name after selecting the New option. This causes the existing information to be updated with the new parameters. You can change only the host name and port number.</p> <p>If more than one URI is specified, there is one remote access point field per URI and the URI displays for informative purposes. If more than one URI exists, each requires a different remote access point. If the URI specified already corresponds to an existing WTC resource, the corresponding remote access point displays, but cannot be modified.</p>

Table 37-5 (Cont.) Tuxedo Transport Properties for Business Services

Property	Description
Local Access Point(s)	<p>Select a local access point to be associated with the newly created remote access point. To create a new access point, select New. Enter the corresponding Local Access Point Name and Local Network Address in the adjacent fields.</p> <p>This field appears only when you select New in the Remote Access Point field.</p> <p>Note: Access points are not deleted by the transport when the endpoints are removed, since they may be used by multiple endpoints. To remove access points, use the Oracle WebLogic Server Administration Console.</p>
Request Buffer Type	Select the type of buffer that the remote Tuxedo service will receive.
Request Buffer Subtype	Enter the buffer subtype with which to associate the request buffer. This option is enabled if the previous Request Buffer Type value is VIEW or VIEW32.
Response Required?	Select this check box to indicate a bidirectional call. If this is not selected, the underlying <code>tpcall</code> is invoked with <code>TPNOREPLY</code> flag, and a null response is posted asynchronously.
Suspend Transaction?	<p>Select this check box to suspend the transaction, if it exists. This is useful when the remote service does not support transactions.</p> <p>When making calls to Tuxedo resources of the type <code>/Q</code>, use the Suspend Transaction option whether or not you expect a reply. A successful return from a one-way call means that a message has been successfully queued.</p> <p>Note: Tuxedo transports to <code>/Q</code> mode endpoints are considered asynchronous transactional if the Suspend Transaction option is not selected. This prevents deadlocks. In <code>/Q</code> mode, when an endpoint expects a reply, multiple threads on multiple Managed Servers may reply using the same destination. Therefore, when a reply is expected, a unique correlation ID is sent along with the request. The dequeue operation then waits for the message containing that correlation ID. Correlation IDs are composed in the same manner as those used by JMS transports in similar situations.</p>
Request Encoding	Specify a character set encoding for requests.
Response Encoding	Specify a character set encoding for responses.
Timeout	<p>Specify the maximum amount of time (in seconds) that the transport runtime waits for replies. This must be an integer greater than or equal to 0. If this is not specified or is set to zero (default), replies will time out at <code>BLOCKTIME</code>, the maximum number of seconds that the local Tuxedo access point allows for a blocking call. At runtime, replies exceeding the timeout value are ignored and an error message with a <code>TPETIME</code> exception is returned.</p> <p>This field is only available for request/response services, and not for <code>m/Q</code> or one-way endpoints. If the outbound call is part of a transaction, the timeout value is ignored.</p> <p>Note: The <code>WTC BLOCKTIME</code> value takes precedence if it is less than the timeout value.</p>

Table 37-5 (Cont.) Tuxedo Transport Properties for Business Services

Property	Description
Transformation Style	<p>Select one of the following methods of ordering or grouping elements when FML or FML32 buffers are transformed into XML:</p> <ul style="list-style-type: none">• None: (default) The order of fields may not be respected.• Ordered: The fields are presented with all their occurrences in the correct order.• Ordered and Grouped: If the fields are logically structured as records, the fields are ordered by occurrence and grouped by record.
Dispatch Policy	<p>Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists. Service Bus uses this Work Manager to asynchronously post a null reply in the case of a one-way invocation.</p> <p>For information about Work Managers, see:</p> <ul style="list-style-type: none">• Using Work Managers with Service Bus• "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>

Using the WS Transport

This chapter provides an overview of the WS transport and describes how to use and configure it in your services. The WS transport makes Web Services Reliable Messaging (WSRM) available in Service Bus.

This chapter includes the following sections:

- [Introduction to the WS Transport](#)
- [Authentication and Authorization of Services](#)
- [Using the WS Transport](#)
- [WS Transport Configuration Reference](#)

Introduction to the WS Transport

The WS transport implements both inbound and outbound requests for services derived from SOAP 1.1 and SOAP 1.2 based WSDL documents with Web Services Reliable Messaging (WSRM) policy. However, the WSRM policy can be a part of the WSDL file or can be attached to the service. In addition, security policies can also be declared in the WSDL file or can be associated with a WSDL-based service. When you configure WSDL-based services with WSRM policies in Service Bus, you must choose the WS transport for the service. Service Bus checks for the WSRM policy when you save the service configuration and throws a validation error if WSRM policies are not declared for the WSDL file associated with the service.

Web Services Reliable Messaging

The Web Services Reliable Messaging is also known as WS-Reliable Messaging or just WSRM. The specification describes a protocol that allows messages to be delivered reliably between distributed applications even if a software, system, or network failure occurs. WS-ReliableMessaging is a specification co-developed by IBM, Oracle, Microsoft and TIBCO Systems. This specification is not the same as WS-Reliability (WSR), which is a competing specification developed by OASIS.

Service Bus supports the specification submitted in February 2005. For more information about the specification, see Web Services Reliable Messaging Protocol (WS-ReliableMessaging) at <http://schemas.xmlsoap.org/ws/2005/02/rm/>.

WS Transport Features

Below are the key features of the WS transport:

- One-way and request/response message patterns. For more information, see [Messaging Patterns](#).

- Exactly-once transfer between WS transport and other transports (JMS, SB, and Tuxedo transports) that support XA transactions.
- HTTPS with basic authentication, and with client certificate authentication (two-way SSL) but without client authentication,. For more information, see [Authentication and Authorization of Services](#) .
- Retaining WSRM security configuration while importing resources. For more information, see [Importing and Exporting Resources](#).
- Assignment of transport-level access control policy to a WS proxy service. Only an administrator can assign this policy. For more information, see [How To Configure Transport-Level Access Policies](#).
- WS-Addressing specification submitted in August 2004. For more information, see Web Services Addressing (WS-Addressing) at <http://www.w3.org/Submission/ws-addressing/>.
- WS-I Basic Profile compliance. For more information, see [Web Services Interoperability](#).
- Quality of Service (QoS) in Service Bus for WS proxy service is always set to Exactly Once. For more information, see [Quality of Service](#).

You can set the QoS in the RM policy file using the `<beapolicy:QOS>` element. This element has one attribute, `QOS`, which can take any of the following values:

- `AtMostOnce`
- `AtLeastOnce`
- `ExactlyOnce`
- `InOrder`

Note:

QoS for the WS transport is different from QoS for Service Bus.

- You can associate only SOAP 1.1 and SOAP 1.2 based WSDL files with WSRM policy with a proxy or business service. For more information, see [Configuring Proxy Services to Use the WS Transport](#) and [Configuring Business Services to Use the WS Transport](#) .

Messaging Patterns

WSRM supports both one-way and request/response messaging patterns. The WS transport does not support reliable response. While the request is always reliable, the response is not sent reliably.

For business services, sending a request to an external web service is asynchronous. Successful invocation implies that the message is given to the RM layer successfully and it will be delivered reliably. However, successful invocation does not mean that the message is sent to the endpoint and has successfully invoked the web service.

For the request/response messaging pattern, the response is received from the external web service for a request. In this case, the request and response paths have two different transactions and run in two different threads. The response pipeline is

executed evenly for one-way messaging message pattern. For the one-way pattern, response pipeline invocation means that the message reliably reached the target destination and the web service invocation is complete.

WS-Policies in the WS Transport

A proxy service or business service that uses the WS transport must have a WS-Policy with RM assertions. This also implies that services that use any other transport must not have any WS-Policy with RM assertions. WS-Policy with RM assertions and WSSP v1.2 transport-level security assertions are supported for the WS transport.

However, WSSP v1.2 message-level security assertions and 9.X Oracle proprietary security assertions are not supported. RM assertions should only be bound at the service level and not at the operation or operation request/response levels.

Note:

You must use only one RM assertion for a WS-Policy.

WS-Policy Configurations

WS-Policies can be configured in any one of the following two ways:

- WS-Policy configuration is specified as part of the WSDL file associated with the service. The policies specified in the WSDL file may be included in the WSDL file or referred in the WSDL file.
- WS-Policy is assigned to the service when configuring the service.

Note:

You can use only one of these methods to associate a security policy with the service. If you configure a policy directly in the Service Bus service, any policies defined in the WSDL file are ignored.

Streaming Content for Large Messages

The WS transport does not have streaming support for large messages because the underlying infrastructure (WLS JAX-RPC stack) uses a fully materialized payload. However, when you configure a proxy service for large message processing, the message is fully materialized into a Java object by the WS transport using the streaming optimization in Service Bus. During the proxy service configuration, you can specify if you want to stream content for large message processing by buffering content either in memory or to disk. For more information, see [Streaming Body Content](#).

Web Services Interoperability

The WS transport supports web services interoperability through WS-I Basic Profile. Currently, Service Bus proxy services do not follow all the WS-I Basic Profile restrictions. However, any services configured to use this transport strictly follow the WS-I Basic Profile specification. WS proxy services do not have a WS-I Compliance check in the service configuration and always follow WS-I Basic Profile. This is valid for SOAP1.1 WSDL bindings as WS-I Basic Profile applies only to SOAP 1.1.

Authentication and Authorization of Services

This section provides information about how WS proxy and business services are authenticated and authorized.

- [Proxy Service Authentication](#)
- [Proxy Service Authorization](#)
- [Business Service Authentication](#)

Proxy Service Authentication

WS proxy services support both basic and client certificate (two-way SSL) authentication. When basic authentication is specified in the WS-Policy, all HTTP requests, including RM protocol messages to the WS proxy service must have a valid user name and password.

Proxy service authentication is supported as follows:

- Outbound client certificate authentication using SSL key-pair assigned to the service key provider referenced by the proxy service.
- User name and password identity propagation through a WS proxy service (with basic authentication) to any other outbound transport, or outbound WSS user name token.
- Credential mapping between WS proxy service (with basic or two-way SSL authentication) and any other transport.
- Sending asynchronous responses from WS proxy service to a RM client through HTTP or HTTPS. The default protocol used by proxy and business services is HTTP.
- Asynchronous responses from a WS proxy service to an RM client connect to the `AcksTo` or `ReplyTo` endpoint references specified by the RM client. The RM client can use either HTTP or HTTPS URL. If the RM client uses HTTPS, the RM client can request a client certificate during the SSL handshake. The WS transport uses the SSL key-pair of the service key provider upon request.

Proxy Service Authorization

Administrators can assign a transport-level access control policy to a WS proxy service. As with all transports, this policy is enforced after the inbound transport provider passes the request message to the Service Bus binding layer before invoking the request pipeline. For more information, see [How To Configure Transport-Level Access Policies](#).

Business Service Authentication

WS business services support basic authentication and client certificate authentication. Outbound basic authentication is supported by means of a service account. User name and password identity propagation and credential mapping are provided by the service account. However, a static account can also be used for authentication. The service account can be static, pass-through, or mapped. Pass-through authentication allows passing a user name and password from the client request to the back-end RM

service. Mapped service accounts allow credential mapping. Static service accounts are used when fixed credentials are required.

WS business services also support SSL client certificate authentication (two-way SSL). The key-pair (private key and certificate) used for outbound two-way SSL is not configured on the WS business service, but on the service key provider referenced by the proxy service.

Routing a single message to a WS business service may involve multiple HTTP/S requests from the Service Bus server and back-end service. All such messages are subject to the authentication method configured in the WS business service. In other words, if the service is configured for basic authentication, all messages sent from Service Bus include the HTTP Authorization header with the user name and password, and, if the message is configured for client certificate authentication, Service Bus uses the key-pair to authenticate all messages.

Using the WS Transport

The WS transport reliably delivers messages in a distributed network. The WSRM functionality is available as a transport only for SOAP 1.1 and SOAP 1.2 based WSDL files with WSRM policy. Ensure that the services are associated with a SOAP 1.1 or 1.2 WSDL files with RM-policy or that an RM-policy is attached to the services. You can configure the WS-Policy in a WSDL file or assign it to a service. For more information, see [Configuring WS Policies](#).

Prior to configuring proxy and business services to use the WS transport, ensure that the required WSDL files are available in your Service Bus domain. For more information, see [Importing the WSDL Document into the Oracle Service Bus Console](#), [Configuring Proxy Services to Use the WS Transport](#), and [Configuring Business Services to Use the WS Transport](#).

You can optionally configure an error queue for services so Service Bus delivers failed messages into the queue. This can be a distributed queue. This queue is not created automatically, so you must create it prior to configuring the services. For more information, see [Configuring an Error Queue](#).

Importing the WSDL Document into the Oracle Service Bus Console

In order to create a service based on a WSDL file with WSRM policies in the Oracle Service Bus Console, you need to import the WSDL file or create the WSDL resource and the content of the file in an editor. For more information, see [Working with WSDL Documents in the Oracle Service Bus Console](#) and [Importing and Exporting Resources and Configurations](#).

Configuring WS Policies

The WS transport can be used only with SOAP WSDL files that have a WSRM policy. You can configure a WS-Policy in a WSDL file or assign a WS-Policy to a service in the JDeveloper or the Oracle Service Bus Console. For more information, see [WS-Policies in the WS Transport](#).

When no RM police assertions are specified for the WSDL file associated with a service, a validation message appears when you activate the session. To resolve this conflict, you need to update the WSDL file or attach the policy to the service. For more information, see [Attaching WS Policies to a Service](#) and [Securing Oracle Service Bus Proxy and Business Services with WS-Policy](#).

Attaching WS Policies to a Service

You can attach policies to a proxy or business service on the service's definition editor in either JDeveloper or the Oracle Service Bus Console. When you attach a WS-Policy to a service, any policies defined in the WSDL file associated with the service are ignored. For information about attaching policies, see [Securing Business and Proxy Services](#).

Configuring an Error Queue

By default, undelivered messages are discarded after the specified number of retries. To save these messages, you can configure error queues for business services. Service Bus delivers any messages that fail in the pipeline into these queues. For errors, you must configure a JMS queue. Oracle recommends that you configure a error queue locally instead of a remote queue.

For business services, when a response timeout occurs, the response pipeline is invoked with an error. If the sequence expiration interval is reached, the message is placed in an error queue configured for the business service and the response pipeline is invoked with an error. However, if the response timeout has already occurred, the message is placed in the error queue, but the response pipeline is not invoked.

Note:

For both one-way and request-response services, putting failed messages in the error queue is only a best effort.

Routing the WS Transport Through an HTTP Proxy Server

When an HTTP proxy server is configured, WS business services send outbound messages using the HTTP proxy server. For information about specifying the HTTP proxy server details in your client application, see "Using a Proxy Server When Invoking a Web Service" in "Invoking Web Services" in *Developing JAX-RPC Web Services for Oracle WebLogic Server*.

WS Transport Error Handling

You can configure WS transport-based business services to handle application errors by specifying whether or not to retry business service endpoint URIs when application errors occur. An application error occurs when a WS-based business service receives a SOAP fault as a response and the BEA-380001 or OSB-380001 error code is generated.

When a response timeout or sequence timeout occurs for a request to a business service, the Service Bus server tries to send the message to the next URI based on the load balancing algorithm. This behavior does not depend on the `Retry Application Errors` option.

Importing and Exporting Resources

When a resource exists in a Service Bus domain, you can preserve the security and policy configuration details while importing that resource to Service Bus by selecting the `Preserve Security and Policy Configuration` option. When you select this option, the values in the existing resource are preserved when you import them, even if the security and policy configurations have been updated in the resource.

For information about importing resources, see [Importing and Exporting Resources and Configurations](#).

Importing and Publishing Services Using UDDI Registries

When a proxy service is published to an UDDI registry, the service is converted into WS business service with the WSDL file. If present, the authentication configuration is also exported to UDDI.

When a WSDL-based business service with WSRM policy is imported from an UDDI registry to Service Bus, the service is imported as a WS business service that is automatically configured to use the WS transport. For more information, see [WS-Policies in the WS Transport](#).

For more information, see [Working with UDDI Registries](#).

WS Transport Configuration Reference

This section describes the endpoint URL format and configuration options for the WS transport.

- [Endpoint URIs for the WS Transport](#)
- [Configuring Business Services to Use the WS Transport](#)
- [Configuring Proxy Services to Use the WS Transport](#)

Endpoint URIs for the WS Transport

Endpoint configuration for a proxy service that uses the WS transport is similar to that of HTTP proxy service configuration. Specify the URI in the following format making sure that the context path is unique for proxy services that use either HTTP or the WS transport.

```
/contextPath
```

Endpoint configuration for a business service using the WS transport is also similar to that for HTTP. Specify the URI in one of the following formats:

```
http://host:port number/name  
https://host:port number/name
```

Configuring Business Services to Use the WS Transport

Business services using the WS transport must be associated with WS-Policy with RM assertions. For more information, see [WS-Policies in the WS Transport](#). A business service acts as a client for invoking an external reliable web service. It sends a request to the service and the response is received by an application deployed by Service Bus, which invokes the response path.

The following table describes the properties you use to configure a WS-based business service. For more information, see [Creating and Configuring Business Services](#).

Table 38-1 WS Transport Properties for Business Services

Property	Description
Response Timeout	<p>Enter the number of seconds to wait for a response before timing out. Leaving this field blank indicates that there is no response timeout. The business service will wait for the duration of the sequence timeout configured in the RM policy. If the response does not come in the defined interval after sending a request, response pipeline is invoked with an error saying that service is timed out.</p> <p>If you enter a zero (0) value, there is no timeout, and the service will never time out.</p>
Service Account	<p>Specify the service account that defines the credentials to use when there is an HTTP basic authentication policy on this service.</p> <p>For more information, see Working with Service Accounts.</p> <p>Note: This is only applicable if the WS business service has a WS-Policy that requires basic authentication</p>
Queue Error Messages	<p>Select this check box to enable sending error messages to the configured error queue.</p>
Error Queue URI	<p>Enter the URI of JMS queue for storing error messages using the following format:</p> <pre>jms://host:port/connFactoryJndiName/queueJndiName</pre> <p>This option is available only when the Queue Error Messages check box is selected.</p> <p>Note: While WebLogic Server allows forward slashes in JNDI names, such as "myqueues/myqueue", JNDI names with forward slashes interfere with the URI format required by Service Bus, and you cannot use those names. To work around this issue, define a JMS foreign server and reference that foreign server in the URI. For more information, see "Configure Foreign Servers" in the <i>Oracle WebLogic Server Administration Console Online Help</i>.</p>
JMS Error Queue Service Account	<p>Specify the service account that defines the credentials to use for JNDI lookups and sending error messages to the error queue. This option is available only when the Queue Error Messages check box is selected.</p> <p>For more information, see "Working with Service Accounts" in <i>Developing Services with Oracle Service Bus</i>.</p>
Use SSL for Error Queue	<p>Select the check box to use SSL for connecting to the error queue.</p> <p>This option is available only when the Queue Error Messages check box is selected.</p>
Send Error Message as Binary	<p>Select this option to send error messages as binary messages instead of <code>soapInvokeState</code> objects.</p>

For more information about configuring business services using the WS transport, the online help provided with Service Bus.

Configuring Proxy Services to Use the WS Transport

Proxy services using the WS transport must be associated with WS-Policy with RM assertions. For more information, see [WS-Policies in the WS Transport](#).

A proxy service receives the requests from clients and passes it to the pipeline after the processing related to WSRM is done. The proxy service could also send the response back to the client after receiving it from the response pipeline. A proxy service using the WS transport can be invoked from any other proxy service and it follows the same behavior as when it is invoked by an external client. When an HTTP proxy server is configured, WS proxy services send asynchronous messages using the HTTP proxy server.

Proxy services based on WSDL with SOAP 1.2 binding support SOAP 1.2 messages only and throw a fault with version mismatch error for SOAP 1.1 messages. Similarly, proxy services based on WSDL with SOAP 1.1 binding support SOAP 1.1 messages only and throw a fault with version mismatch error for SOAP 1.2 messages.

The following table describes the properties you use to configure a WS-based proxy service. For more information, see [Creating and Configuring Proxy Services](#).

Table 38-2 WS Transport Properties for Proxy Services

Property	Description
Dispatch Policy	<p>Select the instance of WebLogic Server Work Manager that you want to use for the dispatch policy for this endpoint. The default Work Manager is used if no other Work Manager exists.</p> <p>For information about Work Managers, see:</p> <ul style="list-style-type: none"> • Using Work Managers with Service Bus • "Using Work Managers to Optimize Scheduled Work" in <i>Administering Server Environments for Oracle WebLogic Server</i>
Retry Count	<p>Specify the number of times the WSRM layer tries to deliver a message to the Service Bus message flow. The default is 3.</p> <p>If an unhandled exception occurs in the request flow of a proxy, the incoming WS Transport message is redelivered to the message flow up to the number of times specified by this value. This is important for reliably processing the WS transport messages.</p> <p>Note: When the message delivery fails, the current transaction is rolled back, but the message is not removed from the queue. The server tries to send the message until the message is successfully delivered or the retry limit is reached. When the retry limit is reached, that message is removed from the queue or moved to an error queue. The error queue can be a distributed queue and can be created from the Oracle WebLogic Server Administration Console. For more information, see Configuring an Error Queue.</p>
Retry Delay	<p>Specify the duration in seconds that the server should wait before retrying to deliver the message. The default is 5 seconds.</p>

For more information about configuring proxy services using the WS transport, see the online help provided with Service Bus.

Part VI

Creating Custom Transport Providers

This part describes the Service Bus Transport SDK, which is for experienced Java developers who want to design, create, and deploy a new custom transport provider in Service Bus. The Transport SDK documentation assumes you have solid knowledge of web services technologies, Service Bus, the transport protocol that you want to use with Service Bus, and WebLogic Server.

This part contains the following chapters:

- [Learning About Custom Transport Providers](#)
- [Developing Custom Transport Providers](#)
- [Developing Custom Transport Providers for JDeveloper](#)
- [Packaging and Deploying a Custom Transport Provider](#)
- [Creating a Sample Socket Transport Provider](#)

For additional information about the custom transport provider API and processing, see the following appendixes:

- [Transport SDK Interfaces and Classes](#)
- [Transport SDK UML Sequence Diagrams](#)

Learning About Custom Transport Providers

This chapter describes the concepts, functionality, and design considerations for developing a custom transport provider for use with Service Bus services. Careful planning of development activities can greatly reduce the time and effort you spend developing a custom transport provider.

This chapter includes the following sections:

- [Introduction to Transport Providers](#)
- [Introduction to the Transport SDK](#)
- [Determining Whether to Develop a Custom Transport Provider](#)
- [Transport Provider Components](#)
- [The Transaction Model](#)
- [Transport SDK Security Model](#)
- [Transport SDK and the Threading Model](#)
- [Designing for Message Content](#)

Introduction to Transport Providers

A transport provider implements the interfaces of the Transport Software Development Kit (SDK) and provides a bridge between Service Bus and mechanisms by which messages are sent or received. Such mechanisms can include specific transport protocols, such as HTTP, as well as other entities, such as a file or an email message. A transport provider manages the life cycle and runtime behavior of transport endpoints, which are resource where messages originate or are targeted.

A client sends a message to Service Bus using a specific transport protocol. A transport provider processes the inbound message, handling communication with the service client endpoint and acting as the entry point for messages into Service Bus. The binding layer packs and unpacks messages, handles message security, and hands messages off to the pipeline. For an illustration of the basic flow of messages through Service Bus, see [Figure 1-3](#).

Tip:

For more information about Service Bus message brokering and the role of the transport layer, see [Learning About Oracle Service Bus](#). For more detailed sequence diagrams that describe the message flow through Service Bus, see [Transport SDK UML Sequence Diagrams](#).

By default, Service Bus includes transport providers that support several commonly used transport protocols, such as HTTP, JMS, File, FTP, and others. These native providers let you configure proxy and business services that require these common transport protocols.

Tip:

For information about using and configuring native transport providers, see [Working with JCA Adapters, Transports, and Bindings](#) .

Introduction to the Transport SDK

Service Bus processes messages independently of how they flow into or out of the system. The Transport SDK provides a layer of abstraction between Service Bus and components that deal with the flow of data in and out of Service Bus. This layer of abstraction allows you to design and develop new transport providers to handle unique transport protocols.

The SDK abstracts the following from the rest of Service Bus:

- Handling specific transport bindings.
- Deploying service endpoints on the transport bindings. An endpoint is either capable of transmitting or receiving a message.
- Collecting monitoring information.
- Managing endpoints (such as performing suspend and resume operations and setting connection properties).
- Enforcing Service Level Agreement (SLA) behavior (such as timing out connections).

Transport SDK Features

This section describes the primary features of the Transport SDK.

- [Handling Inbound and Outbound Messages](#)
- [Deploying Transport-Related Artifacts](#)
- [Processing Messages Asynchronously](#)

Handling Inbound and Outbound Messages

A transport provider developed with the Transport SDK handles inbound and outbound messages as follows:

- Inbound messages typically come into Service Bus from an outside source, such as an HTTP client. The Transport SDK packages the payload and transport level headers, if any, into a generic data structure. The Transport SDK then passes the message, in its generic format, to the Service Bus pipeline.
- Outbound messages originate from Service Bus business services and go to an externally managed endpoint, such as a web service or JMS queue. The Transport SDK receives a generic data structure from the Service Bus pipeline, converts it to the corresponding transport-specific headers and payload, and sends it out to an external system.

The Transport SDK handles outbound and inbound messages independently. An inbound message can be bound to one transport protocol and bound to a different transport protocol on the outbound endpoint.

Deploying Transport-Related Artifacts

Certain transports include artifacts that need to be deployed to Oracle WebLogic Server. For instance, a JMS proxy service is implemented as a message-driven bean. This artifact, an EAR file, must be deployed when the new JMS proxy service is registered. Similarly, the EJB transport provider employs an EAR file that must be deployed when a new EJB business service is registered. Other kinds of artifacts might require deployment, such as a JMS transport, which might create queues and topics as part of the service registration. The SDK allows you to support these artifacts and lets you participate in the Oracle WebLogic Server deployment cycle. If the deployment of one of these artifacts fails, the Service Bus session is notified and the deployment is canceled. This feature of the SDK allows for the atomic creation of services. If one part fails, the session reverts to its previous state.

Note:

To participate in Oracle WebLogic Server deployment cycle, the transport provider must implement the `TransportWLSArtifactDeployer` interface. The primary benefit of this technique is atomic Oracle WebLogic Server deployment, which can be rolled back if needed. For more information on this interface, see [Summary of General Interfaces](#), and [When to Implement TransportWLSArtifactDeployer](#).

Processing Messages Asynchronously

The server has a limited number of threads to work with when processing messages, so asynchrony is important. This feature allows Service Bus to scale to handle large numbers of messages. After a request is processed, the thread is released. When the business service receives a response (or is finished with the request if it is a one-way message), it notifies Service Bus asynchronously through a callback.

For additional information, see [Support for Synchronous Transactions](#) and [Transport SDK and the Threading Model](#).

Transport Provider Modes

With the Transport SDK, you can implement inbound property modes and outbound property modes. These connection and endpoint modes are specified in the transport provider's XML schema definition (XSD) file. For more information about this file, see [Step 3. Create an XML Schema File for Transport-Specific Artifacts](#). This schema is available to the Service Bus pipeline for filtering and routing purposes.

Related Features

This section lists related features that are provided by the transport manager. The transport manager provides the main point of centralization for managing different transport providers, endpoint registration, control, processing of inbound and outbound messages, and other functions. These features do not require specific support by a transport provider.

- [Load Balancing](#)

- [Monitoring and Metrics](#)

Load Balancing

The Transport SDK supports load balancing and failover for outbound messages. The following load balancing options are supported:

- **None:** For each outbound request, the transport provider cycles through the URIs in the list in which they were entered and attempts to send a message to each URI until a successful send is completed.
- **Round Robin:** Similar to None, but in this case, the transport provider keeps track of the last URI that was tried. Each time a message is sent, the provider starts from the last position in the list.
- **Random:** The transport provider tries random URIs from the list in which they were entered.
- **Weighted Random:** Each URI is associated with a weight. An algorithm is used to pick a URI based on this weight.

Monitoring and Metrics

The transport manager handles monitoring metrics such as response-time, message-count, error-count, failover-count, throttling-time, and cache-hit-count.

Determining Whether to Develop a Custom Transport Provider

This section explains the basic use cases for writing a custom transport provider. In some cases, it is appropriate to choose an alternative approach.

- [When to Use the Transport SDK](#)
- [When Alternative Approaches are Recommended](#)

When to Use the Transport SDK

One of the prime use cases for the Transport SDK is to support a specialized transport that you already employ for communication between your internal applications. Such a transport may have its own concept of setup handshake, header fields, metadata, or transport-level security. Using the Transport SDK, you can create a transport implementation for Service Bus that allows you to configure individual endpoints, which can be inbound, outbound or both. With a custom transport implementation, you can map the metadata and header fields of the specialized transport to context variables available in a proxy service pipeline.

Use the Transport SDK when the transport provider needs to be seamlessly integrated into all aspects of Service Bus for reliability, security, performance, management, user interface, and the use of the UDDI registry. Some cases where it is appropriate to use the Transport SDK to develop a custom transport include the following:

- Using a proprietary transport that requires custom interfaces and supports an organization's existing applications.
- Using a CORBA or IIOP protocol for communicating with CORBA applications.
- Using other legacy systems, such as IMS and Mainframe.
- Using variations on existing transports.

- Using industry-specific transports, such as LLP, AS3, and ACCORD.

Alternatively, you can use the Transport SDK to support a specialized protocol over one of the existing transports provided with Service Bus. Examples of this could include supporting any of the following:

- Messages consisting of parsed or binary XML over HTTP.
- WS-RM or other new web service standards over HTTP.
- Request-response messaging over JMS, but with a different response pattern than either of the two patterns supported by the Service Bus JMS transport (for example, a response queue defined in the message context).

When Alternative Approaches are Recommended

Creating a new Service Bus transport provider using the Transport SDK can be a significant effort. The Transport SDK provides a rich, full featured environment so a custom transport can have all of the usefulness and capabilities of the transports that come natively with Service Bus. But such richness brings with it some complexity. For certain cases, you might want to consider easier alternatives.

If you need an extension merely to support a different format message sent or received over an existing protocol, it may be possible to use the existing transport and use a Java Callout to convert the message. For example, suppose you have a specialized binary format (such as ASN.1 or a serialized Java object) being sent over the standard JMS protocol. In this case, you might consider defining the service using the standard JMS transport with the service type being a messaging service with binary input/output messages. Then, if the contents of the message are needed in the pipeline, a Java Callout action can be used to convert the message to or from XML. For information on using Java Callouts, see [Using Java Callouts and POJOs](#).

Below are some additional cases where it is best not to use the Transport SDK to develop a custom transport provider:

- When combining existing Oracle solutions with Service Bus satisfies the transport requirement; for example, Oracle WebLogic Server, Oracle WebLogic Integration, Oracle Data Service Integrator, Oracle Business Process Management, Oracle Tuxedo, and Oracle WebLogic Portal.
- When service enablement tools provide a simpler and more standards-based mechanism to implement SOA practices.
- When alternative connectivity solutions (certified with Service Bus) also address the requirement; for example: iWay adapters and Cyclone B2B.
- When EJBs can be used instead as a means to abstract some type of simple Java functionality.

Transport Provider Components

In general, a custom transport provider consists of a design-time part and a runtime part. The design-time part is concerned with registering endpoints with the transport provider. This configuration behavior is provided by the implementation of the UI interfaces. The runtime part implements the mechanism of sending and receiving messages.

When you develop a new custom transport provider, you need to implement a number of interfaces provided by the SDK. This section includes UML diagrams that model the organization of the design-time and runtime parts of the SDK.

Tip:

In Service Bus, implementations of the `TransportProvider` interface represent the central point for management of transport protocol-specific configuration and runtime properties. A single instance of a `TransportProvider` object exists for every supported protocol. For example, there are single instances of HTTP transport provider, JMS transport provider, and others.

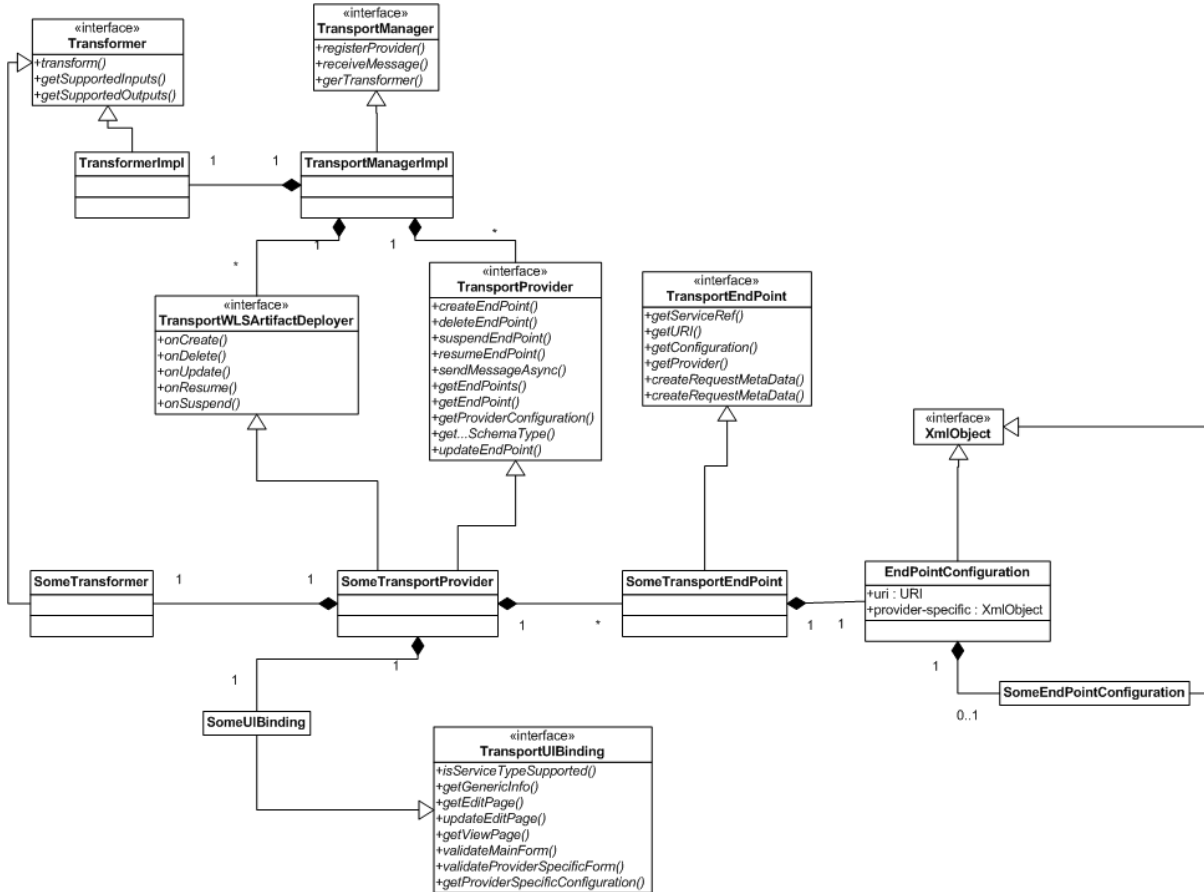
For a list of the required interfaces, see [Developing Custom Transport Providers. Transport SDK Interfaces and Classes](#). provides a summary of the interfaces and classes provided by the Transport SDK. The *Java API Reference for Oracle Service Bus* provides detailed descriptions.

Design-Time Component

The design-time part of a custom transport provider consists of the user interface configuration. This configuration is called by the Oracle Service Bus Console or IDE when a new business or proxy service is being registered. [Figure 39-1](#) shows a UML diagram that depicts the structure of the design time part of a transport provider. Some of the interfaces described in the diagram include:

- **TransportManager:** A transport provider communicates with the transport manager through this interface. The implementation is not public.
- **TransportProvider:** Third parties must implement this interface. The `TransportProvider` keeps track of `TransportEndpoint` objects and also manages the life cycle of the endpoints. For example, you can suspend a transport endpoint managed through the `TransportProvider` interface.
- **TransportUIBinding:** This interface helps the Oracle Service Bus Console render the transport specific pages.

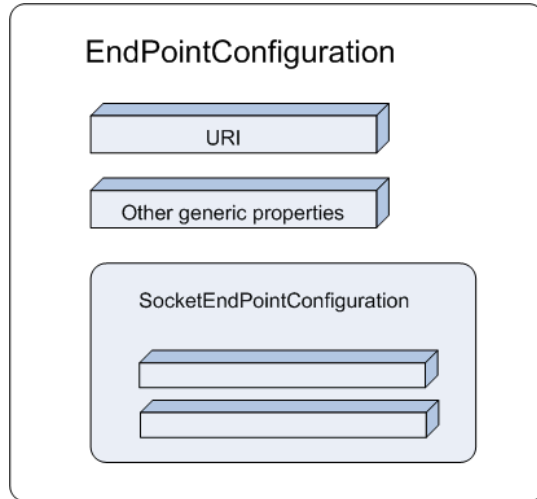
Figure 39-1 Design Time UML Diagram



Note:

Each transport endpoint has a configuration that consists of some properties that are generic to all endpoints of any transport provider, such as a URI, and some properties that are specific to endpoints of that provider only. [Figure 39-2](#) shows the relationship between the shared endpoint configuration properties and transport provider specific configuration properties. For more information, see [Overview of Transport Endpoint Properties](#).

Figure 39-2 EndPointConfiguration Properties



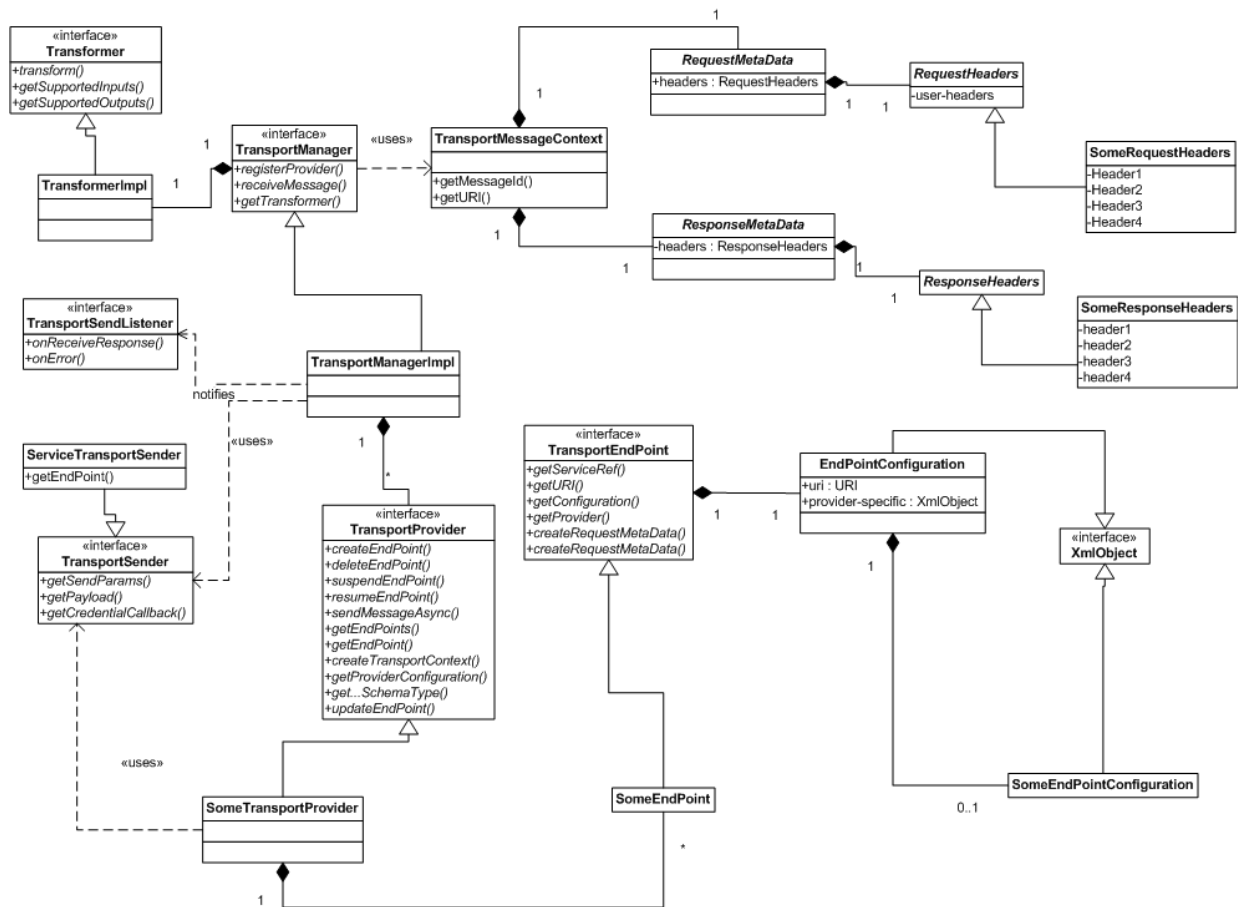
Runtime Component

The runtime part of a custom transport provider receives messages and delivers them to the Service Bus runtime. It also delivers outbound messages from the Service Bus runtime to external services.

In the runtime framework, the transport provider calls the transport manager to acknowledge that an inbound message has been received. The transport message context contains the header and body of the inbound message. For the outbound message, there is a `TransportSendListener` and `TransportSender`. The transport provider retrieves the header and body from the message.

[Figure 39-1](#) shows a UML diagram that depicts the structure of the runtime part of a transport provider.

Figure 39-3 Runtime UML Diagram



The Transaction Model

Before you develop a new transport provider using the Transport SDK, it is important to consider the transaction model for your message endpoints. This section discusses the transaction model used by Service Bus and how that model relates to the Transport SDK.

- [Overview of Transport Endpoint Properties](#)
- [Support for Synchronous Transactions](#)

Overview of Transport Endpoint Properties

A transport endpoint is a Service Bus resource, such as a JMS proxy service, where messages are originated or targeted. In Service Bus, transport endpoints are managed by protocol-specific transport providers, plug-in objects that manage the life cycle and runtime behavior of transport endpoints.

To understand the transactional model of Service Bus, it is useful to review some of the properties of service transport endpoints.

Transactional vs. Non-Transactional Endpoints

A given endpoint may or may not be transactional. A transactional endpoint has potential to start or enlist in a global transaction context when processing a message.

The following examples illustrate how transactional properties vary depending on the endpoint:

- A JMS proxy service that uses the XA connection factory is a transactional endpoint. When the message is received, the container ensures that a transaction is started so the message is processed in the context of a transaction.
- A Tuxedo proxy service may or may not be a transactional endpoint. A Tuxedo proxy service is only transactional if a transaction was started by the Tuxedo client application before the message is received.
- While an HTTP proxy service will not typically have an associated transaction when invoked by an HTTP client, you can set an option in the HTTP proxy service configuration that starts a transaction and executes the message flow in the context of that transaction.

Supported Message Patterns

A given endpoint can use one of the following message patterns:

- **One Way:** No responses are expected. An example of a one-way endpoint is a JMS proxy service that does not expect a response.
- **Synchronous:** A request or response is implied. In this case, the response message is paired with the request message implicitly because no other traffic can occur on the transport channel from the time the request is issued until the time the response is received. In most cases, a synchronous message implies blocking calls for outbound requests. An EJB endpoint is synchronous. An HTTP endpoint is also synchronous: a new request cannot be sent until a response is received.
- **Asynchronous:** A request and response is implied. The response is correlated to a request through a transport-specific mechanism, such as a JMS transport and correlation through a `JMSCorrelationID` message property. For example, a JMS business service endpoint with request and response is asynchronous.

Support for Synchronous Transactions

All Service Bus proxy services support transaction propagation, can start a transaction if none already exists, and can optionally ensure that the response occurs in the context of the transaction, even if the outbound business service is asynchronous. In essence this transforms an asynchronous pattern effectively into a synchronous pattern. Outbound business services can provide additional transaction support, such as suspending an existing transaction.

Synchronous transactional transports support the following use cases:

- [Use Case 1 \(Response Pipeline Processing\)](#)
- [Use Case 2 \(Service Callout Processing\)](#)
- [Use Case 3 \(Suspending Transactions\)](#)
- [Use Case 4 \(Multiple URIs\)](#)

Use Case 1 (Response Pipeline Processing)

Response pipeline processing is included in an incoming transaction when the inbound transport supports synchronous transactions or when you configure a proxy service to propagate a transaction to the response. Service Bus supports this case when

the inbound transport is paired with any other outbound transport, with one exception, as described in the following paragraph.

A deadlock situation occurs when the inbound transport is synchronous transactional and the outbound transport is asynchronous transactional. The deadlock occurs because the outbound request is not available to be received by the business service until after the transaction commits, but the transaction was started externally and does not commit until Service Bus gets the response and returns. The transport manager recognizes this situation and avoids the deadlock by throwing a runtime error. For example, if a synchronous transactional inbound endpoint is used, such as a Tuxedo proxy service, and the outbound endpoint is asynchronous transactional, such as a JMS business service, the outbound request does not commit the transaction until the response is received. It cannot be received until the external entity receives the request and processes it.

Also in this case, the Publish action performed in the response pipeline is part of the transaction just like publish actions in the request pipeline are part of the transaction.

Note:

There are several actions that can potentially participate in a transaction (in either the request or response pipeline). These include Publish, Service Callout, and Report actions.

For example, if an inbound Tuxedo transport is synchronous transactional, it can be committed only after the request and response pipeline have been completed. In this case, the transport manager transfers the transaction context from the inbound to the outbound thread. When the response thread is finished, the transaction control and outcome are returned to the invoking client.

Use Case 2 (Service Callout Processing)

Service Callout pipeline actions allow you to make a callout from the pipeline to another service. If a Service Callout action is made to a synchronous transactional transport, *Exactly Once* and *Best Effort* quality of service are supported. *Exactly Once* means that messages are delivered from inbound to outbound exactly once, assuming a terminating error does not occur before the outbound message send is initiated. *Best Effort* means that each dispatch defines its own transactional context (if the transport is transactional). When *Best Effort* is specified, there is no reliable messaging and no elimination of duplicate messages; however, performance is optimized. For more information, see [Working with TransportOptions](#).

Callouts to synchronous transactional transports are optionally part of an existing transaction. For example, while the request pipeline is executing during a global transaction, Service Callouts are permitted to participate in the transaction. For example, if there is a callout to an EJB service, the service can participate in that transaction if it wants to by setting its quality of service value to *Exactly Once*.

For more information on Service Callouts, see [Adding Service Callout Actions in the Console](#).

Use Case 3 (Suspending Transactions)

Before calling the transport provider to send an outbound request, the transport framework will suspend a transaction if the following conditions apply:

- The outbound service endpoint is transactional.

- There is a global XA transaction in progress.
- The quality of service is set to *Best Effort*.

The suspended transaction resumes after the "send" operation is complete.

Use Case 4 (Multiple URIs)

If a given outbound service endpoint has multiple URIs associated with it, and is transactional, failover only occurs while the transaction, if any, is not marked for rollback. For example, if a URI is called, and the service returns an error, a failover is normally triggered. In this event, the transport framework detects that the transaction has been marked for rollback; therefore, the framework does not perform a failover to a different URI.

Transport SDK Security Model

The Transport SDK allows customers and third-parties to plug in new transports to Service Bus. Within the Service Bus security model, transport providers are considered trusted code. It is critical that transport provider implementations are carefully designed to avoid potential security threats by creating security holes. Although this document does not contain specific guidelines on how to develop secure transport providers, this section discusses certain security goals of the Transport SDK.

Inbound Request Authentication

Transport providers are free to implement whatever inbound authentication mechanisms are appropriate to that transport. For example: the HTTP transport provider supports these authentication methods:

- HTTP basic authentication
- Custom authentication tokens carried in HTTP headers

The HTTPS transport provider supports SSL client authentication, in addition to the ones listed above. Both HTTP and HTTPS transport providers also support anonymous client requests.

The transport provider is responsible for implementing any applicable transport level authentication schemes. If the transport provider authenticates the client it must make the client Subject object available to Service Bus by calling `TransportManager.receiveMessage()` within the scope of `weblogic.security.Security.runAs(subject)`. For information on this method, see the *Java API Reference for Oracle Service Bus*.

Note:

For information on the Java class Subject, see <http://docs.oracle.com/javase/7/docs/api/javax/security/auth/Subject.html>.

The proxy services uses this Subject in the following ways:

- During access control to the proxy service
- To populate the message context variable `$inbound/ctx:security/ctx:transportClient/*`

- As the input for identity propagation and credential mapping (unless there is also message-level client authentication)

If the transport provider does not support authentication, or if it supports anonymous requests, it must make sure the anonymous subject is on the thread before dispatching the request. Typically the transport provider will already be running as anonymous, but if this is not the case, then the provider must make the following calls:

```
Subject anonymous = SubjectUtils.getAnonymousUser()
Security.runAs(anonymous, action)
```

The transport provider is also responsible for providing any Oracle Service Bus Console configuration pages required to configure inbound client authentication. The transport provider must clearly document its inbound authentication model.

Outbound Request Authentication

Transport providers are free to implement whatever outbound authentication schemes are appropriate to that transport. The Transport SDK includes methods to facilitate outbound user name and password authentication, (two-way) SSL client authentication, and JAAS Subject authentication.

Outbound User Name and Password Authentication

Outbound user name and password authentication can be implemented by leveraging Service Bus service accounts. Service accounts are first-class, top-level Service Bus resources. You create and manage service accounts in the Oracle Service Bus Console or in JDeveloper. Transport providers are free to design their transport-specific configuration to include references to service accounts. That way the transport provider can make use of the credential management mechanisms provided by the service accounts.

Transport providers are not concerned with the details of service account configuration. There are three types of service accounts:

- **Static:** A static service account is configured with a fixed user name and password.
- **Mapped:** A mapped service account contains a list of remote-users and remote-passwords, along with a map from local-users to remote-users. Mapped service accounts can optionally map the anonymous subject to a given remote user.
- **Pass-through:** A pass-through service account indicates that the user name and password of the Service Bus client must be sent to the back-end.

An outbound endpoint can have a reference to a service account. The reference to the service account must be stored in the transport-specific endpoint configuration. When a proxy service routes a message to this outbound endpoint, the transport provider passes the service account reference to

```
CredentialCallback.getUsernamePasswordCredential(ref). Service Bus
```

returns the user name and password according to the service account configuration. This has the advantage of separating identity propagation and credential mapping configuration from the transport-specific details, simplifying the Transport SDK. It also allows sharing this configuration. Any number of endpoints can reference the same service account.

Note:

The `CredentialCallback` object is made available to the transport provider by calling `TransportSender.getCredentialCallback()`.

`CredentialCallback.getUsernameAndPasswordCredential()` returns a `weblogic.security.UsernameAndPassword` instance. This is a simple class that has methods to get the user name and password. The user name and password returned depends on the type of service account. If the service account is of type static, the fixed user name and password is returned. If it is mapped, the client subject is used to look up the remote user name and password. If it is pass-through, the client's user name and password is returned.

Note:

A mapped service account throws a `CredentialNotFoundException` if one of the following occurs:

- There is no map for the inbound client.
 - The inbound security context is anonymous and there is no anonymous map.
-

Outbound SSL Client Authentication (Two-Way SSL)

Service Bus supports outbound SSL client authentication. In this case, the proxy service making the outbound SSL request must be configured with a PKI key-pair for SSL. This is done with a reference to a proxy service provider, and the details are out of the scope of this document. To obtain the key-pair for SSL client authentication, the transport provider must call `CredentialCallback.getKeyPair()`. The HTTPS transport provider is an example of this.

Outbound JAAS Subject Authentication

Some transport providers send a serialized JAAS Subject on the wire as an authentication token. To obtain the inbound subject the transport provider must call `CredentialCallback.getSubject()`.

Note:

The return value may be the anonymous subject.

Link-Level or Connection-Level Credentials

Some transports require credentials to connect to services. For example, FTP endpoints may be required to authenticate to the FTP server. Transport providers can make use of static service accounts to retrieve a user name and password for establishing the connection. Note that mapped or pass-through service accounts cannot be used in this case because these connections are not made on behalf of a particular client request. If a transport provider decides to follow this approach, the endpoint must be configured with a reference to a service account. At runtime, the provider must call `TransportManagerHelper.getUsernameAndPassword()`, passing the reference to the static service account.

Uniform Access Control to Proxy Services

Service Bus enforces access control to proxy services for every inbound request. Transport providers are not required to enforce access control or to provide interfaces to manage the access control policy.

The access control policy covers the majority of the use cases; however, a transport provider can implement its own access control mechanisms in addition to the access control check done by Service Bus if they are needed for reasons specific to the transport provider. If that is the case, contact your Oracle representative. In general Oracle recommends transport providers let Service Bus handle access control.

When access is denied, `TransportManager.receiveMessage()` throws an `AccessNotAllowedException` wrapped inside a `TransportException`. Transport providers are responsible for checking the root-cause of the `TransportException`. A transport provider may do special error handling when the root cause is an `AccessNotAllowedException`. For example, the HTTP/S transport provider returns an HTTP 403 (forbidden) error code in this case.

Note:

Service Bus makes the request headers available to the authorization providers for making access control decisions.

Identity Propagation and Credential Mapping

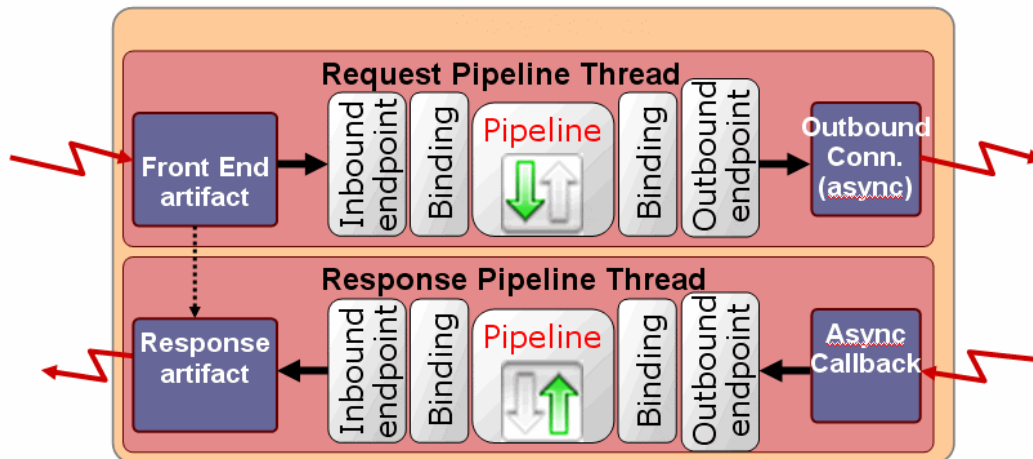
As explained in [Outbound Request Authentication](#), Service Bus provides three types of service accounts. A transport provider can make use of service accounts to get access to the user name and password for outbound authentication. A service account hides all of the details of identity propagation and credential mapping from Service Bus transport providers.

Transport SDK and the Threading Model

[Figure 39-4](#) illustrates the Service Bus threading model for a hypothetical transport endpoint processing a single inbound message.

A front end artifact, such as a servlet, is responsible for getting the inbound message. A request can be routed to an outbound endpoint and sent asynchronously. At this point, the thread is released. At some later point, a response is sent back to Service Bus (using a callback). The response is received, packaged, and handed to the Service Bus pipeline. Later, the pipeline notifies the inbound endpoint that the response is ready to be sent to the client. This processing is scalable because a thread is only tied up as long as it is needed.

Figure 39-4 Sample Service Bus Threading Model



Inbound Request Message Thread

During inbound request message processing, the following actions occur in the same thread:

1. An inbound message is received by the front end artifact of the transport endpoint. This front end artifact could be something like an HTTP servlet or JMS message-driven bean instance.
2. The message is packaged into a `TransportMessageContext` object by the transport endpoint implementation and is passed to the Service Bus runtime. For more information on the `TransportMessageContext` interface, see [Metadata and Header Representation for Request and Response Messages](#).
3. The pipeline performs the configured request pipeline actions.
4. While processing the inbound message in the pipeline, in the same (request) thread, Service Bus runtime calls on the registered outbound transport endpoint, which may or may not be managed by the same provider, to deliver an outbound message to an external service.
5. At some later point, the external service asynchronously calls on the outbound endpoint to deliver the response message. The outbound endpoint must have been registered previously with a transport specific callback object.

Note:

At this point, the initial request thread is released and placed back into the Oracle WebLogic Server thread pool for use by another request.

Outbound Response Message Thread

During outbound response message processing, the following actions occur in the same thread:

1. The response message is packaged into a `TransportMessageContext` object and delivered back to the Service Bus runtime for response processing. This

processing occurs in a different thread than the request thread. This new thread is called the response thread.

2. After the response message is processed, Service Bus runtime calls on the `InboundTransportMessageContext` object to notify it to send the response back to the original caller. For more information on the `InboundTransportMessageContext` interface, see [Metadata and Header Representation for Request and Response Messages](#).

If the transport provider does not have a native implementation of an asynchronous (non-blocking) outbound call, it still needs to deliver the response back to the Service Bus runtime on a separate thread than that on which the inbound request message was received. To do this, it can execute the call in a blocking fashion in the request thread and then use a Transport SDK helper method to deliver the response back to the Service Bus runtime.

For example, the EJB transport provider does not have an asynchronous (non-blocking) outbound call. The underlying API is a blocking API. To work around this, the provider makes its blocking call, then schedules the response for processing with `TransportManagerHelper.schedule()`. For more information on the EJB transport provider, see [Using the EJB Transport](#).

Support for Asynchrony

By design, the transport subsystem interacts asynchronously with Service Bus. This is because asynchronous behavior is more scalable, and therefore, more desirable than synchronous behavior. Rather than create two separate APIs, one for asynchronous and one for synchronous interaction, the Service Bus runtime expects asynchronous interaction. It is up to the transport developer to work around this by a method such as posting a blocking call and posting the response in a callback. In any case, the response must be executed in a different thread from the request.

Publish and Service Callout Threading

The threading diagram shown in [Figure 39-4](#) focuses on routing. The transport subsystem behaves the same way for Service Bus Publish and Service Callout actions, which can occur in the middle of the request or response pipeline processing. These actions occur outside the scope of the transport subsystem and in the scope of a Service Bus pipeline. Therefore, some differences exist between the threading behavior of Publish and Service Callout actions and transport providers.

Note the following cases:

- **Service Callout:** The pipeline processor blocks the thread until the response arrives asynchronously. The blocked thread then resumes execution of the pipeline. The purpose is to bind variables that can later be used in pipeline actions to perform business logic. Therefore, these actions must block so that the business logic can be performed before the response comes back.
- **Publish:** The pipeline processor may or may not block the thread until the response arrives asynchronously. This thread then continues execution of the rest of the request or response pipeline processing.

Tip:

A Service Callout action allows you to configure a synchronous (blocking) call to a proxy or business service that is already registered with Service Bus. Use a Publish action to identify a target service for a message and configure how the message is packaged and sent to that service. For more information on Service Callout and Publish actions, see [Adding Service Callout Actions in the Console](#) and [Adding Publish Actions in the Console](#).

Designing for Message Content

Transport providers have their own native representation of message content. For example, the HTTP transport uses `java.io.InputStream`, JMS has Message objects of various types, Tuxedo has buffers, and the Oracle WebLogic Server Web Services stack uses SAAJ. However, within the runtime of a proxy service, the native representation of content is the Message Context. While Service Bus supports some common conversion scenarios, such as `InputStream` to and from Message Context, this conversion between transport representation and the Message Context is ultimately the transport provider's responsibility.

In general, the Transport SDK is not concerned with converting directly between two different transport representations of content. However, if two transports use compatible representations and the content does not require re-encoding, the SDK may allow the source content to be passed-through directly (for example, passing a `FileInputStream` from an inbound File transport to an outbound HTTP transport). However, if the source content requires any sort of processing, it makes more sense to unmarshal the source content into the Message Context first and then use the standard mechanisms to generate content for the outgoing transport.

Sources and Transformers

Content is represented as an instance of the `Source` interface. Transport SDK interfaces that deal with message content, such as `TransportSender` and `TransportMessageContext`, all use the `Source` interface when passing message payloads. The requirements on a `Source` are minimal. A `Source` must support push- and pull-based conversions to byte-based streams using the two methods defined in the base `Source` interface. A `Source` may or may not take into account various transformation options, such as character-set encoding, during serialization, as specified by the `TransformOptions` parameter.

While all `Source` objects must implement the base serialization interface, the underlying representation of the `Source` object's content is implementation specific. This allows for `Source` objects based on `InputStreams`, JMS Message objects, `Strings`, or whatever representation is most natural to a particular transport. Typically, `Source` implementations allow direct access to the underlying content, in addition to the base serialization methods. For example, `StringSource`, which internally uses a `String` object to store its content offers a `getString()` method to get at the internal data. The ultimate consumer of a `Source` can then extract the underlying content by calling these source-specific APIs and potentially avoid any serialization overheads.

Sources may also be transformed into other types of sources using a `Transformer` object. If a `Source` consumer, such as a transport provider, is given a `Source` instance that it does not recognize, it can often transform it into a `Source` instance that it does recognize. The underlying content can then be extracted from that known `Source` using the source-specific APIs. However, often a transport provider simply serializes the content and send it using the base serialization methods. For more information, see [Source and Transformer Classes and Interfaces](#).

Sources and the MessageContext Object

Sources are the common content representation between the transport layer and the binding layer. The binding layer is the entity responsible for converting content between the `Source` representation used by the transport layer and the `MessageContext` used by the pipeline runtime. How that conversion happens depends upon the type of service (its binding type) and the presence of attachments. While not strictly part of the Transport SDK, any transport provider that defines its own `Source` objects should be familiar with this conversion process.

When attachments are not present, the incoming `Source` represents just the core message content. The `MessageContext` is initialized by converting the received `Source` to a specific type of `Source` and then extracting the underlying content. For example, for XML-based services, the incoming `Source` is converted to an `XmlObjectSource`. The `XmlObject` is then extracted from the `XmlObjectSource` and used as the payload inside the `$body` context variable. SOAP services are similarly converted to `XmlObjectSource` except that the extracted `XmlObject` must be a SOAP Envelope so that the `<SOAP:Header>` and `<SOAP:Body>` elements can be extracted to initialize the `$header` and `$body` context variables.

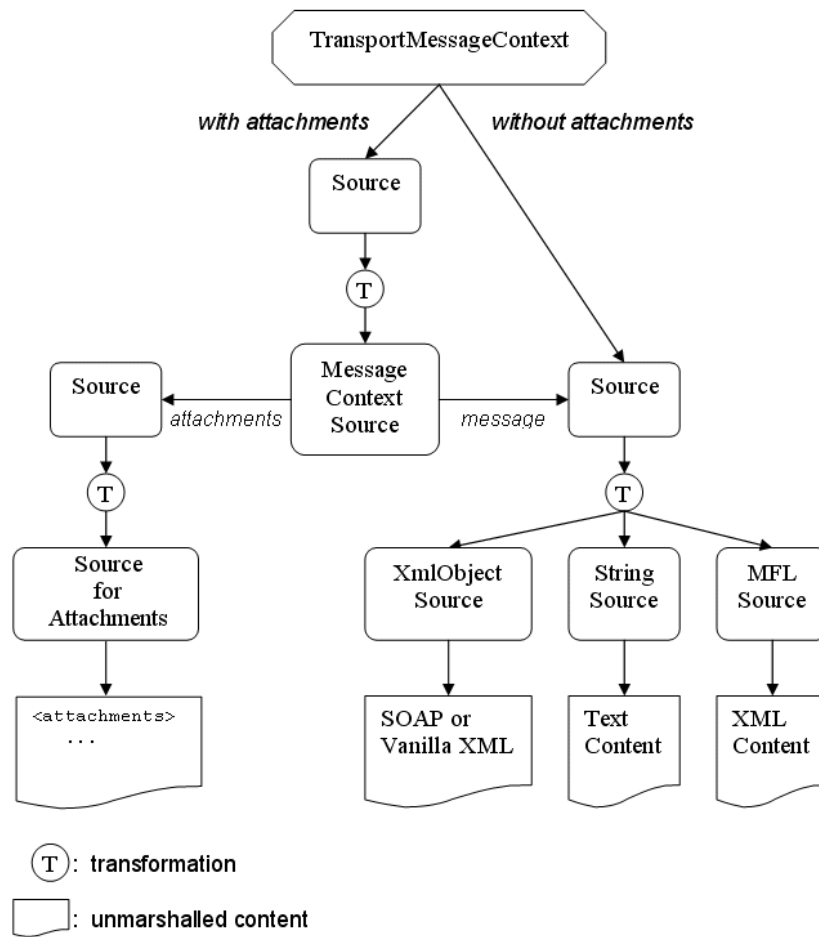
Below are the canonical `Source` types used for the set of defined service-types:

- **SOAP:** `XmlObjectSource`
- **XML:** `XmlObjectSource`
- **TEXT:** `StringSource`
- **MFL:** `MFLSource`

For binary services, no `Source` conversion is done. Instead, the `Source` is registered with a `SourceRepository` and the resulting `<binary-content/>` XML is used as the payload inside `$body`.

When attachments are present, the incoming `Source` is first converted to a `MessageContextSource`. From the `MessageContextSource`, two untyped `Source` objects are obtained, one representing the attachments and one representing the core message. The `Source` for the core message is handled as described previously. The `Source` representing attachments is converted to an `AttachmentsSource`. From the `AttachmentsSource`, XML is obtained and is used to initialize the `$attachments` context variable and a `SourceRepository` containing the registered `Sources` that represent any binary attachment content. This entire process is illustrated in [Figure 39-5](#).

Figure 39-5 Flow of Attachments



A similar conversion occurs when creating a Source from data in the MessageContext to be passed to the transport layer. The core message is represented by a Source instance that can be converted to the canonical Source for the service type. In most cases, the Source is already an instance of the canonical Source, but not always. When attachments are present, the Source delivered to the transport layer will be a source that can be converted to an instance of MessageContextSource. If the transport provider supports Content-Type as a pre-defined transport header, then the delivered Source will likely be an instance of MessageContextSource. Otherwise, the delivered Source is likely an instance of MimeSource, but this can also be converted to a MessageContextSource.

The reason for this difference is that transports that natively support Content-Type as a transport header require that the top-level MIME headers appear in the transport headers rather than in the payload. Examples of this are HTTP and email. Transports that do not natively support Content-Type must have these top-level MIME headers as part of the payload, as the Content-Type header is critical for decoding a multipart MIME package.

Built-In Transformations

Table 39-1 shows sources and lists the source types to which they can be converted by built-in transformers. For example, there is a built-in transformer that handles converting a StringSource into an XmlObjectSource; however, there is no transformer that can convert a StringSource into an MFLSource. Typically, these

transformers take advantage of their knowledge of the internal data representation used by both `Source` types.

Table 39-1 Built-In Transformations

Public Source	Can Be Transformed To
<code>Source</code>	<ul style="list-style-type: none"> • <code>StreamSource</code> • <code>ByteArraySource</code> • <code>StringSource</code> • <code>XmlObjectSource</code> • <code>DOMSource</code> • <code>MFLSource</code> • <code>SAAJSource</code>
<code>StreamSource</code>	<code>StreamSource</code>
<code>ByteArraySource</code>	<code>ByteArraySource</code>
<code>StringSource</code>	<ul style="list-style-type: none"> • <code>StringSource</code> • <code>XmlObjectSource</code> • <code>DOMSource</code>
<code>XmlObjectSource</code>	<ul style="list-style-type: none"> • <code>StringSource</code> • <code>XmlObjectSource</code> • <code>DOMSource</code> • <code>MFLSource</code>
<code>DOMSource</code>	<ul style="list-style-type: none"> • <code>StringSource</code> • <code>XmlObjectSource</code> • <code>DOMSource</code> • <code>MFLSource</code>
<code>MFLSource</code>	<ul style="list-style-type: none"> • <code>XmlObjectSource</code> • <code>DOMSource</code> • <code>MFLSource</code>
<code>MimeSource</code>	<ul style="list-style-type: none"> • <code>MimeSource</code> • <code>SAAJSource</code> • <code>MessageContextSource</code>
<code>SAAJSource</code>	<ul style="list-style-type: none"> • <code>MimeSource</code> • <code>SAAJSource</code> • <code>MessageContextSource</code>
<code>MessageContextSource</code>	<ul style="list-style-type: none"> • <code>MimeSource</code> • <code>SAAJSource</code> • <code>MessageContextSource</code>

These generic transformations are done without any knowledge of the initial `Source` type but instead rely on the base serialization methods that are implemented by all `Sources`: `getInputStream()` and `writeTo()`. So, although it is ultimately possible to convert an `XmlObjectSource` to a `ByteArraySource`, it is not done using any special knowledge of the internal details of `XmlObjectSource`.

Note:

Many custom sources implemented by transports can be handled by these generic transformations, especially if the underlying data is an unstructured collection of bytes. For example, the File Transport uses a custom source that pulls its content directly from a file on disk. However, that data is just a set of bytes without structure, so there is no need to provide custom transformations to, for example, `XmlObjectSource`. The generic transformation can handle this custom `FileSource` using just the base serialization methods that all `Sources` must implement.

For more information, see [Source and Transformer Classes and Interfaces](#).

Developing Custom Transport Providers

This chapter describes the basic steps involved in developing a custom transport provider. The Transport SDK provides a layer of abstraction between transport protocols and the Service Bus runtime system. This layer of abstraction makes it possible to develop and plug in new transport providers to Service Bus. The Transport SDK interfaces provide this bridge between transport protocols, such as HTTP, and the Service Bus runtime.

Tip:

Before beginning this chapter, be sure to review [Learning About Custom Transport Providers](#).

This chapter includes the following sections:

- [Development Road Map](#)
- [Before You Begin](#)
- [Basic Development Steps](#)
- [Important Development Topics](#)
- [Creating Help for Custom Transports](#)

Development Road Map

The process of designing and building a custom transport provider is complex. This section offers a recommended path to follow as you develop your transport provider. Development of a custom transport provider breaks down into three basic stages: Planning, developing, and packaging and deploying.

- [Planning](#)
- [Developing](#)
- [Packaging and Deploying](#)

Planning

Perform the following planning steps before developing a custom transport provider.

1. Decide if you really need to develop a custom transport provider. See [Determining Whether to Develop a Custom Transport Provider](#)
2. Run and study the example socket transport provider. The source code for this provider is installed with Service Bus and is publicly available for you to examine and reuse. See [Creating a Sample Socket Transport Provider](#).

3. Review [Learning About Custom Transport Providers](#). This chapter discusses the architecture of a transport provider and many aspects of transporter provider design, such as the security model and the threading model employed by transport providers.
4. Review [Before You Begin](#).

Developing

Custom transport development steps include creating required artifacts, such as XML schema files, configuration components, and user interfaces. [Basic Development Steps](#) describes steps you need to take to develop a transport provider. Before developing your custom transport, review [Important Development Topics](#), which discusses several topics that you might need to refer to during the development cycle, such as message and error handling and transforming messages.

Packaging and Deploying

For detailed information on packaging and deploying a transport provider, see [Packaging and Deploying a Custom Transport Provider](#).

Before You Begin

There are several design considerations to take into account before you begin to develop a custom transport provider, including the following:

- Determine whether you really need to develop a custom transport provider. See [Determining Whether to Develop a Custom Transport Provider](#).
- Determine whether your message endpoints are transactional or non-transactional. See [Transactional vs. Non-Transactional Endpoints](#).
- Determine whether your message endpoints are one way, synchronous, or asynchronous. See [Supported Message Patterns](#) and [Support for Synchronous Transactions](#).
- Determine the security requirements for outgoing and incoming messages. See [Transport SDK Security Model](#).
- Understand the threading model used by Service Bus. See [Transport SDK and the Threading Model](#).
- Determine whether your transport provider should support synchronous or asynchronous outbound calls. See [Support for Asynchrony](#).
- Review the interfaces and classes provided with the Transport SDK, and become familiar with how they fit into the design-time and runtime parts of a transport provider. See [Transport SDK Interfaces and Classes](#).
- Understand how to package and deploy a custom transport provider. See [Packaging and Deploying a Custom Transport Provider](#).
- Review the flow of method calls through the transport framework. See [Transport SDK UML Sequence Diagrams](#).

Basic Development Steps

Below are the basic steps to follow when developing a custom transport provider:

Step 1. Review the Transport Framework Components

Step 2. Create a Directory Structure for Your Transport Project

Step 3. Create an XML Schema File for Transport-Specific Artifacts

Step 4. Define Transport-Specific Artifacts

Step 5. Define the TransportProviderConfiguration XMLBean

Step 6. Implement the Transport Provider User Interface

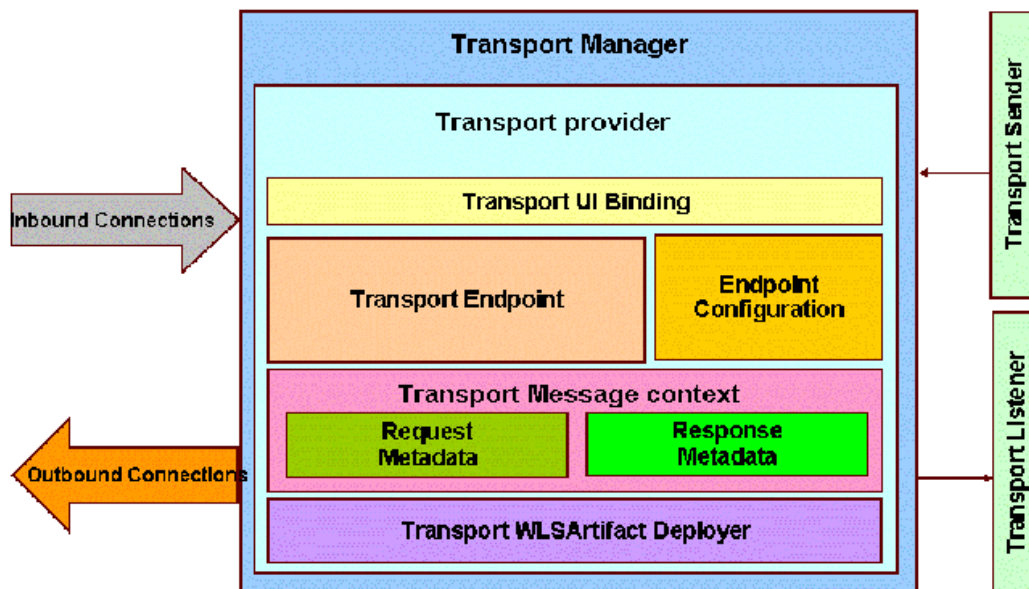
Step 7. Implement the Runtime Interfaces

Step 8. Package and Deploy the Transport Provider

Step 1. Review the Transport Framework Components

Figure 40-1 illustrates the components that you must implement and configure to create a custom transport provider. The transport manager controls and manages the registration of transport providers and handles communication with Service Bus. A transport provider manages the life cycle and runtime behavior of transport endpoints (resources where messages originate or are targeted). You use the Transport SDK to develop custom transport providers.

Figure 40-1 Transport Subsystem Overview



The parts of the transport subsystem that you must implement and configure include the following:

- **Transport UI bindings:** The user interface component for the transport provider. Related interfaces are summarized in [User Interface Configuration](#).
- **Transport endpoint:** Responsible for sending and accepting messages. Related interfaces are summarized in [General Classes and Interfaces](#).

- **Endpoint configuration:** Stores endpoint configurations. Related interfaces are listed in [Schema-Generated Interfaces](#).
- **Transport message context:** Contains metadata for request and response headers and other parts of the message (inbound and outbound). For additional information, see [Source and Transformer Classes and Interfaces](#) and [Metadata and Header Representation for Request and Response Messages](#).
- **WLS artifact deployer:** (optional) Deploys artifacts, such as servlets that receive and send messages.
- **Transport sender:** Retrieves metadata for the outbound message and the payload. Related interfaces are summarized in [Summary of General Interfaces](#).
- **Transport listener:** Allows the outbound endpoint to post the result of an outbound request to the rest of Service Bus. See also [Metadata and Header Representation for Request and Response Messages](#).
- **Request/Response Metadata:** Related interfaces are summarized in [Metadata and Header Representation for Request and Response Messages](#).

Step 2. Create a Directory Structure for Your Transport Project

Before developing a new transport provider, take time to set up an appropriate directory structure for your project. The recommended approach is to copy the directory structure used for the sample socket transport provider. For a detailed description of this structure, see [Sample Location and Directory Structure](#) .

Step 3. Create an XML Schema File for Transport-Specific Artifacts

Create an XML schema (XSD) file for transport-specific definitions. You can base this file on the schema file developed for the sample socket transport provider:

```
OSB_ORACLE_HOME/samples/servicebus/sample-transport/schemas/  
SocketTransport.xsd
```

Note:

The `SocketTransport.xsd` file imports the file `TransportCommon.xsd`. This file is the base schema definition file for service endpoint configurations. This file is located in `OSB_ORACLE_HOME/lib/modules/oracle.servicebus.kernel-api.jar`. You might want to review the contents of this file before continuing.

Step 4. Define Transport-Specific Artifacts

Define XML schemas for the following transport-specific artifacts in the XML schema file described in the previous section, [Step 3. Create an XML Schema File for Transport-Specific Artifacts](#).

- `EndpointConfiguration`
- `RequestMetaDataXML`
- `ResponseMetaDataXML`

Each of these schema definitions is converted into a corresponding Java file and compiled. Once you build the sample socket transport provider, you can find

examples of these converted Java source files in `OSB_ORACLE_HOME/samples/servicebus/sample-transport/build/classes/com/bea/alsb/transports/sock/impl`. Only simple XML types are supported when defining metadata and headers specific to the transport provider. For example, complex types with nested elements are not supported. Furthermore, there can be at most one header with a given name.

EndPointConfiguration

`EndPointConfiguration` is the base type for endpoint configuration, and describes the complete set of parameters necessary for the deployment and operation of an inbound or outbound endpoint. This configuration consists of generic and provider-specific parts. For more information on the `EndPointConfiguration` schema definition, refer to the documentation elements in the `TransportCommon.xsd` file.

You need to specify a provider-specific endpoint configuration in the schema file. The following example shows an excerpt from the `SocketTransport.xsd`.

Example - Sample SocketEndPointConfiguration Definition

```
<xs:complexType name="SocketEndpointConfiguration">
  <xs:annotation>
    <xs:documentation>
      SocketTransport - specific configuration
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:choice>
      <xs:element name="outbound-properties"
        type="SocketOutboundPropertiesType"/>
      <xs:element name="inbound-properties"
        type="SocketInboundPropertiesType"/>
    </xs:choice>
    <xs:element name="request-response" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>
          Whether the message pattern is synchronous
          request-response or one-way.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    ...
  </xs:sequence>
</xs:complexType>
```

RequestMetaDataXML

Each transport provider must store metadata (message headers) in a Plain Old Java Object (POJO) and pass that to the pipeline. Examples of information that might be transmitted in the metadata are the Content-Type header, security information, or locale information. A `RequestMetaData` POJO is a generic object that extends the `RequestMetaData` abstract class and describes the message metadata of the incoming or outgoing request. The transport provider must deliver the message metadata to the Service Bus runtime in a `RequestMetaData` POJO. For additional information, see [Request and Response Metadata Handling](#).

`RequestMetaDataXML` is an XML representation of the same `RequestMetaData` POJO. This XML representation uses Apache XML bean technology. It is only needed by the Service Bus runtime when processing of the message involves any actions in the pipeline that need an XML representation of the metadata, such as setting the entire metadata to a specified XML fragment on the outbound request.

You must specify request metadata configuration in the schema file. The following example shows an excerpt from the `SocketTransport.xsd`.

Example - Sample SocketRequestMetaDataXML Definition

```
<xs:complexType name="SocketRequestMetaDataXML">
  <xs:annotation>
    <xs:documentation/>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="ts:RequestMetaDataXML">
      <xs:sequence>
        <xs:element name="client-host"
          type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:documentation>
              Client host name
            </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="client-port" type="xs:int" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Client port</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

RequestHeadersXML

`RequestHeadersXML` is the base type for a set of inbound or outbound request headers. You need to specify the `RequestHeadersXML` configuration in the schema file. The following example shows an excerpt from the `SocketTransport.xsd`.

Example - Sample SocketRequestHeadersXML Definition

```
<xs:complexType name="SocketRequestHeadersXML">
  <xs:annotation>
    <xs:documentation/>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="ts:RequestHeadersXML">
      <xs:sequence>
        <xs:element name="message-count" type="xs:long" minOccurs="0">
          <xs:annotation>
            <xs:documentation>
              Number of messages passed till now.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

ResponseMetaDataXML

`ResponseMetaDataXML` is the base type for metadata for a response to an inbound or outbound message. You need to specify the `ResponseMetaDataXML` configuration

in the schema file. The following example shows an excerpt from the `SocketTransport.xsd`.

Example - Sample SocketResponseMetaDataXML Definition

```
<xs:complexType name="SocketResponseMetaDataXML">
  <xs:complexContent>
    <xs:extension base="ts:ResponseMetaDataXML">
      <xs:sequence>
        <xs:element name="service-endpoint-host"
          type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:documentation>
              Host name of the service endpoint connection.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="service-endpoint-ip"
          type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:documentation>
              IP address of the service endpoint connection.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

ResponseHeadersXML

`ResponseHeadersXML` is the base type for a set of response headers. You need to specify the `ResponseHeadersXML` configuration in the schema file. The following example shows an excerpt from the `SocketTransport.xsd`.

Example - Sample SocketResponseHeadersXML Definition

```
<xs:complexType name="SocketResponseHeadersXML">
  <xs:annotation>
    <xs:documentation/>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="ts:ResponseHeadersXML"/>
  </xs:complexContent>
</xs:complexType>
```

Step 5. Define the TransportProviderConfiguration XMLBean

To configure the `TransportProviderConfiguration` XML bean, edit the transport provider configuration file. This XML file is located in the `resources` directory in the `sample-transport` directory. Configure the file according to the following guidelines:

- If proxy services can use your transport, set the `inbound-direction-supported` element to `true`.
- If business services use your transport, set the `outbound-direction-supported` element to `true`.

- If your transport is self-described, include an element `self-described` with the value set to `true`. A self-described transport is one whose services are responsible for describing their shape (schema or WSDL file) based on their endpoint configuration.
- If you want to publish a tModel for your transport to a UDDI registry, include an element `UDDI`. See [About Publishing Proxy Services to a UDDI Registry](#) for more info.

Tip:

The schema for `TransportProviderConfiguration` is defined in `TransportCommon.xsd`, which is located in `OSB_ORACLE_HOME/lib/servicebus-schemas.jar`. Refer to the schema file for more information.

Step 6. Implement the Transport Provider User Interface

When you add a business or proxy service using the Oracle Service Bus Console, you select a transport provider in the service creation wizard. The wizard includes the transport providers that are provided with Service Bus and any custom transport providers that were developed with the Transport SDK. When you configure a business or proxy service, properties specific to the transport being used appear in the editor. These are all part of the user interface you need to develop for the custom transport provider.

This section discusses the Transport SDK API components that bind your custom transport provider to the Oracle Service Bus Console user interface. You must implement these APIs to connect your provider to the user interface.

Tip:

This section assumes that you are familiar with the service creation wizards and the service definition editors. See [Creating a Socket Transport Sample Project](#), for a detailed, illustrated example.

Transport Configuration Processing Flow

1. When users create a new service, they must select an appropriate transport provider on the service creation wizard. They then also select a service type, such as SOAP, WSDL-based, XML, or messaging. To validate the selection, the wizard calls the following method of the `TransportUIBinding` interface:

```
public boolean isServiceTypeSupported(BindingTypeInfo binding)
```

This method determines if the transport provider is suitable for the selected service type.

2. Users then enter an endpoint URI. To validate this URI, the wizard calls the following method of the `TransportUIBinding` interface:

```
public TransportUIError[] validateMainForm(TransportEditField[] fields)
```

3. Once you create a service, the console displays the service definition editor, which includes a transport-specific configuration page. To render this page, the editor calls the following method of the `TransportUIBinding` interface:

```
public TransportEditField[] getEditPage(EndPointConfiguration config,  
BindingTypeInfo binding) throws TransportException
```

The Transport SDK offers a set of `TransportUIObjects` that represent fields on the configuration page. For example, you can add text boxes, check boxes, and other types of UI elements. Use the `TransportUIFactory` to create them, and then use the same factory to specify additional properties and obtain `TransportEditField` objects that can be displayed on the service definition editors.

Tip:

You can associate events with most of the UI fields. An event acts like a callback mechanism for the `TransportUIBinding` class and lets you refresh, validate, and update the configuration page. When an event is triggered, the wizard calls the following method:

```
updateEditPage(TransportEditField[] fields, String name) throws TransportException
```

4. When users complete the transport configuration, the editor calls the validation method:

```
TransportUIError[] validateProviderSpecificForm(TransportEditField[] fields)
```

5. After the service is saved, the transport manager calls the following method of the `TransportProvider` class:

```
void validateEndPointConfiguration(TransportValidationContext context)
```

If no error is reported, a new endpoint is created. The transport manager then calls the following method:

```
TransportEndPoint createEndPoint(EndPointOperations.Create context) throws TransportException
```

If this method returns successfully, the new service is listed and the underlying transport configuration is associated with an endpoint on the `TransportProvider`.

Note:

The endpoint configuration is saved in the Service Bus session and does not need to be persisted or recovered in case of a server restart by the transport provider.

6. Once the session is activated, you must deploy the endpoint to start processing requests. To learn more about deploying an endpoint and processing requests, see [When to Implement TransportWLSArtifactDeployer](#) and [Deploying to a Cluster..](#)

Tip:

For the sample socket transport provider, you can find the implementations of these interfaces in the `sample-transport/src` directory.

Step 7. Implement the Runtime Interfaces

A new custom transport provider must implement certain runtime interfaces. For a summary of the Transport SDK interfaces and related classes, see [Transport SDK Interfaces and Classes](#). For detailed information on interfaces and classes, see the *Java API Reference for Oracle Service Bus*.

Tip:

For the sample socket transport provider, you can find the implementations of these interfaces in the `sample-transport/src` directory.

You must implement the following interfaces when developing a custom transport provider:

- `TransportProvider`
- `TransportWLSArtifactDeployer`
- `TransportEndPoint`
- `InboundTransportMessageContext`
- `OutboundTransportMessageContext`
- `Transformer`

Note:

- Only implement the `TransportWLSArtifactDeployer` interface if the transport provider needs to deploy WebLogic Server-related artifacts, such as EAR, WAR, and JAR files, that go into a WebLogic Server change list at the time of endpoint creation. For more information, see [When to Implement TransportWLSArtifactDeployer](#).
 - Only implement the `Transformer` interface if the transport provider needs to work with non-standard payload bindings, for example, anything other than Stream, DOM, SAX, or XMLBean. For more information, see [Transforming Messages](#).
-
-

Step 8. Package and Deploy the Transport Provider

For information about this process, see [Packaging and Deploying a Custom Transport Provider](#).

Important Development Topics

This section discusses several topics that you should consider when developing a custom transport provider, including message and error handling, message transformation, transport options, environment values, UDDI registries, and so on.

- [Handling Messages](#)
- [Transforming Messages](#)
- [Working with TransportOptions](#)
- [Handling Errors](#)
- [Defining Custom Environment Value Types](#)
- [Publishing Proxy Services to a UDDI Registry](#)
- [When to Implement TransportWLSArtifactDeployer](#)

Handling Messages

The Transport SDK features a flexible representation of message payloads. All Transport SDK APIs dealing with payload use the Source interface to represent message content.

The Source-derived message types provided with the Transport SDK include:

- StreamSource
- ByteArraySource
- StringSource
- XmlObjectSource
- DOMSource
- MFLSource
- SAAJSource
- MimeSource

Note:

StreamSource is a single use source; that is, it implements the marker interface SingleUseSource. With the other Sources, you can get the input stream from the source multiple times. Each time the Source object gets the input stream from the beginning. With a SingleUseSource, you can only get the input stream once. Once the input is consumed, it is gone (for example, a stream from a network socket); however, Service Bus buffers the input from a SingleUseSource, essentially keeping a copy of all of its data.

If you implement a Source class for your transport provider, you need to determine whether you can re-get the input stream from the beginning. If the nature of the input stream is that it can only be consumed once, your Source class should implement the marker interface SingleUseStream.

The Transport SDK provides a set of transformers to convert between source objects. You can implement new transformations, as needed, as long as they support transformations to and from a set of canonical representations. For more information, see [Transforming Messages](#) and [Designing for Message Content](#).

Sending and Receiving Message Data

When implementing inbound endpoints to deliver the inbound message to the Service Bus runtime, you need to call `TransportManager.receiveMessage()`. The transport provider is free to expose the incoming message payload in either one of the standard source-derived objects, such as stream, DOM or SAX, or a custom one.

If Service Bus needs to send a response message back to the client that sent the request, it will call methods `setResponseMetaData()` and `setResponsePayload()` followed by `close()` on `InboundTransportMessageContext` to indicate that the response is ready to be sent back. When the Service Bus runtime calls the inbound transport message context `close()` method, this is done from a different thread than that on which the inbound request message was received. The transport provider should be aware of this because it may affect the semantics of transactions. Also, the

transport provider cannot attempt to access the response payload or metadata until `close()` has been called.

Request and Response Metadata Handling

Each transport provider must store metadata and headers in a Plain Old Java Object (POJO) and pass that to the pipeline. There are some cases where Service Bus requires an XMLBean. In these cases, you need to implement a conversion from POJO to XMLBean using the API.

You must provide the following methods to convert from a POJO to XML:

```
RequestHeaders.toXML()
```

```
RequestMetaData<T>.toXML()
```

```
ResponseHeaders.toXML()
```

```
ResponseMetaData<T>.toXML()
```

For the reverse direction (XML to POJO) you need to implement:

```
TransportEndPoint.createRequestMetaData(RequestMetaDataXML)
```

```
InboundTransportMessageContext.createResponseMetaData(ResponseMetaDataXML)
```

Character Set Encoding

Each transport provider is responsible for specifying the character set encoding of the incoming message payload to Service Bus. For outgoing messages (outbound request and inbound response), the transport provider is responsible for telling Service Bus what character set encoding to use for the outgoing payload. The character-set encoding is specified in request and response metadata.

In virtually every case, the character-set encoding that the transport is responsible for inserting into the metadata is exactly the encoding that is statically specified in the service configuration. One of the few exceptions to this is HTTP transport, which inspects Content-Type for any "charset" parameters and overrides any encoding configured in the service. This is necessary in order to conform to HTTP specifications. Other transport protocols may need to handle similar issues.

Tip:

In general, the encoding for a service is fixed. If someone sends a UTF-16 encoded message to a proxy that is specified to be SHIFT_JIS, then that is generally considered to be an error. Transport providers should not need to inspect the message simply to determine encoding.

For outgoing messages, the transport provider tells Service Bus what encoding it requires for the outbound request, and Service Bus performs the conversion if necessary.

Transports should always rely on this encoding for outgoing messages and should not assume that it is the same as the encoding specified in the service configuration. If there is a discrepancy, the transport can choose to allow it, but others could consider it an error and throw an exception. Also the transport has the additional option of leaving the encoding element blank. That leaves the pipeline free to specify the encoding (for example, using pass-through).

Co-Located Calls

If a given transport provider supports proxy service endpoints, you can configure the request pipeline such that there is a routing step that routes to that provider's proxy service. Furthermore there could be a Publish or a Service Callout action that sends a message to a proxy service instead of a business service. This use case is referred to as co-located calls.

The transport provider needs to be aware of co-located calls, and handle them accordingly. Depending on the nature of the proxy service endpoint implementation, the transport provider may choose to optimize the invocation such that this call bypasses the entire transport communication stack and any inbound authentication and authorization, and instead is a direct call that effectively calls `TransportManager.receiveMessage()` immediately.

Tip:

Service Bus has implemented this optimization with the HTTP, File, Email and FTP transport providers. The JMS provider does not use this optimization due to the desire to separate the transactional semantics of send operation versus receive operations.

If you want to use this optimization in a custom transport provider, you need to extend the `CoLocatedMessageContext` class and call its `send()` method when `TransportProvider.sendMessageAsync()` is invoked.

Returning Outbound Responses to the Service Bus Runtime

When the Service Bus runtime sends a message to an outbound endpoint and there is a response message to be returned, the transport provider must return this response asynchronously. That means `TransportSendListener.onReceiveResponse()` or `TransportSendListener.onError()` methods need to be called from a different thread than the one on which `TransportProvider.sendMessageAsync()` was called.

If the transport provider has a built-in mechanism by which the response arrives asynchronously, such as responses to JMS requests or HTTP requests when the async response option is used, it happens naturally. However, if the transport provider has no built-in mechanism for retrieving responses asynchronously, it can execute the outbound request in a blocking fashion and then schedule a new worker thread using the `TransportManagerHelper.schedule()` method, in which the response is posted to the `TransportSendListener`.

Transforming Messages

When Service Bus needs to set either the request payload to an outbound message or the response payload to an inbound message, it asks the transport provider to do so through an object derived from the `Source` interface. The transport provider then needs to decide what representation the underlying transport layer requires and use the `Transformer.transform()` method to translate the `Source` object into the desired source.

Tip:

For more information on message transformation, see [Designing for Message Content](#). For a list of built-in transformations, see [Built-In Transformations and Source and Transformer Classes and Interfaces](#).

A custom transport provider can support new kinds of transformations. Suppose a transport provider needs to work with a DOM object in order to send the outbound message. When called with `setRequestPayload(Source src)`, the transport provider needs to call the method:

```
TransportManagerHelper.getTransportManager().getTransformer().  
transform(src, DOMSource.class, transformOptions)
```

The return value of the method gives a `DOMSource`, which can then be used to retrieve the DOM node.

Note:

If the transport provider requires a stream, there is a shortcut: each `Source` object supports transformation to stream natively.

You can add new transformations to a custom transport provider. For example, suppose you want to add a new kind of `Source`-derived class, called `XYZSource`. For performance reasons, transport providers are encouraged to provide conversions from `XYZSource` to one of the two canonical `Source` objects, `XmlObjectSource` and `StreamSource` when applicable. Without such transformation, generic transformers are used, which rely on the `StreamSource` representation of `XYZSource`. Of course, if `XYZSource` is a simple byte-based `Source` with no internal structure, then relying on the generic transformers is usually sufficient. Note that any custom transformer that is registered with `TransportManager` is assumed to be thread-safe and stateless.

To support attachments, the transport provider has the following three options:

- The `Source` returned by `TransportMessageContext` must be an instance of `MessageContextSource`. A limitation of this option is that `MessageContextSource` requires that the content has already been partitioned into a core-message `Source` and an attachments `Source`.
- The `Source` is an instance of `MimeSource` and the `Headers` objects contain a multipart Content-Type header.
- The Content-Type is a pre-defined header for the transport provider with the specific value `multipart/related`. Both HTTP and email transports rely on this third option for supporting attachments.

Working with TransportOptions

A `TransportOptions` object is used to supply options for sending or receiving a message. A `TransportOptions` object is passed from the transport provider to the transport manager on inbound messages. On outbound messages, a `TransportOptions` object is passed from the Service Bus runtime to the transport manager, and finally to the transport provider.

Inbound Processing

The transport provider supplies the following parameters to `TransportManager.receiveMessage()`:

- **QOS:** Specifies exactly-once or best-effort quality of service. Exactly-once quality of service is specified when the incoming message is transactional.

- **Throw On Error:** If this flag is set, an exception is thrown to the callee of method `TransportManager.receiveMessage()` when an error occurs during the Service Bus pipeline processing. The options for throwing the exception include: throw the exception back to the inbound message or create a response message from the error and notify the inbound message with the response message. Typically, you set **Throw On Error** to true when QOS is exactly-once (for transactional messages).

For example, JMS/XA sets this flag to true to throw the exception in the same request thread, so it can mark the exception for rollback. HTTP sets the flag to false, because there is no retry mechanism. The error is converted to a status code and a response message is returned.

- **Any transport-specific opaque data:** Opaque data can be any data that is set by the transport provider and passed through the pipeline to the outbound call. This technique optimizes performance when the same transport is used on inbound and outbound. The opaque data is passed directly through the pipeline from the inbound transport to the outbound transport. For example, the HTTP transport provider can pass the user name and password directly from the inbound to the outbound to efficiently support identity pass-through propagation.

Outbound Processing

For outbound processing, the Service Bus runtime supplies parameters to the transport manager, which uses some of the parameters internally and propagates some parameters to `TransportProvider.sendMessageAsync()`. These parameters include the following:

- **QOS:** Specifies whether or not exactly-once quality of service can be achieved. For example, for HTTP, if quality of service is set to exactly once, the HTTP call is blocking. If it is set to best effort, it is a non-blocking HTTP call.
- **Mode:** Specifies one-way or request response. For more information, see [Transport Provider Modes](#).
- **URI, Retry Interval, and Count:** The transport provider uses the URI to initialize the outbound transport connection. For example, the HTTP transport provider uses the URI when instantiating a new `HttpURLConnection`. The transport provider is not required to use retry interval and count.
- **OperationName:** The transport provider can use `OperationName` if it needs to know what outbound web service is being used. The transport manager uses this parameter to keep track of monitoring statistics.
- **Any transport-specific opaque data:** An example of transport-specific opaque data is the value of the Authorization header for HTTP.

Request Mode

The request mode is defined as an enumeration with two values: `REQUEST_ONLY` (also called *one-way*) and `REQUEST_RESPONSE`. These modes are interpreted as follows for requests and responses:

- On outbound requests, the pipeline indicates the mode through `TransportOptions` and the transport provider must honor the mode.
- On inbound requests, the pipeline knows the mode and closes the inbound request and does not send a response if it computes the mode `REQUEST_ONLY`.

- If a response is sent by the pipeline, then there is a response even if the response is empty.
- For transports that are inherently one-way, the transport must not specify response metadata.

Handling Errors

There are three different use cases to consider with respect to the effect runtime exceptions have on the transactional model. The use cases are:

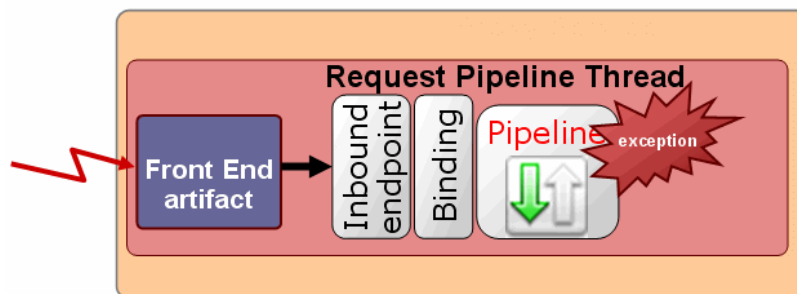
- The exception occurs somewhere in the request pipeline but before the outbound call to the business service.
- The exception occurs during the business service call.
- The exception occurs sometime after the business service call in the response pipeline.

Case 1: The Exception Occurs Before the Outbound Call

In this case, the exception occurs somewhere in the request pipeline but before the outbound call to the business service, as shown in [Figure 40-2](#). For example, executing a specific XQuery against the contents of the request message raises an exception.

If there is a user-configured error handler configured for the request pipeline, the error is handled according to the user configuration. Otherwise, the proxy service either catches an exception when calling `TransportManager.receiveMessage()` or is notified in the `InboundTransportMessageContext.close()` method of the error through response metadata, based on the transport options passed as an argument to the `receiveMessage()` call. If the proxy service indicates that the exception should be thrown, surround `receiveMessage()` with a try/catch clause and mark the transaction for rollback.

Figure 40-2 Error Case 1



Case 2: The Exception Occurs During the Outbound Call

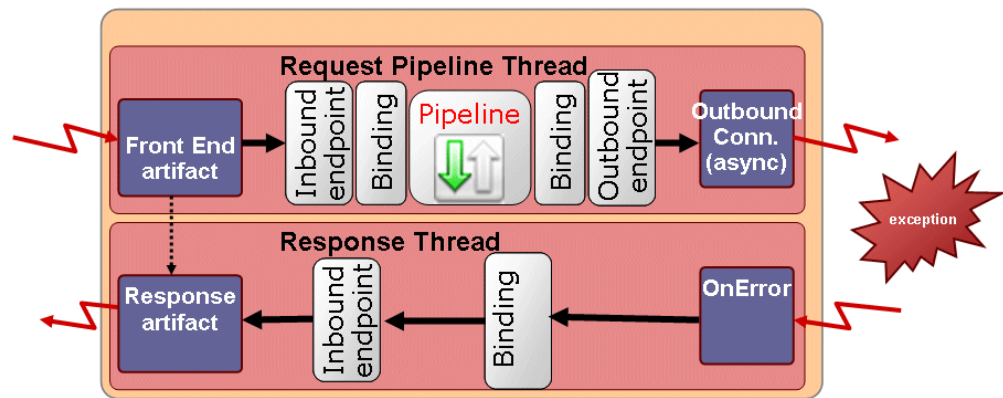
In this case, exception occurs during the business service call, as shown in [Figure 40-3](#). The outbound transport provider does one of the following:

- Throws an exception from `TransportProvider.sendMessageAsync()`. For example, the outbound provider throws an exception if there was an error while establishing a socket connection to external service. This situation could occur if the business service cannot be called because of an incorrect URL, a faulty connection, or other reasons. In these cases, the transport provider must raise an exception.

- Notifies the listener through `TransportSendListener.onError()`. For example, if the business service was called, but the call resulted in an error (such as a SOAP fault), the transport provider needs to call `TransportSendListener.onError()` instead of raising an exception.

In the first instance, the exception handling is the same as that described in [Case 1: The Exception Occurs Before the Outbound Call](#). In the second instance, if there is an error handler configured for the response pipeline, the error is handled according to the user configuration. Otherwise, the exception is propagated back to the proxy service endpoint in `InboundTransportMessageContext.close()` through the response metadata.

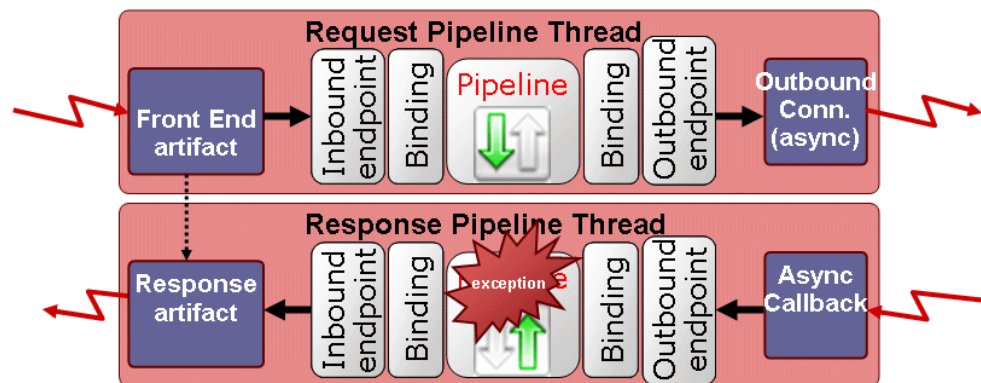
Figure 40-3 Error Case 2



Case 3: The Exception Occurs After the Outbound Call

In this case, the exception occurs sometime after the business service call in the response pipeline, as shown in [Figure 40-4](#). Again, in the absence of a user-defined error handler for the response pipeline, the proxy service endpoint is notified of the error with the `InboundTransportMessageContext.close()` method with appropriate response metadata. If the inbound transport endpoint is a synchronous transactional endpoint, it is guaranteed that the transaction that was active at the time request was received is still active and it may be rolled back. If the inbound endpoint is not transactional or not synchronous, there is not an inbound transactional context to roll back, so some other action might need to be performed.

Figure 40-4 Error Case 3



Catching Application Errors

When business services try to access an external service and an error occurs in the external service application, such as a SOAP fault, subsequent retries by the services are likely to produce errors until the application is fixed. Service Bus lets you identify application errors, giving you the option of preventing retries on application errors when your transport is used.

This section describes how to catch application errors in your transport and configure your transport to prevent application error retries.

Identifying Application Errors

In your transport provider code you must identify the conditions under which an application error occurs. For example, in the HTTP transport, an application error is one in which the HTTP response has a 500 status code, has a non-empty payload, and has a content type that is consistent with the service type, such as XML for SOAP. When an error meets the application error conditions you define, return a `TRANSPORT_ERROR_APPLICATION` type using one of the following methods:

- **Errors in the request:** Throw a `TransportException` with the error code `TRANSPORT_ERROR_APPLICATION` in `TransportProvider.sendMessageAsync()`.
- **Errors in the response:** Schedule `TransportSendListener.onError()` with the error code `TRANSPORT_ERROR_APPLICATION`.

The Transport SDK can then identify application errors when they occur and handle them accordingly. The Transport SDK also sends application errors to the pipeline `$fault` variable.

Configuring Application Error Retries

In your `<Transport>Config.xml` file, enter the following element as a child of the `<ProviderConfiguration>` element, according to the `TransportCommon.xsd` schema located in `/OSB_ORACLE_HOME/lib/modules/oracle.servicebus.kernel-api.jar`:

```
<declare-application-errors>true</declare-application-errors>
```

This entry provides an option to retry application errors on the business service's main transport configuration page when a user selects your transport. If you do not provide this element, the default value is `false`, application errors are not caught, and no option is provided to retry application errors.

Catching Connection Errors

Service Bus lets you identify connection errors in your transport, which trigger the Transport SDK to take inaccessible endpoint URIs offline automatically. For example, in a cluster with Service Bus running on Managed Servers, a Managed Server that experiences a connection error on a service request can automatically mark the endpoint URI as offline.

You can re-enable endpoint URIs in the following ways:

- On configuring the business service, you can set a retry count and retry iteration interval to determine the frequency and number of retries after connection errors.

A successful connection to the service after a retry automatically reactivates the endpoint URI.

A retry iteration interval of zero (0) takes the endpoint URI offline indefinitely and requires you to manually re-enable the endpoint URI.

- You can manually re-enable offline endpoint URIs from Fusion Middleware Control on the Dashboard for the business service.

The automated connection error functionality does not apply to the following situations:

- If a service pipeline dynamically sets an endpoint URI in `$outbound/ctx:transport/ctx:uri`, the Transport SDK cannot take the URI offline, because the endpoint URI is not defined in the service configuration.
- The Transport SDK does not persist connection status. After a server restart, all endpoint URIs are considered online.

For more information, see "Managing and Monitoring Endpoint URIs for Business Services" in *Administering Oracle Service Bus*.

Identifying Connection Errors

Once caught, a connection error triggers the Transport SDK to take the affected endpoint URI offline automatically. In your transport provider code, you must identify the conditions under which a connection error occurs. For example, in the Service Bus HTTP transport, a connection error is one in which the HTTP response has a 404 status code or there is an `IOException` when a connection is attempted on the endpoint URI.

When an exception meets the connection error conditions you define, return a `TRANSPORT_ERROR_CONNECTION` type using one of the following methods:

- **Errors in the request:** Throw a `TransportException` with the error code `TRANSPORT_ERROR_CONNECTION` in `TransportProvider.sendMessageAsync()`.
- **Errors in the response:** Schedule `TransportSendListener.onError()` with the error code `TRANSPORT_ERROR_CONNECTION`.

The Transport SDK can then identify connection errors when they occur and handle them accordingly. The Transport SDK also sends connection errors to the pipeline `$fault` variable and adds them to the response.

Defining Custom Environment Value Types

You can define custom environment value types to use in your transport implementation. When you use the `TransportProvider.getEnvValues()` method to return environment values for an endpoint, you can declare environment values of these custom types.

When your transport is used, custom environment value types can be used in the same ways that standard environment value types are used in Service Bus, such as for customization, find and replace, and preservation of values on configuration import. For example, you may want to be able to define and preserve references to a service account or a JMS queue in your transport configuration. Environment value types can be any of the following categories: environment, operational, and security.

Add custom variables to your `<Transport>Config.xml` file. The schema that determines the XML structure is `TransportCommon.xsd`, located in `located in /OSB_ORACLE_HOME/lib/servicebus-schemas.jar`.

Following is an example of a custom security variable in the JMS transport's `JmsConfig.xml`:

```
<env-value-type>
  <name>JMS Service Accounts</name>
  <localized-display-name>
    <localizer-class>com.bea.wli.sb.transports.messages.
      TransportsTextLocalizer</localizer-class>
    <message-id>JMS_SERVICE_ACCOUNTS</message-id>
  </localized-display-name>
  <localized-description>
    <localizer-class>com.bea.wli.sb.transports.messages.
      TransportsTextLocalizer</localizer-class>
    <message-id>JMS_SERVICE_ACCOUNTS</message-id>
  </localized-description>
  <simple-value>true</simple-value>
  <category>security</category>
</env-value-type>
```

Following are descriptions of key elements for custom environment value types:

- **name:** The variable name used by the Transport SDK.
- **display-name:** The name for the variable that appears in the Service Bus user interface. Following is the localization alternative:
 - **localized-display-name:** Alternative, localized version of display-name.
 - **localizer-class:** The localization properties text file containing the localized display-name. The `.properties` extension is not required.
 - **message-id:** The property in the localization properties file that provides the value of the display name.
- **description:** Description of the environment value type. For localization, use the `localized-description` element instead with the `localizer-class` and `message-id` child elements as described in `display-name`.
- **simple-value:** If the environment value type is of the category "environment," `simple-value` determines whether or not this type is searchable with find and replace functionality in Service Bus (value of true or false).
- **category:** The category of the environment value type: environment, security, or operational. When the category is security or operational, you can decide whether or not to preserve the environment value type during configuration import. When the category is environment, the environment value type is available for find and replace.

Publishing Proxy Services to a UDDI Registry

Universal Description, Discovery, and Integration (UDDI) is a standard mechanism for describing and locating web services across the internet. You might want to publish proxy services based on a custom transport provider to a UDDI registry. This allows proxy services to be imported into another Service Bus server in a different domain from the one hosting the business service.

To publish a proxy service, the transport provider needs to define a tModel that represents the transport type in the UDDI section of `TransportProviderConfiguration` XML schema definition. This tModel must contain a `CategoryBag` with a `keyedReference` whose `tModelKey` is set to the UDDI Types Category System and whose `keyValue` is **transport**. You are required to provide only the UDDI V3 tModel key for this tModel. If UDDI already defines a tModel for this transport type, you should copy and paste the definition into the UDDI section. For more information on the schema-generated interfaces, see [Schema-Generated Interfaces](#).

The following example provides an example of such a tModel.

Example tModel

```
<?xml version="1.0" encoding="UTF-8"?>
<ProviderConfiguration xmlns="http://www.bea.com/wli/sb/transport">
  . . .
  . . .
  <UDDI>
    <TModelDefinition>
      <tModel tModelKey="uddi:bea.uddi.org:transport:socket">
        <name>uddi-org:socket</name>
        <description>Socket transport based webservice</description>
        <overviewDoc>
          <overviewURL useType="text">
            http://www.bea.com/wli/sb/UDDIMapping#socket
          </overviewURL>
        </overviewDoc>
        <categoryBag>
          <keyedReference keyName="uddi-org:types:transport"
            keyValue="transport"
            tModelKey="uddi:uddi.org:categorization:types"/>
        </categoryBag>
      </tModel>
    </TModelDefinition>
  </UDDI>
</ProviderConfiguration>
```

If UDDI does not already define a tModel for this transport type, Service Bus can publish the tModel you define here to configured registries. When a UDDI registry is configured for Service Bus, the **Load tModels into Registry** option can be specified. That option causes all of the tModels of Service Bus, including the tModels for custom transport providers, to be published to the UDDI registry. After deploying your transport provider, you can update your UDDI registry configuration to publish your tModel.

During UDDI export,

`TransportProvider.getBusinessServicePropertiesForProxy(Ref)` is called and the resulting map is published to the UDDI registry. The provider is responsible for making sure to preserve all important properties of the business service in the map so that during the UDDI import process the business service definition can be correctly constructed without loss of information.

During UDDI import,

`TransportProvider.getProviderSpecificConfiguration(Map)` is called and the result is an `XmlObject` that conforms to the provider-specific endpoint configuration schema, which goes into the service definition.

When to Implement TransportWLSArtifactDeployer

Two sets of transport provider interfaces are provided that deal with individual service registration. **TransportProvider** has methods like create, update, delete, suspend, and resume, and **TransportWLSArtifactDeployer** has the same methods. This section discusses these two implementations and explains when you need to implement **TransportWLSArtifactDeployer**.

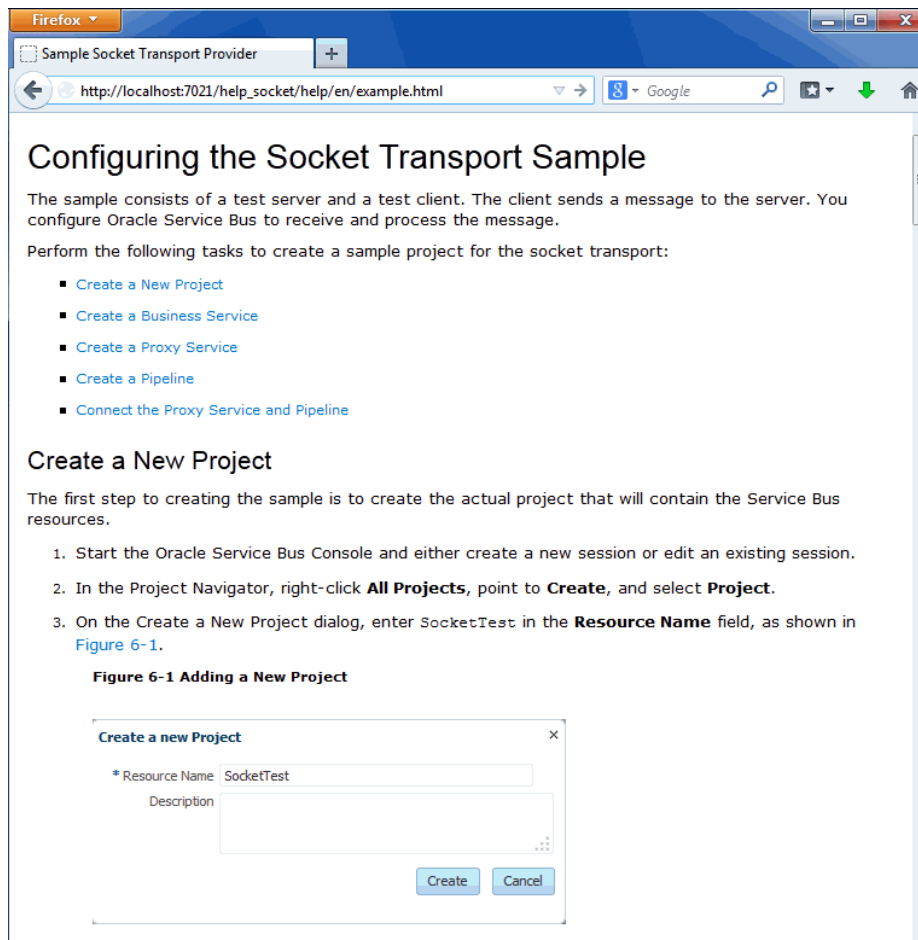
Only implement **TransportWLSArtifactDeployer** if your provider needs to make changes to Oracle WebLogic Server artifacts in the Service Bus domain. The methods in **TransportWLSArtifactDeployer** are only called on an Admin Server. In this case, changes are made through the **DomainMBean** argument that is passed in, and then the changes are propagated to the entire cluster.

The **TransportProvider** methods are called on all servers (Administration and Managed Servers) in the domain. Because you cannot make changes to Service Bus domain artifacts on a Managed Server, the purpose of the method calls on **TransportProvider** is to update its internal data structures only.

When a given Transport provider implements the **TransportWLSArtifactDeployer** interface, the methods on **TransportWLSArtifactDeployer** are called before the corresponding methods on **TransportProvider**. For example, **TransportWLSArtifactDeployer.onCreate()** is called before **TransportProvider.createEndPoint()**. For more information about **TransportWLSArtifactDeployer**, see [Summary of General Interfaces](#).

Creating Help for Custom Transports

You can provide online help for your custom transports for both JDeveloper and Oracle Service Bus Console. Both provide their own integrated help system. In JDeveloper, you need to incorporate the help into the existing help system. The Oracle Service Bus Console displays custom transport help stand-alone in its own browser window, as shown in [Figure 40-5](#). Custom transport help is displayed when you click the Help icon on the transport configuration page of the service definition editor. Providing help is optional, but it can be extremely useful in guiding service creators through the transport configuration process.

Figure 40-5 Custom Transport Help from the Service Bus Console

About Custom Transport Online Help

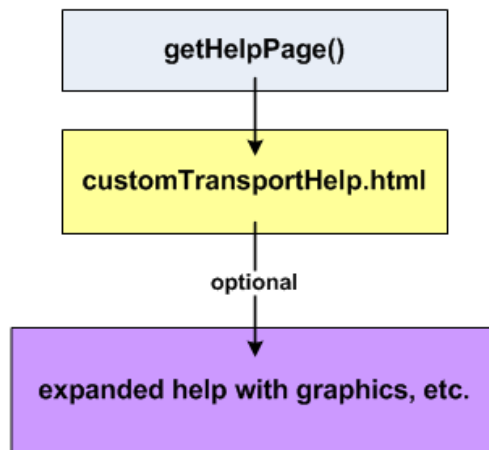
You have a lot of flexibility in deciding what type of help content to provide, from a simple page of text with no graphics to multiple pages with many graphics, PDF files, embedded video and so on. For example, you could create a single HTML file and reference it from the help link; or you could create separate help files that describe the transport configuration fields for business services and proxy services and also provide a high-level overview. You can create separate help topics for Oracle Service Bus Console and JDeveloper or use the same ones.

Service Bus provides a sample help implementation in its sample socket transport, located at `OSB_ORACLE_HOME/samples/servicebus/sample-transport`. The sample transport is a good reference implementation for developing your own custom transports and help.

How to Provide Custom Transport Help in the Console

This section shows you how to provide help for your custom transport at runtime in the Oracle Service Bus Console. Service Bus displays custom transport help as a stand-alone help page in a browser, as shown in Figure 40-5.

Figure 40-6 provides a high-level view of the console help framework for custom transports.

Figure 40-6 Oracle Service Bus Console Help Framework

By implementing a specific Service Bus interface, you use the `getHelpPage()` method to launch a single HTML page when the user clicks the **Help** icon for the custom transport configuration page in the console. The HTML file can contain the following:

- Text, inline CSS definitions, inline JavaScript functions
- References to graphics and other resources, as long as those resources are hosted in a web application or an external web site

In most situations, you should be able to provide all the help for your custom transport with text and inline formatting.

However, if you want to provide full-featured web-based help that includes graphics and other external resources, those resources must be hosted in a web application or an external web site. You must either reference those external resources in the HTML file or provide a link from the HTML file to an external location. For example, the sample socket transport help provides a link from the starting HTML file to a help topic with graphics that is running in a custom web application. Using an embedded JavaScript call, you could also set up your HTML file to automatically redirect to the expanded help URL you want.

Perform the following tasks to implement online help in the console:

- [Implement the CustomHelpProvider Interface](#)
- [Create an HTML File to Launch](#)
- [Create a Simple Web Application to Display Expanded Help \(Optional\)](#)

When you are done creating your help files, package the files as described in [Packaging Help for the Transport Plug-in](#).

Implement the CustomHelpProvider Interface

To develop the configuration user interface for your custom transport, you implement the `TransportUIBinding` interface in a custom class. To provide help for your transport configuration user interface in the Oracle Service Bus Console, you must also implement the `CustomHelpProvider` interface. `CustomHelpProvider` contains the `getHelpPage()` method you need to launch help for your transport configuration page in the Oracle Service Bus Console.

The sample socket transport implements `CustomHelpProvider` in its `SocketTransportUIBinding` class (located at `OSB_ORACLE_HOME/samples/servicebus/sample-transport/src/com/beanlsb/transports/sock`).

The following example contains snippets that illustrate the implementation of `CustomHelpProvider`.

Example - Implementing `CustomHelpProvider` to provide console online help

```
public class SocketTransportUIBinding
    implements TransportUIBinding, CustomHelpProvider {
    ...
    public Reader getHelpPage() {
        String helpFile = "help/en/contextsocketTransport.html";
        ClassLoader clLoader = Thread.currentThread().getContextClassLoader();
        InputStream is = clLoader.getResourceAsStream(helpFile);
        InputStreamReader helpReader = null;
        if(is!=null)
            helpReader = new InputStreamReader(is);
        else
            SocketTransportUtil.logger
                .warning(SocketTransportUtil.formatText(uiContext.getLocale(), "800138"));
        return helpReader;
    }
}
```

In the previous example, `getHelpPage()` returns a `Reader` stream that the Oracle Service Bus Console uses to send the HTML page to the browser. The `helpFile` path is relative to the root within the transport JAR.

If you are providing help in multiple languages, use `TransportUIContext.getLocale()` to help provide the appropriate path to the localized content; in this case, providing the locale value for `/help/<locale>/<localized>.html`.

Create an HTML File to Launch

You can create an HTML file for the `getHelpPage()` method to launch, as illustrated by `help/en/contextsocketTransport.html` in the example in [Implement the CustomHelpProvider Interface](#). If you want to keep your help implementation simple, create the HTML file to use text, inline CSS definitions, and inline JavaScript functions. If you do this, you do not need to create a separate web application to host graphics or other external resources.

However, if you want to provide more expanded help with graphics and other resources, reference those external resources in your HTML file, such as

```
img src="/help_socket/help/en/wwimages/addProject.gif"
```

or

```
a href="http://www.yoursite.com"
```

You can also set the HTML file up to automatically redirect to the expanded help with an embedded JavaScript call, as shown in the following example, which redirects from the sample socket transport HTML page to the expanded `help_socket` web application help content.

Example - JavaScript function that provides a redirect

```
<script language="JavaScript" type="text/javascript">
<!-- Begin
```

```

window.location="/help_socket/help/en/example.html";
// End -->
</script>

```

The sample socket transport HTML file provides a link to its expanded help. The HTML file, `contexts_socketTransport.html`, is located at `OSB_ORACLE_HOME/samples/servicebus/sample-transport/resources/help/en/`.

Create a Simple Web Application to Display Expanded Help (Optional)

If you want to go beyond a basic text HTML file for your transport help, you can provide expanded help with graphics and other resources in various ways:

- Link from the self-contained HTML file to an existing URL; for example, if you have an existing web site that contains your transport documentation. All that is required is that you provide a link to the URL from the self-contained HTML file. You can also insert references to graphics and other resources hosted on an external site.
- Create a web application for the expanded help, bundle it with your transport, and link to it or reference graphics and other resources from the HTML file. This topic provides instructions on creating a web application that is bundled in your transport EAR to display your expanded transport help.

Create the files described in [META-INF/application.xml](#) and [WEB-INF/web.xml](#) for your web application.

META-INF/application.xml

In `application.xml`, give your web application a context root that is used for the web application's root URL. The following example shows the context root for the sample socket transport web application.

Example - application.xml for the sample socket transport Web application

```

<application>
  <display-name>Socket Transport</display-name>
  <description>Socket Transport</description>
  <module>
    <web>
      <web-uri>webapp</web-uri>
      <context-root>help_socket</context-root>
    </web>
  </module>
</application>

```

The sample socket transport `application.xml` file is located at `OSB_ORACLE_HOME/samples/servicebus/sample-transport/META-INF/`.

This entry maps the file system directory `/webapp` to an alias web application root URL:

```
http://server:port/help_socket/
```

With your help files inside the web application in a directory such as `/help/en/`, the full URL to your expanded help would be:

```
http://server:port/help_socket/help/en/index.html
```

But your internal links to it only need to be

/help_socket/help/en/index.html

where index.html is the landing HTML page.

WEB-INF/web.xml

In web.xml, enter a display name and description for the web application. This is standard deployment descriptor information. The following example shows the name and description of the sample socket transport web application.

Example - web.xml for the sample socket transport Web application

```
<web-app>
  <display-name>Sample Socket Transport Help WebApp</display-name>
  <description>
    This webapp implements the help webapp for the socket transport.
  </description>
</web-app>
```

The sample socket transport web.xml file is located at `OSB_ORACLE_HOME/samples/servicebus/sample-transport/webapp/WEB-INF/`.

Help Content and Resources

Create and package your expanded help files inside the web application directory. In the sample socket transport, the help files are stored in `OSB_ORACLE_HOME/samples/servicebus/sample-transport/resources/help/en`.

Note:

The reason the socket transport help files are not stored in the `/webapp` directory is because the help directory contains help files and resources for both JDeveloper and the Oracle Service Bus Console. When the sample socket ANT build creates the transport JAR and transport EAR, it packages the help in different ways. For the transport EAR build, it moves the help files under the `/webapp` directory.

How to Provide Custom Transport Help in JDeveloper

If you make your custom transport available for service configuration in JDeveloper, you can incorporate a help topic for the page that appears in the service definition editor. You do this by adding your help files to the existing help JAR file.

To provide online help in JDeveloper:

1. Create your help topics. For more information, see [About Custom Transport Online Help](#) and [How to Provide Custom Transport Help in the Console](#).
2. Package all the help files. See [Packaging Help for the Transport Plug-in](#).
3. Navigate to `MW_HOME/osb/plugins/jdeveloper/doc/studio_doc`.
4. Make a backup copy of `osbjh.jar`.
5. Extract the files from `osbjh.jar` into a temporary directory.

6. Copy the sample transport help files you created to the temporary directory, maintaining the structure of your help file system. You can copy your files into a subdirectory or at the same level as the existing `f1*.html` files.
7. In the temporary directory, open `map.xml`, and add a `mapID` entry for each help topic to make available.

The map ID includes the target (ID) and URL (relative path and file name) for each help topic, along with a description. Below is an example for the sample sockets transport provider.

```
<mapID target="contexts_socketTransport" url="example.html" />
<!-- "Sample Sockets Transport Configuration"-->
```

8. Save and close the map file, and then JAR all the help files into a new `osbjh.jar` file.
9. Replace the old `osbjh.jar` file in `MW_HOME/osb/plugins/jdeveloper/doc/studio_doc.` with the new `osbjh.jar` file you just created.

Packaging Help for the Transport Plug-in

Your transport plug-in should contain the following:

- A transport JAR file containing your transport classes and supporting files. The JAR file includes help resources in a `/help/en` directory.
- A transport EAR file that contains your runtime components, including the help system in a `/webapp/help/en` directory.

Notice that with the `/en` directory the help is packaged to support localization. To provide localization, you must create a plug-in for each locale. For more information about packaging the transport and help, see [Packaging and Deploying a Custom Transport Provider](#).

Developing Custom Transport Providers for JDeveloper

This chapter describes the best practices, design considerations, and packaging to develop custom Service Bus transports for use in JDeveloper. The Transport SDK interface provides a bridge between transport protocols and the Service Bus runtime.

Tip:

Before you begin this chapter, review [Learning About Custom Transport Providers](#).

This chapter includes the following sections:

- [Introduction](#)
- [Services Runtime and Services Configuration](#)
- [Packaging Transports for JDeveloper](#)
- [Custom Transport Provider Reference for Offline Tools](#)

Introduction

Service Bus transports were originally designed to be deployed on Service Bus servers and configured through the Oracle Service Bus Console. With design environments like JDeveloper, some modifications to the SDK are necessary to ensure transport design-time features can be used on platforms other than the Oracle Service Bus Console.

The sample socket transport installed along with Service Bus was ported to JDeveloper and can be considered a best practice for JDeveloper integration. The sample socket resources are located at `OSB_ORACLE_HOME/samples/servicebus/sample-transport`. The Java source files are in the `/src` subdirectory. The sample also contains a build script that automatically packages the sample socket transport for both JDeveloper integration and Oracle Service Bus Console deployment. For information on building and deploying the sample socket transport, see [Creating a Sample Socket Transport Provider](#).

Services Runtime and Services Configuration

When you develop a transport, distinguish the runtime aspects from the configuration aspects. The runtime aspects include proxy or business service deployment and service runtime invocation. The configuration aspects include proxy or service configuration and validation. The runtime aspects do not need to change since they are always exercised in the context of a running Service Bus server. However, the configuration aspects are dependent on the design environment.

Developers should consider three different deployment modes:

1. **Online mode:** The services using the custom transport are configured with the Oracle Service Bus Console on a running Service Bus server.
2. **Offline mode:** The services using the custom transport are configured with a design environment running outside the Service Bus server. No remote server is available.
3. **Offline mode with remote server:** The services using the custom transport are configured with a design environment running outside the Service Bus server. However, a remote server is available and can be used for both validation and configuration purposes.

Transports running in JDeveloper must support offline mode and, optionally, offline mode with a remote server.

Offline Methods

When you deploy a transport in offline mode, the configuration framework creates a single session for all the resource configurations. This session is never activated. Since proxy or business services can only be deployed on a running Service Bus server, there is no need to activate the session. However, it is still important to detect conflicts and configuration errors, and the validation methods are still exercised.

Following is a list of the minimum set of classes and methods defined by the Transport SDK that must be implemented in offline mode. The exceptions were removed from the methods signature for better readability.

Note:

You do not need to completely re-implement your transport for offline mode. In most cases your transport will only need a few changes to existing methods to support both online and offline modes.

Classes and Methods You Must Implement for Offline Mode

- `public interface TransportProvider`, specifically the following methods:
 - `String getId()`
 - `void validateEndPointConfiguration(TransportValidationContext context)`
 - `SchemaType getEndPointConfigurationSchemaType()`
 - `SchemaType getRequestMetaDataSchemaType()`
 - `SchemaType getRequestHeadersSchemaType()`
 - `SchemaType getResponseMetaDataSchemaType()`
 - `SchemaType getResponseHeadersSchemaType()`
 - `TransportProviderConfiguration getProviderConfiguration()`
 - `TransportUIBinding getUIBinding(TransportUIContext context)`

- void shutdown()
- Collection<NonQualifiedEnvValue> getEnvValues(Ref ref, EndPointConfiguration epConfig)
- void setEnvValues(Ref ref, EndPointConfiguration epConfig, Collection<NonQualifiedEnvValue> envValues)
- Collection<Ref> getExternalReferences(EndPointConfiguration epConfig)
- void setExternalReferences(Map<Ref, Ref> mapRefs, EndPointConfiguration epConfig)
- Map<String, String> getBusinessServicePropertiesForProxy(Ref ref)
- XmlObject getProviderSpecificConfiguration(Ref ref, Map<String, String> props)
- public interface TransportProviderFactory
This interface registers transports in offline mode. For more information, see [Packaging and Deploying a Custom Transport Provider](#).
- public interface TransportUIBinding
Implement all the methods in this interface and define the user interface used to configure a proxy or business service.

Helper Classes

- public class TransportManagerHelper
This class, which is typically used by TransportProvider developers, provides a boolean `isOffline()` method to help the provider implementor determine whether the code is running offline or not. Some of the methods that are not valid in offline mode will throw exceptions, which are described below. Other methods are meant only for runtime or deployment, such as `public isAdmin()`.

The following methods are also available when working in offline mode with remote server:

- public Set<String> getDispatchPolicies(JMXConnector connector)
- public DomainRuntimeServiceMBean getDomainRuntimeServiceMBean(JMXConnector connector)
See [Working Offline with a Remote Server](#) for more information.

Do not invoke the following methods in offline mode:

- public static boolean isAdmin()
This method throws a `java.lang.IllegalStateException` message.
- public static boolean clusterExists()
This method always returns false.

Note:

The preferred method for checking runtime status is to use `isRuntimeEnabled()` in conjunction with `getRuntimeServers()`.

Restrictions when Working Offline

When you work offline, none of WebLogic Server services running on the server are available. Do not use these services inside the methods described in [Offline Methods](#).

Following are examples of restrictions for working offline:

- The Oracle WebLogic Server MBeans are not available.
- The server Java properties are not available.
- You cannot access the JNDI tree directly. However, if JNDI properties are defined in the service configuration, you can attempt to use them.
- You can not determine if the service is going to run in a cluster or a standalone server.
- You do not have access to the Oracle WebLogic Server security infrastructure.
- You do not have access to any static singleton service located on the server.

Because some of the services are not available, it is necessary to evaluate how the transport user interface is affected. In general, the user interface should be more flexible to let users manually configure values instead of trying to retrieve values from the server environment.

For example, some transports retrieve the list of available Work Manager (dispatch policy) items by using the `TransportManagerHelper` and letting the user pick one through a list. However, in offline mode, the MBeans are not available so the list cannot be populated. The transport provider has two choices:

1. Let the user type the correct Work Manager name. In that case, the user interface must be changed to be a text box and not a list when working offline.
2. Another less flexible option is to populate the list with just the default Work Manager. When the service is pushed to a running Service Bus server, the Work Manager name can be switched using an environment value substitution.

Working Offline with a Remote Server

When you work offline, a remote server might be available. For instance, when you configure a service in JDeveloper, you can associate a remote Service Bus server to the current project. The transport provider can take advantage of the remote server by accessing the Oracle WebLogic Server MBeans and retrieving information. This mode is similar to working online; however, some restrictions still apply since the code is not running on the server and only the MBeans are available.

When you work offline with a remote server, the following restrictions apply:

- The server Java properties are not available.
- You cannot use many of `TransportManagerHelper` methods as described in [Offline Methods](#).

- You cannot access the JNDI tree directly. However if JNDI properties are defined in the service configuration, you can attempt to use them.
- You do not have access to any static singleton service located on the server.

To access the MBeans, the framework provides an instance of `JMXConnector` when it requests the `TransportUI` object, or when it asks the provider to validate a configuration. The `JMXConnector` is available in the `TransportUIContext` or the `TransportValidationContext`:

```
JMXConnector connector =  
(JMXConnector)uiContext.get(TransportValidationContext.JMXCONNECTOR);
```

For more information, see the sample transport in [Custom Transport Provider Reference for Offline Tools](#).

If the connector is not present, a remote server is not available. This connector object can then be used to access the MBeans. Helper methods have been added to the `TransportManagerHelper` to retrieve the list of `WorkManager` and `WebLogic Server domain MBean`.

Note:

This behavior is generalized for both online and offline modes. The public static `Set<String> getDispatchPolicies()` method defined in the `TransactionManagerHelper` will be deprecated and must be replaced by the same method with `JMXConnector` as a parameter. If you do not replace it, the following error appears:

```
com.bea.wli.sb.transports.TransportException.
```

Bootstrapping Transports in Offline Mode

In online mode, transports must be packaged as EAR files and deployed on a Service Bus server. When the EAR is loaded at startup, the transport registers a callback on a startup event and registers an instance of the `TransportProvider` to the `TransportManager`.

In offline mode, the SDK provides an interface called `com.bea.wli.sb.transports.TransportProviderFactory` that registers transports. A transport developer must implement this interface and must make the default constructor public. The interface is provided in [Custom Transport Provider Reference for Offline Tools](#), as well as a sample implementation.

If the `TransportProvideFactory` is instantiated, you can assume the transport needs to work in offline mode (with or without a remote server).

Note:

You can set a boolean operator in the `TransportManagerHelper` when the constructor is invoked to determine if the transport is running in offline mode. This information can also be passed in the `TransportUIContext` and the `TransportValidationContext`. Your engineering department can assist you in making this decision.

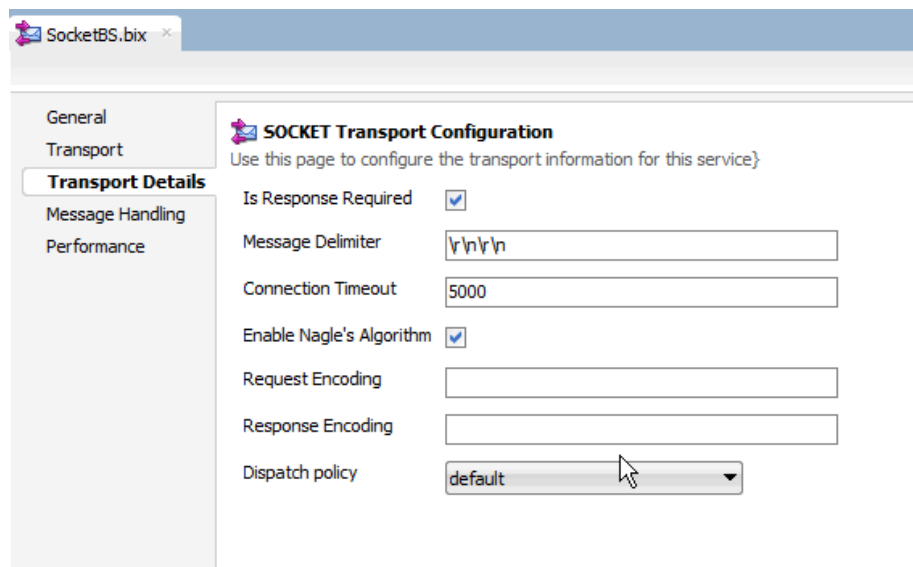
Packaging Transports for JDeveloper

In order to use your custom transport provider in JDeveloper, you must add the JAR file you generated when you created the transport provider to your Service Bus installation. Packaging your custom transport as a JDeveloper plug-in, in conjunction with your transport user interface implementation, lets service developers select and configure your transport in the development environment.

In offline mode, you can use transports in different design environments, including JDeveloper. In general, transports simply need to be available to external design time environments as a self-contained JAR file. A self-contained JAR file includes the transport `config.xml` file, the header, metadata schemas, XBeans classes, `TransportProviderFactory` implementation, and the compiled transport classes.

Figure 41-1 shows the service editor in JDeveloper—after a service has been created—with a configuration page for the sample socket transport.

Figure 41-1 Transport Configuration Page in JDeveloper



For information about packaging and deploying the custom transport provider, see [Packaging and Deploying a Custom Transport Provider](#).

Note:

Your implementation of the `TransportUIBinding` interface determines the user interface for selecting and configuring your transport, both in JDeveloper and in the Oracle Service Bus Console.

Custom Transport Provider Reference for Offline Tools

This section provides reference information that you might need to know when developing a custom transport provider for offline use in JDeveloper.

- [Working in Different Modes](#)
- [TransportProviderFactory](#)

- [TransportManagerHelper Methods](#)

Working in Different Modes

Dispatch policies are used by most transports and allow service throttling. This code distinguishes the three modes described in [Services Runtime and Services Configuration](#):

- Online mode
- Offline mode
- Offline mode with remote server

The connection to the remote server is retrieved from the context, as shown in the following example.

Example - Connection to the Remote Server

```
/**
 * Builds the dispatch policies in the ui object.
 *
 * @param curDispatchPolicy
 * @return TransportEditField containing existing dispatch policies.
 */
public TransportEditField getDispatchPolicyEditField(String curDispatchPolicy {
    TransportUIFactory.TransportUIObject uiObject = null;
    Set<String> wmSet = null;

    if (SocketTransportManagerHelper.isOffline())
        { // if on JDeveloper try to get the MBeans from the UIContext
            JMXConnector connector =
                (JMXConnector)uiContext.get(TransportValidationContext.JMXCONNECTOR);
            if (connector != null) {
                try {
                    wmSet = TransportManagerHelper.getDispatchPolicies(connector);
                } catch (Exception ex) {
                    wmSet = null;
                }
            }
        } else { // if running on the server use the helper to get the policies
            try {
                wmSet = TransportManagerHelper.getDispatchPolicies();
            } catch (TransportException transexcept) {
                SocketTransportUtil.logger.error(SocketTransportMessagesLogger.noDispatchPolicies(),
                    transexcept);
            }
        }

    if (wmSet == null) // if JMXConnector not available or impossible to connect provide a simple
        edit field
        {
            uiObject = TransportUIFactory.createTextBox(curDispatchPolicy);
        } else // create a drop down list
        {
            // adding default work manager to the list.
            wmSet.add(DEFAULT_WORK_MANAGER);
            String[] values = wmSet.toArray(new String[wmSet.size()]);
            uiObject = TransportUIFactory.createSelectObject(values, values, curDispatchPolicy,
                TransportUIFactory.SelectObject.DISPLAY_LIST, false);
        }
    return TransportUIFactory.createEditField(DISPATCH_POLICY,
        TextMessages.getMessage(TextMessages.DISPATCH_POLICY, locale),
```

```
        TextMessages.getMessage(TextMessages.DISPATCH_POLICY_INFO, locale), uiObject);  
    }
```

TransportProviderFactory

`TransportProviderFactory` lets you provide design-time functionality in JDeveloper. It includes methods for registering the custom transport provider and retrieves the ID you define for the provider. For information about the methods provided in this interface, see the *Java API Reference for Oracle Service Bus*.

The following sample shows how the sample socket transport implements this interface.

Example - Example of the Socket Transport Implementing the Interface

```
package com.bea.alsb.transports.sock;  
  
import com.bea.wli.sb.transports.TransportManager;  
import com.bea.wli.sb.transports.TransportException;  
import com.bea.wli.sb.transports.TransportProviderFactory;  
  
public class SocketTransportProviderFactory implements TransportProviderFactory  
{  
  
    public void registerProvider(TransportManager tm) throws TransportException  
    {  
        SocketTransportProvider instance = SocketTransportProvider.getInstance();  
        tm.registerProvider(instance, null);  
    }  
  
    public String getId() {  
        return SocketTransportProvider.ID;  
    }  
}
```

TransportManagerHelper Methods

The `TransportManagerHelper` class provides methods to retrieve the Work Managers (dispatch policies), among others. For a complete list of methods provided in this interface, see the *Java API Reference for Oracle Service Bus*.

Packaging and Deploying a Custom Transport Provider

This chapter describes how to package and deploy a custom transport provider for use with Service Bus.

This chapter includes the following sections:

- [Packaging the Transport Provider](#)
- [Deploying the Transport Provider](#)
- [Undeploying a Transport Provider](#)
- [Deploying to a Cluster](#)

Packaging and Deployment Overview

Each custom transport provider requires two files: an EAR file and a JAR file. A third file, a transport plug-in, is required in order to use the custom transport provider in JDeveloper. You must package your custom transport provider as a self-contained JAR file, which defines the transport, and an EAR file, which can be deployed on the WebLogic Server. The EAR file can include the JAR file, or you can make the JAR file a library on which the EAR file depends. Using the latter method means you only need to maintain one copy of the JAR file.

To make the transport available to Service Bus, install the EAR file and, optionally, the JAR file, in `/MW_HOME/osb/lib/transport`s. Typically, both the EAR file and the JAR file are placed in this directory for Service Bus transports, but it is not required that the JAR file be placed there. The plug-in file you create, which makes the custom transport provider available to offline tools, points to the JAR file.

To make the transport available to the Oracle Service Bus Console and runtime, deploy the EAR file to the server with the Service Bus Kernel EAR file and other Service Bus related applications. The sample socket transport provider example illustrates how a transport provider is organized and deployed. For more information, see [Creating a Sample Socket Transport Provider](#).

Custom Transport Provider Components

Each transport provider consists of two distinct components:

- **Configuration:** The configuration part of a transport provider is used by the Oracle Service Bus Console to register endpoints with the transport provider. This configuration behavior is provided by the implementation of the user interfaces. For more information, see [User Interface Configuration](#).

- **Runtime:** The runtime part of a transport provider implements the business logic of sending and receiving messages.

A best practice is to package the transport provider so the configuration and runtime parts are placed in separate deployment units. This practice makes cluster deployment simpler. For more information, see [Deploying to a Cluster](#) and [Transport Provider Components](#).

Custom Transport Provider Resources

Your transport JAR file must include the following resources:

- A `MANIFEST.MF` file that contains key information about your transport plug-in. Use the sample socket transport `MANIFEST.MF` for reference.
- A `TransportConfig.xml` file, which configures the transport provider. Use the sample socket transport plug-in as a reference. See [Step 5. Define the TransportProviderConfiguration XMLBean](#).
- The compiled Java classes containing your transport implementation.
- The compiled XML bean generated classes.
- (Optional) Resources for providing online help.

Packaging the Transport Provider

This section describes the structure of the JAR and EAR files for your custom transport providers. To see an example of any of the files listed in this section, build the sample socket transport provider, as described in [Building and Deploying the Sample](#). You can then view all the packaged artifacts. You can also review the sample `build.xml` file to see an example of how to compile and deploy the custom transport provider.

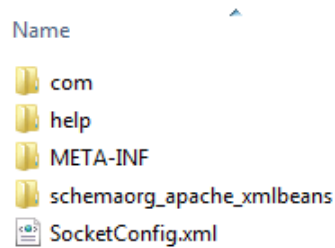
Transport JAR File Packaging

You package your transport provider as a JAR file, which makes the transport portable. Use the following guidance for packaging your transport provider:

- To construct the plug-in JAR, append "transport" to the name of the custom transport to create the JAR file name. For example, the sample socket transport JAR file is named `sock_transport.jar`.
- Package the file with the following directory structure, as illustrated in [Figure 42-1](#):
 - `/com` (transport classes and resources)
 - `/help`

If you are providing help for your transport, include a `/help` directory for your help resources, as described in [Creating Help for Custom Transports](#).
 - `/META-INF/MANIFEST.MF`
 - `/schemaorg_apache_xmlbeans` (XML bean classes and resources)
 - `TransportConfig.xml` (the XMLBean transport provider configuration; see [Step 5. Define the TransportProviderConfiguration XMLBean](#))

The following figure shows the sample socket transport provider JAR directory.

Figure 42-1 Plug-In Packaging

To see an example of plug-in packaging, build the sample socket transport, as described in [Creating a Sample Socket Transport Provider](#). View the generated `sock_transport.jar` and `sock_transport.ear`.

Transport EAR File Packaging

You package the runtime components of the custom transport provider in an EAR file, which can then be deployed to the WebLogic Server. This file can either contain the JAR file or can depend on the JAR file as a library. A typical packaging structure for the EAR file would include the following:

- `APP-INF/lib/name-transport.jar`
- `META-INF/MANIFEST.FM` and additional schema files
- Any additional web application files, such as packaged help files

Transport Plug-in Registration for JDeveloper

In order for JDeveloper to pick up the new transport, you need to create a plug-in file that describes the transport provider implementation, transport ID, help ID (if any), and additional libraries that are required for the transport. You do not need to create this file if you do not plan to use the transport in JDeveloper. The naming convention for a plug-in file is `transport-transport_name.xml`.

Use the following format to create the plug-in registration file:

```
<plugin xmlns="http://www.bea.com/alsb/offline/extensions">
  <transport
    class="transport_provider_class"
    id="transport_id"
    helpId="ID_to_access_help"/>
  <libraries>
    <library name='name_and_path_for_transport_jar' />
  </libraries>
</plugin>
```

The following example shows the sample plug-in file provided with the sample socket transport provider installed with Service Bus.

Example - Sample Socket Transport Provider Plug-in File

```
<plugin xmlns="http://www.bea.com/alsb/offline/extensions">
  <transport
    class="com.bea.alsb.transports.sock.SocketTransportProviderFactory"
    id="socket"
    helpId="contexts_socketTransport"/>
  <libraries>
    <library name='lib/transports/sock_transport.jar' />
  </libraries>
</plugin>
```

```
</libraries><
</plugin>
```

Transport Plug-in Installation

Once you create your transport provider EAR file, JAR file, and optional plug-in for JDeveloper, you need to add the files to the JDeveloper installation so they can be picked up by JDeveloper.

- Copy the generated EAR and JAR files to `/MW_HOME/osb/lib/transport`s.
- Copy the plug-in registration file to `/MW_HOME/osb/config/plugin`s.

Deploying the Transport Provider

After you create a deployable EAR file for your transport provider, you need to deploy it to the Service Bus domain. You can deploy the EAR file by whichever of the following methods you prefer:

- Programmatically (using WebLogic Deployment Manager JSR-88 API)
- Using the Oracle WebLogic Server Administration Console
- Adding an entry similar to the following example to the Service Bus domain `config.xml` file

Example - Application Deployment Entry

```
<app-deployment>
  <name>My Transport Provider</name>
  <target>AdminServer, myCluster</target>
  <module-type>ear</module-type>
  <source-path>${MW_HOME}/osb/lib/transport/mytransport.ear</source-path>
  <deployment-order>6</deployment-order>
</app-deployment>
```

Note:

The deployment order of your transport provider EAR file should be high enough so that the entire Service Bus Kernel EAR is deployed before the transport provider.

Transport Registration

On server restart, you need to ensure that your deployed transport can immediately begin to handle service requests. To ensure immediate transport availability, extend the `weblogic.application.ApplicationLifecycleListener` class and use the `preStart()` method to register your transport using `TransportManager.registerProvider()`.

For an example implementation, see the `ApplicationListener` class provided with the sample socket transport, located at `OSB_ORACLE_HOME/samples/servicebus/sample-transport/src/com/bea/alsb/transport/sock`. When extending `ApplicationLifecycleListener`, be sure to register your extending class in `META-INF/weblogic-application.xml`. The sample socket transport provides the following entry for its `ApplicationListener` class in `OSB_ORACLE_HOME/`

```

samples/servicebus/sample-transport/META-INF/weblogic-
application.xml:

<weblogic-application>
  <listener>
    <!-- This class gives callbacks for the deployment lifecycle and socket
         transport is registered with ALSB whenever the application is started.
    -->
    <listener-class>com.bea.alSB.transports.sock.ApplicationListener
  </listener-class>
</listener>
</weblogic-application>

```

Undeploying a Transport Provider

Once a transport provider has been registered with Service Bus, the undeployment or unregistration of the transport provider is not supported.

Deploying to a Cluster

Your transport provider must be deployed on all the servers and clusters where Service Bus is deployed. This means that if Service Bus is deployed only on the Admin Server (which it always is), you must deploy the transport provider on the Admin Server. If Service Bus is deployed in an admin + Managed Server topology, you must deploy the transport provider on the Admin Server and that particular Managed Server. If Service Bus is deployed in a cluster, you must deploy your transport provider on the Admin Server and the cluster. Note that Service Bus is always deployed on the Admin Server regardless of the domain topology.

The application code inside your transport provider EAR file needs to be aware dynamically of where the transport is being deployed (such as the Admin Server or a Managed Server) and exhibit only configuration behavior on the Admin Server and only runtime behavior on the Managed Server.

For example, in the initialization pseudo code in `some_transport.ear`, you can use this logic to decide whether or not to activate the configuration or runtime portion of the provider:

```

protected SomeTransportProvider() throws TransportException {
    . . . some other initialization code . . .
    if (!isRuntimeEnabled)
        _engine = new RuntimeEngine(. . .);
}

```

In this case, creating an instance of the `RuntimeEngine` class is runtime behavior and only needs to happen on a managed node in a multi-server domain or on the administration node in a single server domain.

Furthermore, in a cluster environment, `TransportProvider.createEndPoint()` and `deleteEndPoint()` are called on an Admin Server as well as Managed Servers in the cluster (with the exception of WLS HTTP router/front-end host). Some transport providers can choose not to do anything other than registering the fact that there is an endpoint with the given configuration, such as HTTP. In general the transport provider needs to examine whether `createEndPoint()` or `deleteEndPoint()` is called on the administration or Managed Server to decide the appropriate behavior.

Creating a Sample Socket Transport Provider

This chapter describes how to build and run the sample socket transport provider. This sample and its source code are installed with Service Bus. The sample serves as an example implementation of a custom transport provider.

This chapter includes the following sections:

- [Sample Socket Transport Provider Design](#)
- [Sample Location and Directory Structure](#)
- [Building and Deploying the Sample](#)
- [Creating a Socket Transport Sample Project](#)
- [Testing the Socket Transport Provider](#)

Sample Socket Transport Provider Design

The primary purpose of the sample socket transport provider is to serve as an example transport provider implementation. This publicly available sample demonstrates the implementation and configuration details of the Transport SDK.

Concepts Illustrated by the Sample

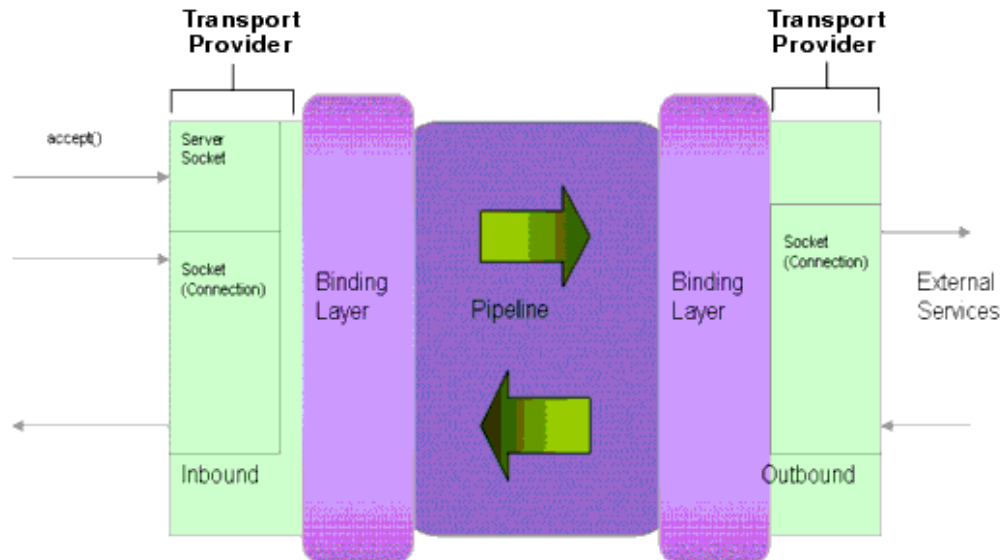
The sample transport is designed to send and receive streamed data to and from a configured TCP socket in Service Bus. The sample transport illustrates the following Transport SDK concepts:

- Implementing the set of Transport SDK APIs that are required to build a custom transport.
- Performing transport endpoint validations, such as checking that no socket endpoint is listening on the configured address.
- Implementing several UI configuration options, including socket properties and message patterns.
- Implementing a one-way or synchronous request-response message pattern.
- Using POJOs (Plain Old Java Objects) for metadata and headers of endpoint requests and responses.
- Using streaming in Service Bus pipelines.

Basic Architecture of the Sample

Figure 43-1 shows the basic architecture of the sample socket transport provider. Any client can connect to the server socket. Data is received at the server socket and passes through the pipeline. The response comes back through the outbound transport. The response is finally sent back to the inbound transport and back to the client.

Figure 43-1 Sample Socket Transport Architecture



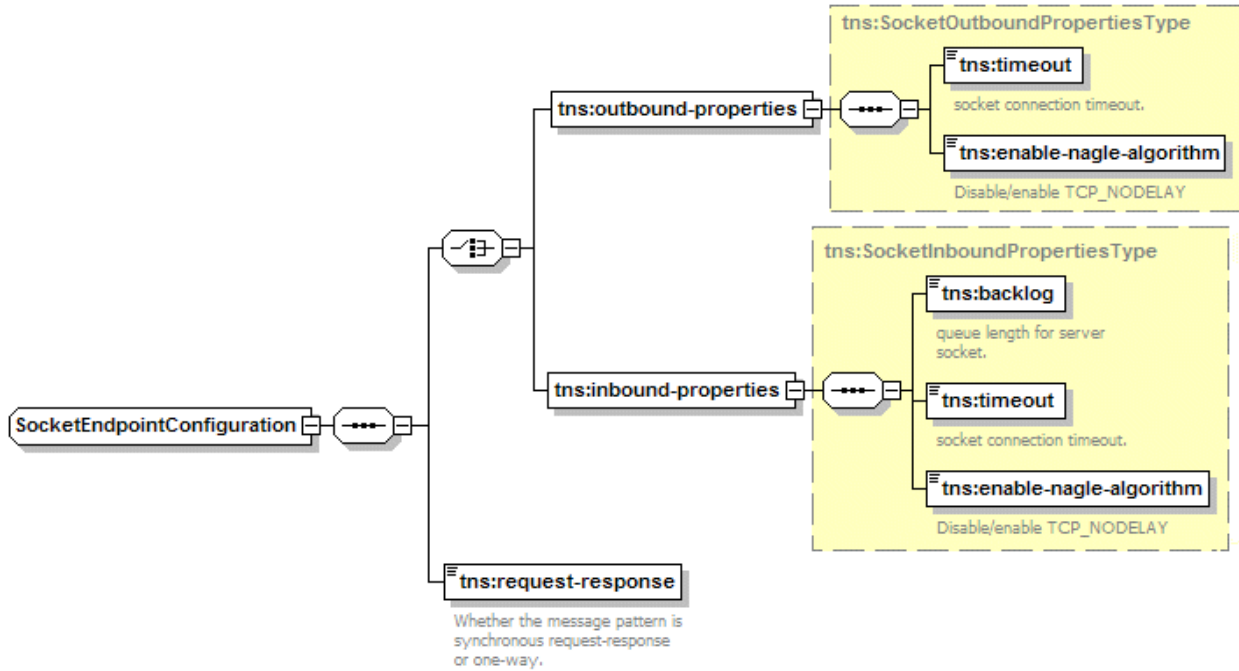
Configuration Properties

Figure 43-2 illustrates the configuration properties for the transport endpoint. These properties are configured in the schema file, `SocketTransport.xsd`. For the location of this file, see [Sample Location and Directory Structure](#) . This file allows you to extend the basic set of properties defined in the common schema provided with the SDK. Refer to the `SocketTransport.xsd` file for information on each of the properties.

Tip:

For more information about these configuration properties, see [Step 4. Define Transport-Specific Artifacts](#) .

Figure 43-2 SocketEndpointConfiguration Properties



Also in the `SocketTransport.xsd` file are the request and response header and metadata properties, as illustrated in Figure 43-3. Refer to the `SocketTransport.xsd` file for more information about these properties.

Figure 43-3 Request and Response Header and Metadata Configurations



Sample Location and Directory Structure

This section briefly describes some of the key folders in the sample project. You can use this directory structure as a model for developing your custom transport provider. The sample socket transport provider is installed with Service Bus and is located in the following directory: `OSB_ORACLE_HOME/samples/servicebus/sample-transport`.

Table 43-1 lists and briefly describes key `sample-transport` directories.

Table 43-1 Key Sample Transport Provider Directories

Directory	Description
build	This directory is created when you build the sample socket transport. It contains the built and packaged transport for use in Service Bus.
l10n	This directory contains these internationalization files. There is one of each file for each supported locale. <i>SocketTransportMessages.xml</i> : The configuration file for messages that are displayed on the Oracle Service Bus Console. <i>SocketTransportTextMessages.xml</i> : The configuration file for custom transport field names and their descriptions.
META-INF	This directory contains these application deployment descriptor files: <i>application.xml</i> : J2EE application descriptor file <i>weblogic-application.xml</i> : WebLogic application descriptor file
offline	This directory contains one file, <i>transport-socket.xml</i> , which specifies the fully-qualified class name and the name and location of its library JAR file.
resources	This directory contains the socket transport provider configuration file used by the Transport SDK, which is named <i>SocketConfig.xml</i> . It also includes sample help files for the transport.
schemas	This directory contains the relevant schemas defined for this transport, in this case, <i>SocketTransport.xsd</i> . This file describes the socket endpoint request and response metadata and headers.
src	This directory contains the source tree of the sample transport.
test	This directory includes a testing utility for the sample transport provider, along with the source tree for the test server and client.
webapp	This directory contains the deployment descriptors required for the sample transport help web application.

The following Ant build files are also located in the `sample-transport` directory:

- `build.properties` – Properties file for Ant.
- `build.xml` – An Ant build file with different targets for compile, build, stage, and deploy.

Building and Deploying the Sample

Perform the steps provided in this section in the order given to build and deploy the sample transport provider.

- [How to Set Up the Environment](#)
- [How to Build the Sample Transport Provider](#)
- [How to Deploy the Sample Transport Provider](#)
- [Registering the Sample Transport Provider With JDeveloper](#)

How to Set Up the Environment

A script is provided in the Service Bus domain to configure the environment for building the sample.

To set up the environment:

1. Create a new domain or use one of the preconfigured domains installed with Service Bus.
2. Set the domain environment by running the following script:

```
DOMAIN_HOME/bin/setDomainEnv.cmd (or setDomainEnv.sh on a UNIX system)
```

How to Build the Sample Transport Provider

Once you set the environment configuration, you can build the transport using the Ant build files provided in the `sample-transport` directory.

To build the sample transport provider:

1. In a command window, navigate to `OSB_ORACLE_HOME/samples/servicebus/sample-transport`.
2. Run the following command:

```
ant build
```

This command compiles the source files in `OSB_ORACLE_HOME/samples/servicebus/sample-transport/build`.

3. After the transport builds successfully, run the following command:

```
ant stage
```

This copies `sock_transport.ear` and `sock_transport.jar` to `OSB_ORACLE_HOME/lib/transport` and copies `transport-socket.xml` to `OSB_ORACLE_HOME/config/plugins`.

How to Deploy the Sample Transport Provider

The sockets sample also provides automated scripts for you to deploy the sample transport provider to the WebLogic Server. Once the sample is built and staged successfully, you can run the deploy command.

To deploy the sample transport provider:

1. Set the following variables in `sample-transport/build.properties`:

```
wls.hostname
```

```
wls.port
```

```
wls.username
```

```
wls.password
```

```
wls.server.name
```

2. Deploy the transport provider on the server by running the following command:

```
ant deploy
```

Registering the Sample Transport Provider With JDeveloper

If you want to be able to create or import projects in JDeveloper using the sample socket transport provider, you need to modify the transport registration file.

To register the sample transport provider with JDeveloper:

1. Navigate to `/MW_HOME/osb/config/plugins`.
2. Open `transport-socket.xml` in an XML or text editor.
3. In the `transport` element, add the following attributes:

```
id='socket'  
helpId="contexts_socketTransport"
```

The final file should look like this:

```
<plugin xmlns="http://www.bea.com/alsb/offline/extensions">  
  
  <transport  
    class="com.bea.alsb.transports.sock.SocketTransportProviderFactory"  
    id="socket"  
    helpId="contexts_socketTransport" />  
  
  <libraries>  
    <library name='lib/transports/sock_transport.jar' />  
  </libraries>  
  
</plugin>
```

4. Save and close the file.
5. Restart JDeveloper if it is running.

Creating a Socket Transport Sample Project

The sample consists of a test server and a test client. The client sends a message to the server. You configure Service Bus to receive and process the message.

Perform the tasks in this section in the order given. These instructions are for creating the sample project in the Oracle Service Bus Console, but you can do this in JDeveloper as well.

- [Creating the Project](#)
- [Creating the Business Service](#)
- [Creating the Proxy Service](#)
- [Creating the Pipeline](#)
- [Connecting the Proxy Service and Pipeline](#)

Creating the Project

The first step to creating the sample is to create the actual project that will contain the Service Bus resources.

To create the project:

1. Start the Oracle Service Bus Console and either create a new session or edit an existing one.
2. In the Project Navigator, right-click **All Projects**, point to **Create**, and select **Project**.
3. On the Create a New Project dialog, enter `SocketTest` in the **Resource Name** field.
4. Click **Create**.

The new project appears in the Project Definition Editor.

Creating the Business Service

In the sample project, you create a business service to talk to the server.

To create the business service:

1. In the Project Navigator, right-click the `SocketTest` project, point to **Create**, and select **Business Service**.

The Create Business Service wizard appears.

Figure 43-4 Create Business Service Wizard

2. In the **Resource Name** field, enter `SocketBS`.
3. In the **Transport** field, select `socket`. Click **Next**.

4. For the **Service Type**, select **Any XML Service**, and click **Next**.
5. In the Endpoint URI field, change the default URI to `tcp://localhost:7031`.
6. Click **Create**.

The business service appears in the Business Service Definition Editor. You can click the different subtabs to view the configuration.

7. Click **Save**.

Creating the Proxy Service

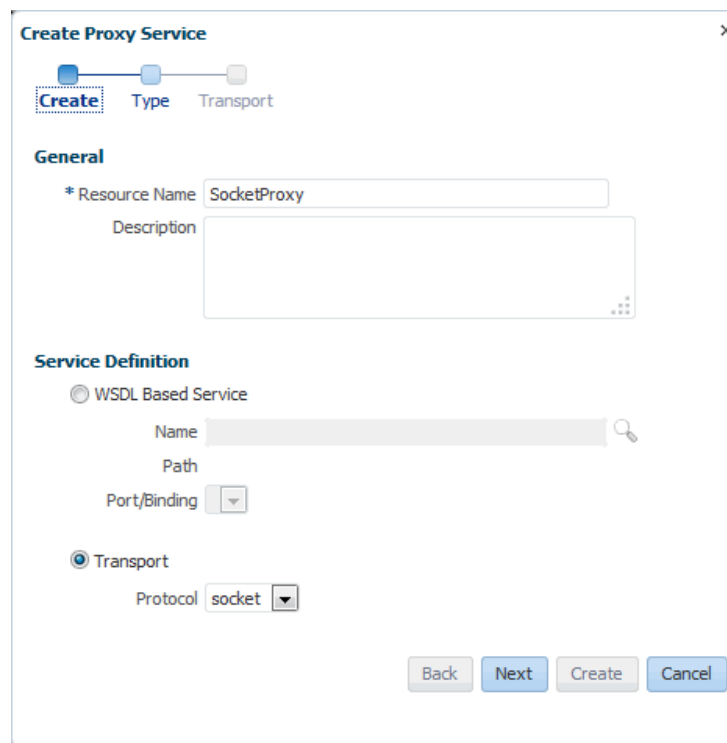
In this section, you create a proxy service to accept messages from the client.

To create the proxy service:

1. In the Project Navigator, right-click the SocketTest project, point to **Create**, and select **Proxy Service**.

The Create Proxy Service wizard appears.

Figure 43-5 Create Proxy Service Wizard



2. In the **Resource Name** field, enter `SocketProxy`.
3. In the **Transport** field, select **socket**. Click **Next**.
4. For the **Service Type**, select **Any XML Service**, and click **Next**.
5. In the Endpoint URI field, change the default URI to `7032`.
6. Click **Create**.

The proxy service appears in the Proxy Service Definition Editor. You can click the different subtabs to view the configuration.

7. Click **Save**.

Creating the Pipeline

Now that the business and proxy services are defined, you can create a pipeline to route incoming messages to the business service.

To create the pipeline:

1. In the Project Navigator, right-click the SocketTest project, point to **Create**, and select **Pipeline**.

The Create Pipeline dialog appears.

2. In the **Pipeline Name** field, enter `SocketPipeline`.
3. For the **Service Type**, select **Any XML Service**.
4. Deselect **Expose as Proxy Service**.

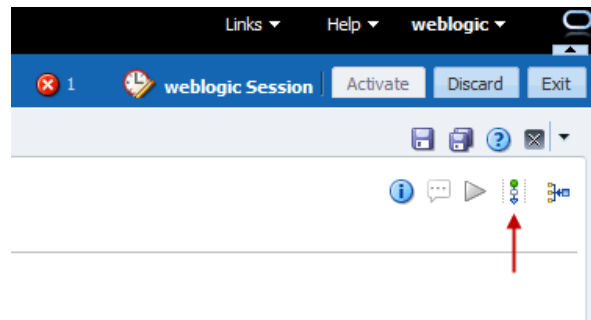
Figure 43-6 Create Pipeline Dialog

5. Click **Create**.

The pipeline appears in the Pipeline Definition Editor. You can click the different subtabs to view the configuration.

6. Click the **Open Message Flow** icon in upper right section of the editor.

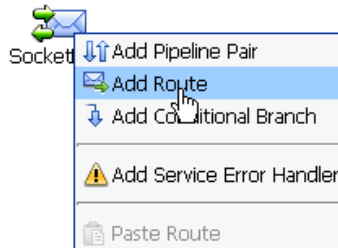
Figure 43-7



The Edit Message Flow window appears.

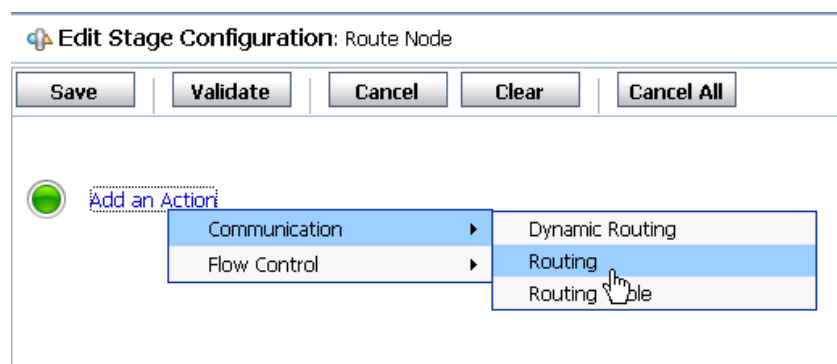
7. Click the **SocketPipeline** icon and select **Add Route** from the menu, as shown in [Figure 43-8](#).

Figure 43-8 *Editing the Message Flow*



8. Click the **RouteNode1** icon and select **Edit Route**.
9. In the Edit Stage Configuration window, click **Add an Action** and select **Communication > Routing**, as shown in [Figure 43-9](#).

Figure 43-9 *Adding an Action*



10. Next to **Route to**, select **<Service>**.
11. In the Select Service window, select **SocketBS** from the list, and click **Submit**.
12. In the Edit Stage Configuration window, click **Save**.
13. Optionally, click the **RouteNode1** icon, change the name to **SocketBS**, and then click **Save**.

14. Click **Save** again.

You are returned to the Pipeline Definition Editor.

Connecting the Proxy Service and Pipeline

In this section, you configure the proxy service to send messages to the pipeline you created.

To connect the proxy service and pipeline:

1. In the Project Navigator, click the SocketProxy proxy service.
The Proxy Service Definition Editor appears.
2. On the General subtab of the Configuration tab, click **Choose a Service Resource** by the **Target Name** field.
3. On the Search and Select dialog, enter **SocketPipeline** in the **Name** field and click **Search**.
4. Select the pipeline in the results list, and then click **OK**.
5. Click **Save**.
6. Click **Activate** to deploy the new resources to the WebLogic Server.

Testing the Socket Transport Provider

You can test the socket transport provider using the client and server tools provided with the sample files, and you can test project components using the Test Console in the Oracle Service Bus Console.

- [Using the Sample Server and Client for Testing](#)
- [Using the Test Console](#)

Using the Sample Server and Client for Testing

The sample project includes a simple socket server and a client to test the socket transport provider. First you need to start the sample server and client, and then you can work with the Test Console to test the transport provider.

Starting the Sample External Service

Run the following command from the `sample-transport` directory to start the test server, which is a server socket that listens on a specified port and receives and sends the messages.

```
java -classpath .\test\build\test-client.jar -Dfile-encoding=utf-8  
-Drequest-encoding=utf-8 com.bea.alsb.transports.sample.test.TestServer 7031  
<message-file-location>
```

7031 is the port number in the business service endpoint URI, where `ServerSocket` is listening. The file and request encoding indicate the encoding of the request and response. `message-file-location` is the path and name of the message file to send as a response to the business service.

If the server is started successfully, a message appears indicating that it is started and listening on a socket. If you specify a file to process, the text of the file appears in the command window.

Starting the Sample Initiating Service

Run the following command to start the initiating socket service, which is a client to the configured socket proxy service. It sends a message and receives the response from Service Bus.

```
java -classpath .\test\build\test-client.jar -Dfile-encoding=utf-8
-Dresponse-encoding=utf-8 com.bea.alsb.transports.sample.test.TestClient
<host-name> <port> <thread-ct> <message-file-location>
```

where:

- `host-name` is the host name on which the Service Bus server is located.
- `port` is the port number at which the proxy service is listening (7032, in our example).
- `thread-ct` is the number of clients that can send a message to Service Bus.
- `message-file-location` (optional) is the location of the message file to send as a response to the business service.
- `file-encoding` is an optional argument specifying the encoding of the file. The default is `utf-8`.
- `response-encoding` is the encoding of the response received from the socket proxy service. The default is `utf-8`.

Using the Test Console

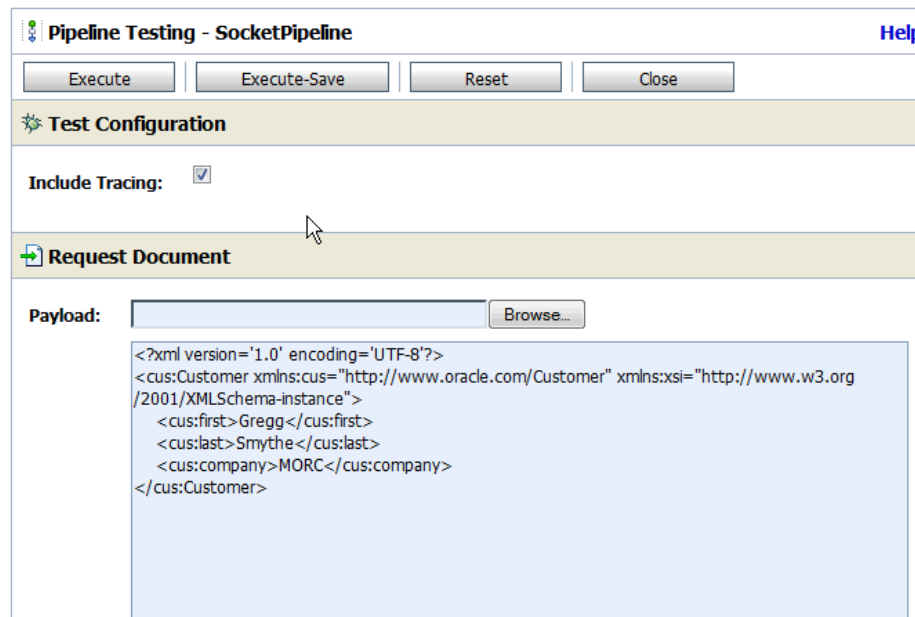
In this section you test the transport provider using the Oracle Service Bus Console.

To test using the Test Console:

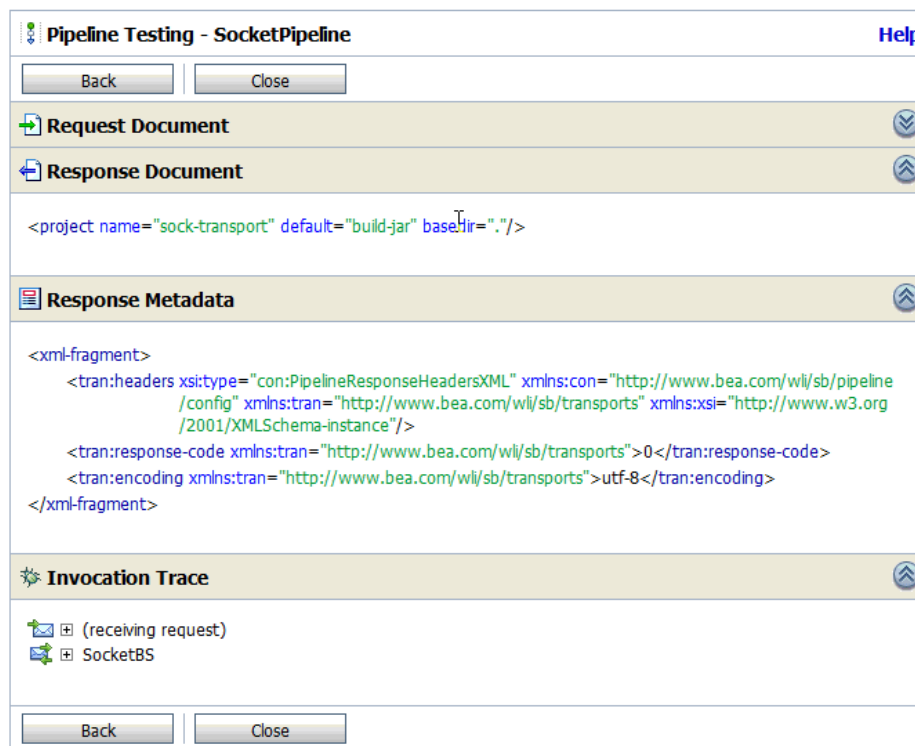
1. Start the test server, as explained previously in [Starting the Sample External Service](#).
2. In the Project Navigator, click **SocketPipeline** to open it in the Pipeline Definition Editor.
3. In the upper right portion of the editor, click the **Launch Test Console** icon.

The Test Console appears in a new browser window.

4. In the Test Console, enter any valid XML stanza in the text area, or use the **Browse** button to select a valid XML file on the local system.

Figure 43-10 Testing the Sample Transport Provider in the Test Console

5. Click **Execute**. If the test is successful, information similar that shown in [Figure 43-11](#) appears in the Test Console. In addition, the XML text input into the Test Console is echoed in the server console.

Figure 43-11 Successful Sample Transport Provider Test

6. Close the Test Console.

Part VII

Sharing Artifacts and Services

There are several methods by which you can share Service Bus projects and resources across Service Bus servers, instances, and applications. You can also access Oracle SOA Suite artifacts using common repositories. The import and export features provided by Service Bus let you create Service Bus projects and then share those projects or just individual resources with other Service Bus instances and servers. In addition, Oracle SOA Suite and Service Bus share common repositories that allow you to share deployed artifacts between projects and applications. UDDI registries are another way to share services.

This part contains the following chapters:

- [Importing and Exporting Resources and Configurations](#)
- [Sharing Data Using the Metadata Services Repository](#)
- [Working with UDDI Registries](#)

Importing and Exporting Resources and Configurations

Service Bus provides multiple ways to import and export resources from JDeveloper or the Oracle Service Bus Console, and to export resources from a command line. You can also import and export from Fusion Middleware Control Console. This chapter describes how to import and export resources using JDeveloper and the Oracle Service Bus Console, and how to export using a command line.

This chapter includes the following sections:

- [About Importing and Exporting Resources](#)
- [Importing and Exporting Resources in JDeveloper](#)
- [Importing and Exporting Resources in the Oracle Service Bus Console](#)
- [Exporting a Service Bus Configuration Offline](#)

For information about importing and exporting projects using Fusion Middleware Control, see "Importing and Exporting Oracle Service Bus Resources" in *Administering Oracle Service Bus*.

About Importing and Exporting Resources

Service Bus provides import and export features to help you move your Service Bus projects and resources between domains and between development tools. For example, when you move from development to testing, you can export projects from the development environment and import them into the testing environment. When shared resources change in a production environment, the resources can be updated for each server using the export and import features. These processes support an orderly promotion of resource configurations from staging and test environments into production environments.

You can also export and import resources to move them between JDeveloper and the Oracle Service Bus Console. For example, you create certain artifacts, such as JCA adapter files and WADL files, in JDeveloper. To use those artifacts in projects in the Oracle Service Bus Console, exporting them from JDeveloper and importing them to the console maintains any references between those artifacts.

You can use existing source code control systems in conjunction with the configuration JAR files to provide version and change management for Service Bus configurations.

About Exporting Resources

You can export entire Service Bus projects or just individual resources. The export process creates a configuration JAR file that you can then import into a different instance of JDeveloper, the Oracle Service Bus Console, or Fusion Middleware Control. When you export at the project level, you can select one or more projects in the current

instance to export. The Export Service Bus Resources wizard displays the projects, folders, and resources in a tree view, but component selection is only at the project level. When you export at the resources level, you can expand the projects to view and select individual resources to export. To avoid validation errors, you can also choose to automatically include any dependencies on the resources you select for export.

In the Oracle Service Bus Console, you can export projects and resources whether you are working within a session or outside of a session. If you export within a session, the resources are session resources and the configuration may be incomplete or have conflicts. If you export outside of a session, the resources you can export are the activated resources.

Service Bus cannot export the users, groups, or roles that you create for the console; nor can it export credential maps or other security-provider data that you create in the WebLogic Server Administration Console. Instead, use the WebLogic Server Administration Console to export this data. See "Migrating Security Data" in *Administering Security for Oracle WebLogic Server*.

Data Encryption During Export

When you export Service Bus projects or resources to a configuration JAR file, you can encrypt the user name and password data in service account, service key provider, UDDI registry, JNDI provider, and SMTP server resources. When you then import this JAR file, Service Bus will not import the resources with encrypted user name and password data unless you specify the correct password. You can import all of the other non-encrypted resources in the JAR file without specifying the password.

For each service account, Service Bus exports the user name and password or the local-user to remote-user map (depending on which data was stored in the service account). For each service key provider, Service Bus exports the alias to key-pair binding from the PKI credential mapping provider; it does not export private keys, certificates, or other data from the key stores. Key store data must be exported using tools that the key store vendor provides.

About Importing Resources

You can import complete Service Bus projects or just specific resources contained in configuration JAR files and resource JAR files that were previously exported from another Service Bus domain. You can choose to import only a subset of the exported data. If the resource already exists in the importing system, it will be updated. Otherwise, it will be created. Resources are only scheduled for deletion when the JAR being imported is a full project JAR and there are resources located in the same project in the importing system that are not present in the imported JAR file. It will not delete resources which are located in other projects.

Service Bus gives you the option to Service Bus import resources at the resource level even though they were exported at the project level. For example, even if system resources were exported in a full project JAR file, you can deselect them when importing. You cannot export users, groups, roles, or certificates when you export a configuration. Therefore, you must create these objects again when you import an exported configuration.

Improving Import Performance

When you import a Service Bus project, several validations are performed against the imported resources. Depending on the size of the configuration JAR file being imported, this can slow down the import process. By default, Service Bus processes four validation threads in parallel during import. To improve performance, you can

control how many threads can be used for validation by adding the following argument to the Java options in the Service Bus server start scripts (substitute the maximum number of threads to use for *no_of_threads*).

```
-Doracle.osb.config.parallelism=no_of_threads
```

This sets the number of threads for the Oracle Service Bus Console and for Fusion Middleware Control. To set this value for JDeveloper, add the argument to the `jdev.conf` file, located in `MW_HOME/jdeveloper/jdev/bin`.

Importing Service Accounts or Service Key Providers

If the JAR file being imported was created by AquaLogic Service Bus 3.0 or later and contains service accounts or service key providers, you can import these resources along with the user names, passwords, local-user to remote-user mappings, and alias to key-pair bindings that they contain.

For each service key provider, Service Bus imports the alias to key-pair binding into the PKI credential mapping provider. If this data was encrypted during export, you must supply the password that was used to encrypt the data. If you do not know the password, you can import all other non-encrypted resources.

If you import a service account or service key provider and a corresponding resource of the same name already exists in your domain, the imported resource will overwrite the one already in your domain, even if the one already in your domain has been modified during the current session, *unless* you specify to preserve security settings during import. For more information, see [Preserving Security Configuration During Import](#).

Preserving Operational Settings During Import

There are two types of operational values: global operational settings and operational settings for individual components. Global operational settings are imported like any other resource, though you can preserve operational settings in the importing domain and prevent them from being overwritten during import by selecting `Preserve Operational Values` during the import. If `Preserve Operational Values` is not specified, the values from the JAR file being imported are set in the domain.

Preserving Security Configuration During Import

You can export and import Service Bus resources without losing any associated security configuration data. The import and export wizards let you specify whether to preserve the existing security configuration on the importing system or to overwrite the existing configuration with that of the resources you are importing. This is only an issue when the resources being imported already exist on the importing system and are updating those existing resources.

For example, you might want to configure your credentials in a staging area, export a project that contains these credentials, and then import the project in your production environment. When you export the project, the security configuration is included in the Service Bus configuration JAR file. When you then import the project on your target system, security handling depends on whether the imported resources already exist on the target system:

- When the resources you import are new, that is they exist in the JAR file and not the target system, the resources use the security configuration from the JAR file.
- When the resources you import exist both on the import target server and in the JAR file, the resources use the security configuration you specify. You can preserve

the existing security configuration on the target system or overwrite it with the configuration in the JAR file.

The options you select during import allow you to decide which aspects of the security configuration for the resources on the importing system are preserved during import. These options work the same way when importing JAR files created by both project-level exports and for individual resource exports. The security options during import include the following:

- Preserve Security and Policy Configuration
- Preserve Credentials
- Preserve Access Control Policies

Note that when you import services configured to use WebLogic Server (WLS) policies, you will be unable to edit those policies because support for WLS policy is deprecated. You should reconfigure those services with Oracle Web Services Manager policies.

Preserve Security and Policy Configuration

Selecting **Preserve Security and Policy Configuration** preserves the following configuration parameters of the resources on the importing system. The configuration of the resources being imported is not preserved.

- Proxy service security and policy configuration:
 - A reference to the service key provider.
 - The set of policies that are bound directly to the service through the Policies tab. If the service uses WSDL-based policies, the policies are not preserved because the WSDL document itself might be updated and the service must reflect the WSDL document.
 - The state of the Process WS-Security Header option.
 - Message-level custom authentication configuration.
- Proxy service transport-specific security configuration:
 - For HTTP, the HTTPS flag and the authentication mode (anonymous, basic, client certificate, or custom token).
 - For JMS, the JMS and JNDI service accounts.
 - For email and FTP, the service account reference.
 - For SFTP, the authentication configuration.
- Business service security and policy configuration:
 - WS-Policy bindings.
 - The Pass Caller's Subject setting.
 - A reference to the service account for outbound WS-Security.
- Business service transport-specific security configuration:

- For HTTP, the authentication mode (anonymous, basic, or client certificate) and the service account reference.
- For JMS, references to the JMS and JNDI service accounts.
- For FTP, EJB, Tuxedo, and DSP, the service account reference.
- For SFTP, the authentication configuration.

Preserve Credentials

Selecting **Preserve Credentials** preserves the following credentials for the resources on the importing system. The configuration of the resources being imported is not preserved.

- PKI credentials in service key providers.
A PKI credential mapping provider maps service key providers to key-pairs that can be used for digital signatures and encryption and for outbound SSL authentication.
- User name and passwords in service accounts.
- User name and password in SMTP server, JNDI provider, and UDDI registries.

Preserve Access Control

Selecting **Preserve Access Control Policies** preserves all access control policies for the proxy services on the importing system during the import process. The access control policies of the proxy services being imported are not preserved.

Customizing Environment Values After an Import

Using the customization feature, imported resources can be tailored for the new domain before activating them. Configuration files let you globally change environment-specific attributes for resources using the import functionality along with the find and replace feature. This is not meant to replace a more careful tuning of configuration that may be required by complex deployment scenarios.

In addition to service endpoint URIs, directory names, and security configuration, your Service Bus configuration may contain other settings that must be updated to operate correctly in the new environment. Items that commonly require update include the following:

- Service references to other Service Bus resources.
- JMS queues and connection factories in proxy or business service URLs.
- Routing destinations in pipelines.
- Endpoint URIs for business and proxy services.
- Load balancing settings for business services.
- Directory names in the transport configuration for certain polling proxy and business services.

For information about configuration files, see "Customizing Oracle Service Bus Environments" in *Administering Oracle Service Bus*.

Importing and Exporting Resources in JDeveloper

Service Bus lets you import from and export to a variety of sources. You can export Service Bus resources to a configuration JAR file for later import, and you can export resources directly to a running WebLogic Server. You can import resources from a configuration JAR file that was previously exported from a Service Bus instance, from a ZIP file contains Service Bus resources, or from a URL where Service Bus resources are located. You can also access resources in a UDDI registry, which is described in [Working with UDDI Registries](#).

Caution:

Before performing an export, be sure to save all the resources you want to export. Before performing an import, be sure to save all the files in the application to which you are importing.

How to Export Resources to a Configuration JAR File in Oracle JDeveloper

When you export Service Bus resources to a file from JDeveloper, Service Bus generates a JAR file that can later be imported into a different Service Bus instance. When you export complete projects, the overview file, `servicebus.sboverview`, is not included in the export, but it is regenerated when the project is imported again.

To export resources to a configuration JAR file in JDeveloper:

1. With a Service Bus application open, right-click in the Application Navigator and select **Export**.

2. On the Export dialog, select **Service Bus Resources**, and click **OK**.

The Export Service Bus Resources wizard appears.

3. On the Type page, select **Configuration JAR**, and click **Next**.

A list of resources included in the selected source appears.

4. In the Export Level field, select the **project** to export complete projects, or select **resource** to export individual resources.
5. Expand the list of resources, and make sure only the ones you want to export are selected.
6. For a resource-level export, select **Include Dependencies** to export any additional resources referenced by the selected resources. Clear the check box to only export the resources you select.
7. Enter the path and file name for the configuration JAR file that will be generated.
8. To encrypt sensitive data in the exported resources, select **Protect Sensitive Data** and then enter the password to unlock the file in the **Passphrase** and **Confirm Passphrase** fields.

For more information, see [Data Encryption During Export](#).

9. Click **Finish**.

Service Bus generates the configuration JAR file in the location you specified.

How to Export Resources to a Server in Oracle JDeveloper

When you export Service Bus resources directly to a server, Service Bus does not generate a configuration JAR file that you can later import. Rather it exports the resources and then imports them directly into the WebLogic server you specify.

Before exporting to a server, the server must have an application connection defined in JDeveloper. Otherwise, you will be unable to select it from the list. To create an application server connection, see [How to Create an Application Server Connection](#).

To export resources to a server in JDeveloper:

1. With a Service Bus application open, right-click in the Application Navigator and select **Export**.
2. On the Export dialog, select **Service Bus Resources**, and click **OK**.
The Export Service Bus Resources wizard appears.
3. On the Type page, select **Server**.
4. Click **Next**.
A list of resources included in the selected source appears.
5. In the Export Level field, select the **project** to export complete projects, or select **resource** to export individual resources.
6. Expand the list of resources, and make sure only the ones you want to export are selected.
7. For a resource-level export, select **Include Dependencies** to export any additional resources referenced by the selected resources. Clear the check box to only export the selected resources.
8. In the **Server** field, select the application server connection for the server to which you want to export the resources.

Tip:

If the server you are connecting to is not running, click **Start Server** at the bottom of the window to start it.

9. In the **Session** field, enter the name of the session with which you want to associate the imported resources.
10. Click **Next**.
A list of the exported resources appears.
11. Expand the list of resources, and make sure only the ones you want to import are selected.
12. To automatically import any dependent resources, select **Include Dependencies**.
13. Select any of the environment or security settings you want to preserve from the imported resources.

For information about these settings see [Preserving Security Configuration During Import](#) and the online help provided with JDeveloper.

14. Do the following to configure the session:
 - a. To create and activate the session in the Oracle Service Bus Console, select **Activate session after publish** and enter a description for the session.
 - b. To discard the session if conflicts result from the import or the session is unable to be activated, select **Discard session if activation fails**.
15. To specify a configuration file to use to update environmental values, click **Browse** next to the **Deployment customization file** field, and navigate to and select the file to use.
16. Click **Finish**.

The selected resources are published to the specified server.

How to Import Resources in JDeveloper

You can import Service Bus from a variety of sources, including a previously exported configuration JAR file, an archived ZIP file, and a URL. When you import resources from a URL, you can browse multiple sources for the resource, including the file system, MDS repository, the current application, application server, project libraries, UDDI registries, and WSIL sources. The available sources vary based on the type of resource you are importing.

If you import a new Service Bus project, Service Bus generates the project overview file and adds all derivable Service Bus resources to the diagram in the Service Bus Overview Editor. If your import updates an existing Service Bus project, the overview diagram is also updated with any new resources or updated wiring. Any system resources included in the imported file are added to the **Service Bus System Resources** folder in the Application Resources panel. System resources are located in the `System` folder in the application folder.

Caution:

When you import an archived ZIP file, the file extension for any XQuery resources in the ZIP file must be `.xqy` and not `.xquery`, which was the default extension in previous versions. The extension is updated automatically when you import a configuration JAR file.

To import resources in JDeveloper:

1. With a Service Bus application open, right-click in the Application Navigator and select **Import**.
2. On the Import dialog, select **Service Bus Resources**, and click **OK**.

The Import Service Bus Resources wizard appears.
3. On the Type page, select one of the following sources to import:
 - Configuration JAR
 - Resources from URL
 - Zipped Resources

4. Click **Next**.

The options that appear on the Source page vary depending on your previous selection.

5. Do one of the following:

- If you selected **Configuration JAR**, browse to and select the name of the Service Bus JAR file to import.
- If you selected **Resources from URL**, select the resource type from the list of options, browse to and select the resource URL, and specify the resource name.
- If you selected **Zipped Resources**, browse to and select the name of the ZIP file containing the resources to import.

6. Click **Next**.

A list of resources included in the selected source appears, along with the type of operation (create, update, or delete) to be performed on each.

7. Expand the list of resources, and make sure only the ones you want to import are selected.

Caution:

Review all resources that are marked for deletion, and be sure you want to delete them. If you do not want them to be deleted, clear their check boxes.

8. If you are importing a configuration JAR file, do any of the following:

- To automatically import any dependent resources, select **Include Dependencies**.
- If the imported source is password protected, enter the password in the **Passphrase** field.
- Select any of the environment or security settings you want to preserve from the imported resources.

For information about these settings see [Preserving Security Configuration During Import](#) and the online help provided with JDeveloper

9. Click **Finish**.

The selected resources are imported and appear in the Application Navigator.

Importing and Exporting Resources in the Oracle Service Bus Console

The Service Bus import feature lets you import from a configuration JAR file that was previously exported from a Service Bus instance, from a ZIP file containing Service Bus resources, or from a URL where Service Bus resources are located. You can also import from and publish to a UDDI registry, which is described in [Working with UDDI Registries](#).

Caution:

Before performing an export, be sure to save all the resources you want to export. Before performing an import, be sure to save all the files in the application to which you are importing.

How to Export Resources to a Configuration JAR File in the Console

When you export Service Bus resources from the Oracle Service Bus Console, Service Bus generates a JAR file that can later be imported into a different Service Bus instance.

To export resources to a configuration JAR file in the console:

1. In the Project Navigator, do any of the following:
 - To select all projects to export, click **Export** in the Resources toolbar or right-click **All Projects** and select **Export**.
 - To select a specific project to export, select that project and then right-click it. Select **Export** from the menu that appears.

The Export Resources dialog appears with list of resources available to export.

2. Under **Export Contents**, select **Projects** to export complete projects, or select **Resources** to export individual resources.
3. Expand the list of resources, and make sure only the ones you want to export are selected.
4. For a resource-level export, select **Include Resource Dependencies** to export any additional resources referenced by the selected resources. Clear the check box to only export the resources you select.
5. To encrypt sensitive data in the exported resources, select **Protect Sensitive Data** and then enter the password to unlock the file in the **Passphrase** and **Confirm Passphrase** fields.

For more information, see [Data Encryption During Export](#).

6. Click **Export**.
7. On the File Download dialog, click **Save**, specify a location and filename for the configuration JAR file and click **Save** again.

Service Bus generates the configuration JAR file in the location you specified.

How to Import Resources from a Configuration JAR File in the Console

Note:

When you import a configuration JAR file containing XQuery resources from previous versions, the file extension of the resources is automatically updated from `.xquery` to `.xqy`. The default extension in previous versions was `.xquery`.

To import resources from a JAR file in the console:

1. Do one of the following:
 - In the Project Navigator, right-click **All Projects**, point to **Import**, and then select **Config Jar**.
 - Click the **Import** icon in the Resources toolbar.
2. On the Import Config JAR wizard, browse to and select the JAR file to import.
3. Click **Next**.

A list of resources included in the selected source appears, along with the type of operation (create, update, or delete) to be performed on each.

4. Expand the list of resources, and select the ones you want to import. By default, all resources are selected.

Caution:

Review all resources that are marked for deletion, and be sure you want to delete them. If you do not want them to be deleted, clear their check boxes.

5. To automatically import any dependent resources, select **Include Dependencies**.
6. If the imported source is password protected, enter the password in the **Passphrase** field.
7. Select any of the advanced settings you want to preserve from the imported resources. You can preserve any of the following:
 - Security and policy values
 - Credentials
 - Access control policies
 - Environmental variable values
 - Operational settings

For more information, see [Preserving Security Configuration During Import](#) or the online help provided with the console.

8. Click **Import**.

The selected resources appear in the Project Navigator. The Import Config JAR wizard displays a summary of the import along with any errors in the imported resources.

9. To view any resources that were deleted during the import process, click the **Deleted Resources** tab.
10. To import another JAR file, click **Import Another** and repeat the above steps. Otherwise, click **Close**.

How to Import Resources from a ZIP File in the Console

When you import a ZIP file, each file in the ZIP file is a possible resource and Service Bus identifies the type of resource in each file by its file extension. Each resource has a default extension in Service Bus, but if you have additional extensions to describe those resources, you can define those extension mappings when you import the ZIP file. An extension can only be associated with one file type.

During the import process, Service Bus scans the contents of the ZIP file and tries to associate a resource type with each file. A file without an extension or one that does not have an extension defined in the map is considered *unknown* and is automatically excluded from the load. For known files, the name of the resource is the name of the file without its extension. The folder structure of the Zip file is recreated in the target project or folder.

Resources like WSDL documents or XML Schemas can define full trees of dependent resources. The bulk load feature lets you upload a set of resources at once so you do not need to resolve the dependencies manually.

Caution:

When you import an archived ZIP file, the file extension for any XQuery resources in the ZIP file must be `.xqy` and not `.xquery`, which was the default extension in previous versions.

To import resources from a ZIP file in the console:

1. In the Project Navigator, right-click the project or folder into which you want to import resources, point to **Import**, and then select **Zip File**.
2. On the Import from a ZIP File wizard, browse to and select the ZIP file to import.
3. In the Extension Mappings table, review the mapping of resource types to file extensions. Change or add file extensions, according to your needs. Separate multiple extensions with commas.
4. Click **Next**.

A list of resources included in the selected source appears, along with the type of operation (create, update, or delete) to be performed on each.

5. Expand the list of resources, and select the ones you want to import. By default, all resources are selected.
6. Click **Import**.

The selected resources appear in the Project Navigator. The Import from a ZIP File wizard displays a summary of the import along with any errors in the imported resources.

7. To import another ZIP file, click **Import Another** and repeat the above steps. Otherwise, click **Close**.

How to Import Resources from a URL in the Console

You can perform a bulk import of resources available at a URL or on the file system. Bulk import lets you import a root resource, such as a WSDL document, along with its dependents, such as other WSDL documents and schemas. The dependency map is resolved automatically.

A Warning icon next to file name indicates that the resource type is unknown. (A file without an extension or one that does not have an extension defined in the map is considered *unknown*.) Files of unknown file types cannot be imported.

To import resources from a URL in the console:

1. In the Project Navigator, right-click the project or folder into which you want to import resources, point to **Import**, and then select **From URL**.
2. On the Import from URL wizard, enter the following information:
 - **Resource Type:** Select the type of resource to import from the list of options.
 - **URL Source:** Enter the URL or local path to the resource to import. To specify a local resource, use the `file` protocol. For example:
`file:///c:/osbresources/ForeachAction.jar`
 - **Resource Name:** Enter the name of the resource to import.

3. Click **Next**.

A list of resources identified in the URL appears.

4. Expand the list of resources, and select the ones you want to import. By default, all resources are selected.
5. Click **Import**.

The selected resources appear in the Project Navigator. The Import from URL wizard displays a summary of the import along with any errors in the imported resources.

6. To import more resources from a URL, click **Import Another** and repeat the above steps. Otherwise, click **Close**.

Exporting a Service Bus Configuration Offline

This section describes how to export a Service Bus configuration when you are not connected to a server. You can export complete projects or individual resources into a configuration JAR file that can then be imported into a new Service Bus environment or instance.

- [About the Export Process](#)
- [Preparing to Export a Service Bus Configuration](#)
- [Exporting a Service Bus Configuration Offline](#)
- [Export Settings File Format, Samples, and Schema](#)

About the Export Process

You can export a Service Bus configuration in offline mode using a command line, Ant task, or the WebLogic Scripting Tool (WLST). All methods use an export settings file to define how the export is executed and which files, folders, projects, or system resources to include in the generated configuration JAR file.

The export tool runs in two phases, load and export, and each phase is configured in an export settings file. During the load phase, the export tool traverses the file system, identifies the files to read, converts the file content into the corresponding resource, and imports the file into the configuration framework.

For the load phase, you can configure the following:

- **Directory structure:** There is no dependency on the JDeveloper work directory structure; you can specify project root directories, directories where system resources are located, and specific folders and files to include.
- **File extensions:** In Service Bus, each resource type uses a specific file extension, such as `.proxy` for proxy services, `.xsl` and `.xslt` for XSLT resources, and so on. Each extension can map to only one Service Bus resource type, but a resource type can map to multiple file extensions. Each Service Bus resource has one default extension that cannot be changed, but you can also specify custom file extensions for a resource. In order for the export tool to recognize files with custom extensions, you need to define file extension mappings in the export settings file.
- **Inclusion and exclusion rules:** When you specify a project or system resources folder, all Service Bus files are included in the export. You can use include and exclude statements for finer control over which files are included. For example, certain versioning systems create additional system folders and files within the project folders. These files might be recognized as Service Bus resources and included in the exported JAR file unless they are specifically excluded from the export.

The Service Bus configuration can be exported at the project level and at the resource level, and you can define multiple configuration exports in one export settings file. For the export phase, you can configure the following:

- **Project-level export:** At the project level, you specify the names of the projects to include as well as whether to include system resources. If no project names are provided, all projects added during the load phase are included in the generated export file.

Note:

Do not export at the project level if detailed inclusions have been defined for the load phase; this could result in deleting all resources that were not included.

- **Resource-level export:** At the resource level, you specify the resources to export using inclusion and exclusion rules as described above. You can also specify whether to include dependencies. If no resources are specified in the export settings file, all resources are exported.

- **Multiple configuration JAR files:** You can define multiple `configjar` elements in the export settings file, each of which defines the export to a specific file. You can also specify whether to overwrite existing files of the same name.

Preparing to Export a Service Bus Configuration

Perform the following steps before you export a Service Bus configuration in offline mode:

- [Before You Begin](#)
- [Creating the Export Settings File](#)
- [Configuring the Environment](#)

Before You Begin

Before you begin exporting resources, verify that you have the following installed:

- Service Bus 12.1.3
- Java 1.7.x

When you export the configuration, note the following:

- The resource JAR names in your scripts contain the correct version numbers.
- You may see exception stack traces in the output or the workspace log file if JDeveloper work files are read-only.
- An exit value of 0 means the export succeeded.

Note:

If you have developed a custom transport, you need to create an offline transport plug-in file and save it to `OSB_ORACLE_HOME/config/plugins` in your Service Bus installation.

Creating the Export Settings File

Rather than using command line arguments when performing an offline export, the export tool refers to a settings file that you create. This file is in XML format and contains all of the required information for the export tool to be able to find and load files and then create the configuration JAR file. For information about the file format and schema definition, as well as examples of usage, see [Export Settings File Format, Samples, and Schema](#).

Configuring the Environment

Before performing the export, you need run the `setenv.*` file to set the environment variables used by the tool. You can customize this file before running it by modifying the JVM heap size, adding JAR files to the `CLASSPATH`, or adding system properties. If you have JAR resources that are included in the export and that require custom JAR files, make sure to add the custom JAR files to the `setenv.*` file.

Once you have customized the file, navigate to `OSB_ORACLE_HOME/tools/configjar` and run the following command:

For Windows:

```
setenv.bat
```

For UNIX or Linux:

```
source setenv.sh
```

Exporting a Service Bus Configuration Offline

The following sections describe scripting and command-line options for performing an offline export of a Service Bus configuration:

- [Exporting a Configuration Offline Using a Command Line](#)
- [Exporting a Configuration Offline Using Ant](#)
- [Exporting a Configuration Offline Using WLST](#)

Exporting a Configuration Offline Using a Command Line

Exporting from the command line generates a Service Bus configuration JAR file from the folders and files you specify in the export settings file.

Syntax

Use the following syntax for Windows:

```
configjar.bat -settingsfile <FILE_NAME> [-debug -help]
```

Use the following syntax for UNIX or Linux:

```
./configjar.sh -settingsfile <FILE_NAME> [-debug -help]
```

Parameters

The following parameters can be used at the command line:

- `-settingsfile FILE_NAME`: Enter the path and filename of the export settings file. This parameter is required.
- `-debug`: Include this optional parameter in the command to enable debug logging of the export process. If this flag is not included, debug logging is not performed.
- `-help`: Include this optional parameter in the command to view usage information.

Exporting a Configuration Offline Using Ant

You can export a Service Bus configuration in offline mode using an Apache Ant build file. Exporting using Ant generates the configuration JAR file from the folders and files you specified in the export settings file.

Sample Build File

Below is a sample Ant build file:

```
<project name="ConfigExport" basedir=".">
  <target name="run">
    <ant antfile="configjar-ant.xml" target="run">
      <property name="settingsFile" value="/osb/config/exportProps1.xml"/>
    </ant>
    <ant antfile="configjar-ant.xml" target="run">
      <property name="settingsFile" value="/osb/config/exportProps2.xml"/>
    </ant>
  </target>
</project>
```

```
</target>
</project>
```

Note:

This is only a sample script. You can use it as a basis for your own script, but be sure to check paths and file names against your current installation for accuracy.

Parameters

For Ant, the following parameters are supported:

- `settingsFile`: The path and filename of the export settings file. This parameter is required
- `debug`: Set this parameter to **true** to enable debug logging of the export process; otherwise set it to **false**. This parameter is optional and set to false by default.
- `failonerror`: Set this parameter to **true** if you want the task to fail when the export tool fails. If you set it to **false**, the task does not fail even if the export tool fails. This parameter is optional and set to true by default.
- `errorProperty`: If specified, this parameter is set to **true** if execution fails. This parameter is optional and is not specified by default.

Exporting a Configuration Offline Using WLST

You can export a Service Bus configuration file in offline mode using the WebLogic Scripting Tool (WLST). Exporting using WLST generates a Service Bus configuration JAR file from the folders and files you specified in the export settings file.

Sample WLST Script

Below is a sample WLST script:

```
from com.bea.alsb.tools.configjar import ConfigJar

args = []
args.append('-settings')
args.append('/osb/config/exportProps2.xml')
ConfigJar.main(args)
```

Parameters

The following parameters can be used:

- `-settingsfile FILE_NAME`: Enter the path and filename of the export settings file. If you use a relative path, the path resolves from the directory where the export tool resides (`OSB_ORACLE_HOME/tools/configjar`). This parameter is required.
- `-debug`: Include this optional parameter in the command to enable debug logging of the export process. If this flag is not included, debug logging is not performed.
- `-help`: Include this optional parameter in the command to view usage information.

Export Settings File Format, Samples, and Schema

By configuring the export settings, you can specify the files and folders to be included in the configuration JAR file that is exported. You can do this at the project and resource level, and you can use a series of exclusion and inclusion rules for a finer level of control. The included files and folders make up the *content set* for the export.

Export Settings File Format

The export settings file contains two main sections: `source` and `configjar`. The `source` element defines the files to be picked up by the export tool, and the `configjar` element defines which of the files that were picked up will be included in the generated configuration JAR file.

In the `source` element, you specify the project root directories for the projects to export, the location of the system resources to export, and specific files to include or exclude. You can also map custom file extensions to Service Bus resource types so the export tool can recognize them as valid Service Bus components.

Note:

If you specify relative directories for the project root or resource folders, the path is resolved relative to the directory in which the export settings file is located.

In the `configjar` element, you name the generated JAR file and specify rules at the project and resource level, including whether to include system resources at the project level and whether to include dependencies at the resource level. You can define multiple `configjar` elements; a JAR file is generated for each `configjar` element you define.

For guidance and restrictions on naming configuration JAR files, see [Naming Guidelines for Service Bus Components](#).

Validation

The same naming validation rules that are applied in Service Bus are also applied to the files to be included in the content set. Any files or folders that do not conform to these rules are excluded from the content set.

- Folders and files must have valid names.
- File extension must map to a Service Bus resource.

Two different files cannot map to the same instance of `com.bea.wli.config.Ref` (see the *Java API Reference for Oracle Service Bus* for more information).

Inclusion and Exclusion Rules

When you use inclusion and exclusion rules, a file must match at least one of the inclusion rules and none of the exclusion rules to be included in the content set. For a file to be excluded, it must match at least one of the exclusion rules or none of the inclusion rules. If no inclusion or exclusion rules are defined, all files are automatically included.

The inclusion and exclusion rules are based on a simple pattern.

- The pattern is applied on the file sub-path starting with the project name. It is not applied on the full file path.
- An asterisk (*) is a wildcard representing any character. It can be used as part of a file or directory name.
- When a double asterisk (**) is used alone, it matches zero or more directories and files.

Export Settings File Samples

This section provides sample export settings files that show ways to export Service Bus configurations using a variety of rule combinations for the load and export phases.

Example - Exporting from a Non-JDeveloper File Structure

The following sample illustrates how to package a project and its associated system resources from a structure like Maven. This sample outputs two configuration JAR files. The first is a project-level configuration with no system resources, and the second is a resource-level configuration with JNDI and SMTP resources.

```
<configjarSettings xmlns="http://www.bea.com/alsb/tools/configjar/config">
  <source>
    <project dir="/scratch/modulePO/src/main/resources/PO"/>
    <system dir="/scratch/modulePO/src/main/system"/>
  </source>
  <configjar jar="/scratch/modulePO/sbconfig-po.jar">
    <projectLevel includeSystem="false"/>
  </configjar>
  <configjar jar="/scratch/modulePO/sbconfig-po-system.jar">
    <resourceLevel>
      <resources>
        <include name="**/*.jndi"/>
        <include name="**/*.smtp"/>
      </resources>
    </resourceLevel>
  </configjar>
</configjarSettings>
```

Example - Exporting at the Project Level

The following sample illustrates how to package system resources from different locations and export them by project. You can delete any unneeded resources when you import the configuration.

```
<configjarSettings xmlns="http://www.bea.com/alsb/tools/configjar/config">
  <source>
    <system dir="/scratch/moduleX/src/main/system"/>
    <system dir="/scratch/moduleY/src/main/system"/>
    <system dir="/scratch/moduleZ/src/main/system"/>
  </source>
  <configjar jar="/scratch/sbconfig-systems.jar">
    <projectLevel includeSystem="true"/>
  </configjar>
</configjarSettings>
```

Example - Excluding File Extensions in the Load Phase

This sample illustrates how to package all non-service resources into a configuration JAR file by defining exclusion rules in the `source` element. Compare this with the

next example, "Excluding File Extensions in the Export Phase". Note that excluding files during the load phase is the recommended method for performance reasons.

```
<configjarSettings xmlns="http://www.bea.com/alsb/tools/configjar/config">
  <source>
    <project dir="/scratch/jdeveloper/mywork/projectX"/>
    <project dir="/scratch/jdeveloper/mywork/projectY"/>
    <fileset>
      <exclude name="**/*.proxy"/>
      <exclude name="**/*.biz"/>
      <exclude name="**/*.flow"/>
    </fileset>
  </source>
  <configjar jar="/scratch/jdeveloper/mywork/sbconfig-resources.jar">
    <resourceLevel/>
  </configjar>
</configjarSettings>
```

Example - Excluding File Extensions in the Export Phase

The following sample illustrates how to package all non-service resources into a configuration JAR file by defining exclusion rules in the `configjar` element.

```
<configjarSettings xmlns="http://www.bea.com/alsb/tools/configjar/config">
  <source>
    <project dir="/scratch/jdeveloper/mywork/projectX"/>
    <project dir="/scratch/jdeveloper/mywork/projectY"/>
  </source>
  <configjar jar="/scratch/jdeveloper/mywork/sbconfig-resources.jar">
    <resourceLevel>
      <fileset>
        <exclude name="**/*.proxy"/>
        <exclude name="**/*.biz"/>
        <exclude name="**/*.flow"/>
      </fileset>
    </resourceLevel>
  </configjar>
</configjarSettings>
```

Example - Mapping File Extensions

The following example illustrates how to map additional file extensions to specific Service Bus resource types, in this case to XQuery and XML types.

```
<configjarSettings xmlns="http://www.bea.com/alsb/tools/configjar/config">
  <source>
    <project dir="/scratch/jdeveloper/mywork/projectX"/>
    <project dir="/scratch/jdeveloper/mywork/projectY"/>
    <extensionMapping>
      <mapping type="Xquery" extensions="xquery,xq,xqy"/>
      <mapping type="XML" extensions="toplink"/>
    </extensionMapping>
  </source>
  <configjar jar="/scratch/jdeveloper/mywork/sbconfig.jar">
    <resourceLevel/>
  </configjar>
</configjarSettings>
```


Note:

When mapping file extensions, the type attribute must match a Service Bus resource type defined in `com.bea.wli.config.Ref`. For more information, see the *Java API Reference for Oracle Service Bus*.

Export Settings File Schema Definition

Below is the schema definition for the export settings XML file. Note that some of the text has been wrapped for readability.

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://www.bea.com/alsb/tools/configjar/config"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://www.bea.com/alsb/tools/configjar/config">
<xs:element name="configjarSettings" type="tns:configjarSettings"/>
<xs:complexType name="configjarSettings">
  <xs:sequence>
    <xs:element name="source" type="tns:source" />
    <xs:element name="configjar" type="tns:configjar" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="source">
  <xs:sequence>
    <xs:choice minOccurs="1" maxOccurs="unbounded">
      <xs:element name="project">
        <xs:complexType>
          <xs:attribute name="dir" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="system">
        <xs:complexType>
          <xs:attribute name="dir" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:choice>
    <xs:element name="extensionMapping" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="mapping" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="type" type="xs:string" use="required"/>
              <xs:attribute name="extensions" type="xs:string"
                use="required"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="fileset" type="tns:contentSet" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="configjar">
  <xs:sequence>
    <xs:choice minOccurs="1" maxOccurs="1">
      <xs:element name="projectLevel" type="tns:projectLevel"/>
      <xs:element name="resourceLevel" type="tns:resourceLevel"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

```
</xs:sequence>
  <xs:attribute name="jar" type="xs:string" use="required"/>
  <xs:attribute name="overwrite" type="xs:boolean" use="optional" default="true"/>
</xs:complexType>
<xs:complexType name="projectLevel">
  <xs:sequence>
    <xs:element name="project" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="includeSystem" type="xs:boolean" use="optional"
    default="false"/>
</xs:complexType>
<xs:complexType name="resourceLevel">
  <xs:sequence>
    <xs:element name="resources" type="tns:contentSet" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="includeDependencies" type="xs:boolean" use="optional"
    default="true"/>
</xs:complexType>
<xs:complexType name="contentSet">
  <xs:sequence>
    <xs:element name="include" type="tns:contentSetPattern" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="exclude" type="tns:contentSetPattern" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="contentSetPattern">
  <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
</xs:schema>
```

Sharing Data Using the Metadata Services Repository

When using JDeveloper, you can leverage a common Metadata Services (MDS) repository to store and share the artifacts generated for Service Bus and SOA Suite applications. Use this repository to create a backup of the artifacts you create and to share deployed artifacts across multiple servers, instances, applications, and products.

This chapter includes the following sections:

- [Service Bus and the MDS Repository](#)
- [Managing the MDS Repository](#)
- [Sharing Artifacts Using the MDS Repository](#)
- [Consuming Artifacts Stored in the MDS Repository](#)

For more information about the MDS Repository, see "Managing Shared Data with the Design-Time MDS Repository" in *Developing SOA Applications with Oracle SOA Suite*.

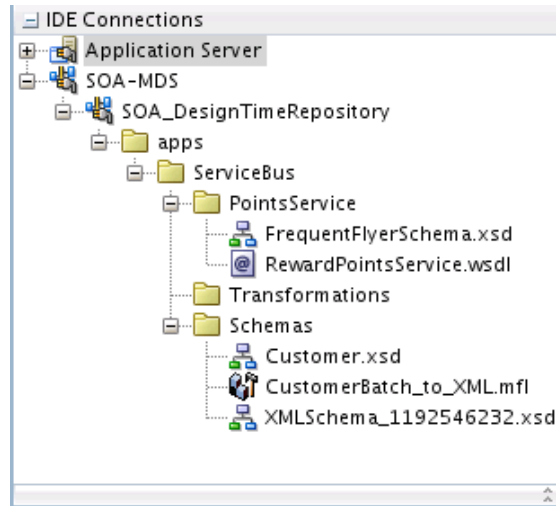
You can also share artifacts using UDDI registries. For more information, see [Working with UDDI Registries](#).

Service Bus and the MDS Repository

An MDS Repository stores information about Oracle Fusion Middleware components. The repository can be either file-based or database-based, but several design-time activities can only be performed against a file-based repository. For complete information about repository types, see *Managing the Metadata Repository in Administering Oracle Fusion Middleware*.

A file-based, design-time MDS Repository and connection is automatically included when you create a Service Bus application in JDeveloper. You can modify this connection to point to a different existing repository, or you can create new connections to point to different repositories. Sharing operations are done against the design-time repository, and cannot be done against a database-backed MDS Repository.

The Resources window in JDeveloper lets you browse the folders and artifacts stored in the MDS Repository. You can also create and delete folders; add, import, and delete Service Bus and SOA Suite artifacts, and generate Service Bus business services from WSDL files stored in the repository. The Resources window lets you export artifacts from the repository and import artifacts into the repository, both in the form of JAR files. The following figure shows Service Bus components in the MDS Repository in the Resources window.

Figure 45-1 Service Bus Artifacts in the MDS Repository

For more information about the MDS repository, see "Introduction to Design-Time MDS Repository Management" in *Developing SOA Applications with Oracle SOA Suite*.

Managing the MDS Repository

You can perform general management tasks on the MDS Repository, including seeding the repository with source data, transferring the contents of one repository to another, creating and deleting folders, and importing and exporting artifacts. The following topics in *Developing SOA Applications with Oracle SOA Suite* provide information to help you manage the repository:

- Populating the Default SOA-MDS Connection with Source Data
- Creating and Deleting Subfolders Under the /apps Folder
- Exporting the Selected Contents of the /apps Folder to a JAR File
- Importing the Contents of the JAR File into the /apps Folder
- Transferring the Selected Contents of the /apps Folder to Another MDS Repository

Sharing Artifacts Using the MDS Repository

You can share the following Service Bus artifact types with the design-time MDS Repository: WSDL, WADL, JCA, JAR, cross-reference, domain value map, WS policy, XSD, XML, XSLT, MFL, and XQuery files. These files can then be shared with other Service Bus applications and, for many of these artifact types, with SOA Suite applications.

How to Publish Service Bus Artifacts to the MDS Repository

Project components are published to the MDS repository using the SOA-MDS Transfer wizard, accessed by right-clicking any component that can be shared. These component can then be shared with other Service Bus and SOA Suite applications.

Before You Begin

Before you can work with artifacts in the MDS repository, you need to create a connection to the repository from JDeveloper. For instructions, see [How to Create a SOA-MDS Connection](#).

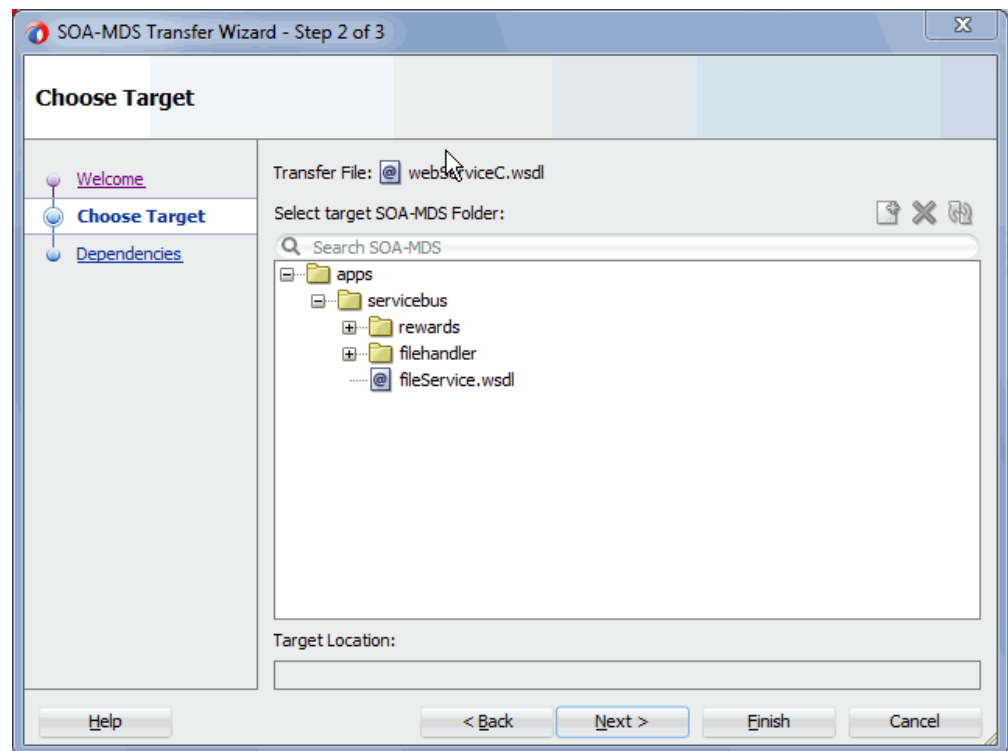
To publish Service Bus artifacts to the MDS Repository:

1. Make sure the component to import is not open in any JDeveloper editors. Also close any files on which the component depends, such as XML schema files.
2. In the JDeveloper Application Navigator, display the component you want to publish to the MDS Repository.
3. Right-click the component, point to **Service Bus**, and select **Publish to SOA Designtime Repository**.

The SOA-MDS Transfer wizard appears, listing the component to be published.

4. Click **Next**.
5. On the Choose Target window, expand the folders to the location where you want to publish the Service Bus component.

Figure 45-2 SOA-MDS Transfer Wizard - Choose Target Window

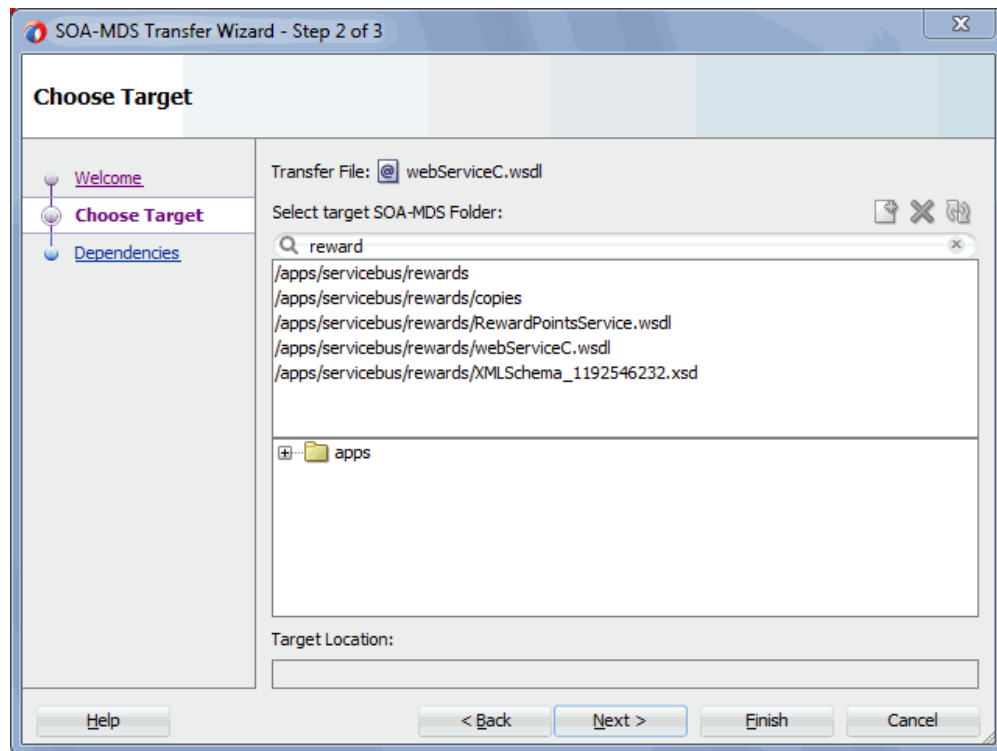


6. To add a new folder, do the following:
 - a. Select the folder in which you want to create the new folder.
 - b. Click the **Create Folder** icon.
 - c. Enter a name for the new folder and click **OK**.

The new folder appears in the tree.

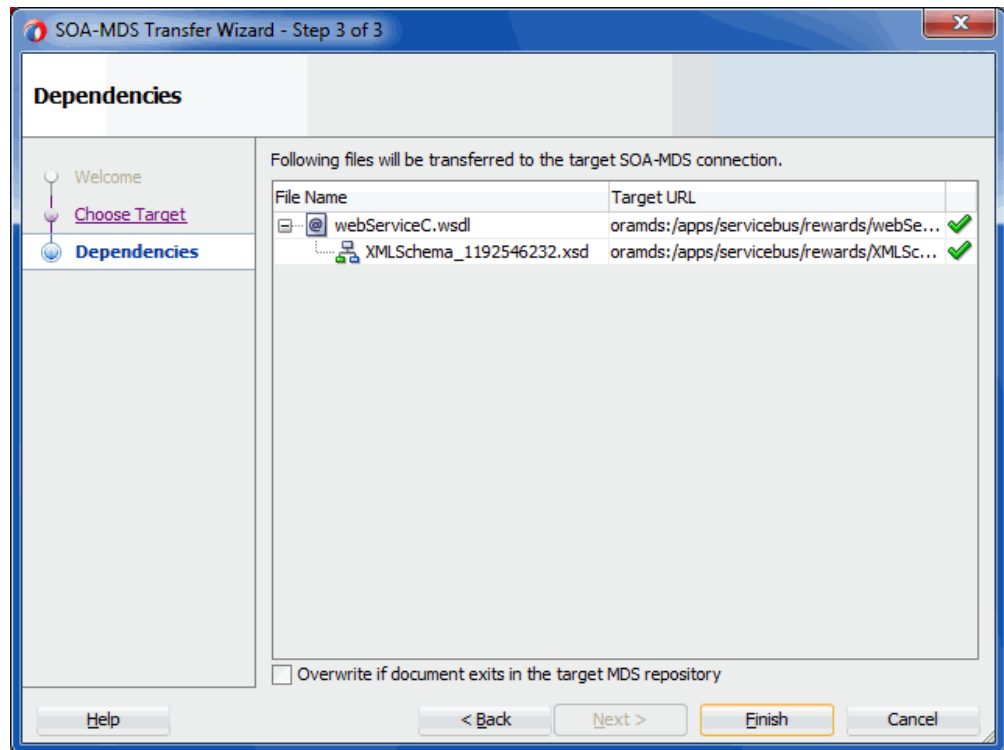
7. To search for a component or folder that exists in the MDS Repository, enter the full or partial name in the search field. The results appear below your input.

Figure 45-3 SOA-MDS Transfer Wizard - Search Targets



8. Select the folder to which you want to publish the selected component, and click **Next**.
9. On the Dependencies window, verify the files to copy to the MDS Repository.

This window displays the file you selected to copy along with any files on which that file depends.

Figure 45-4 SOA-MDS Transfer Wizard - Dependencies Window

10. To overwrite existing files in the MDS Repository, select **Overwrite if document exists in the target MDS repository**.
11. Click **Finish**.
12. On the confirmation dialog, click **OK**.

You can now view the files you copied in the SOA-MDS connection in the JDeveloper Resources window.

Consuming Artifacts Stored in the MDS Repository

When a Service Bus application consumes an artifact stored in the MDS Repository, it does not refer to the files in the repository like a SOA Suite applications does. Instead, the files are imported into the Service Bus application. You can consume artifacts stored in the MDS Repository directly from the Resources window in JDeveloper or by adding an artifact (such as a WSDL file) using the Select dialog. The Select dialog is accessed from the wizards and editors you use to create and configure components, such as business and proxy services.

How to Consume MDS Repository Artifacts Using the Resource Browser

The Resource Browser is most commonly launched in Service Bus when you are selecting a WSDL file on which to base a pipeline, split-join, proxy service, or business service. It also appears when you import a WSDL or XML file on the Type Chooser dialog, or when you import an XSLT or XQuery map into a pipeline action.

Before You Begin

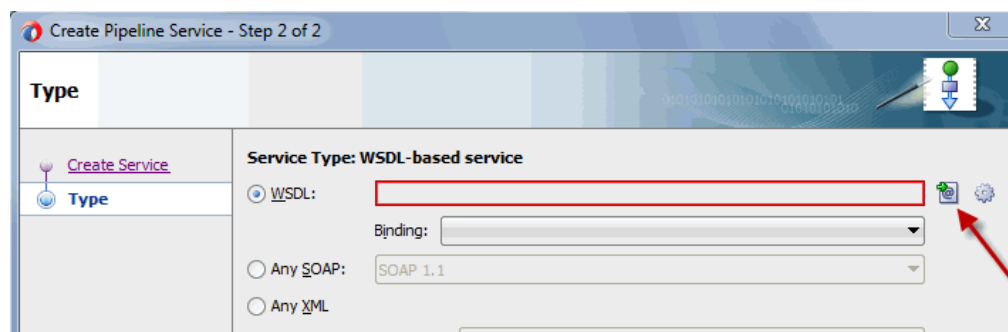
Before you can work with artifacts in the MDS repository, you need to create a connection to the repository from JDeveloper. For instructions, see [How to Create a SOA-MDS Connection](#).

To consume artifacts using the Resource Browser:

1. When you reach a point in creating or configuring a Service Bus component where you need to select a file from the MDS Repository, click the icon that lets you select an existing file.

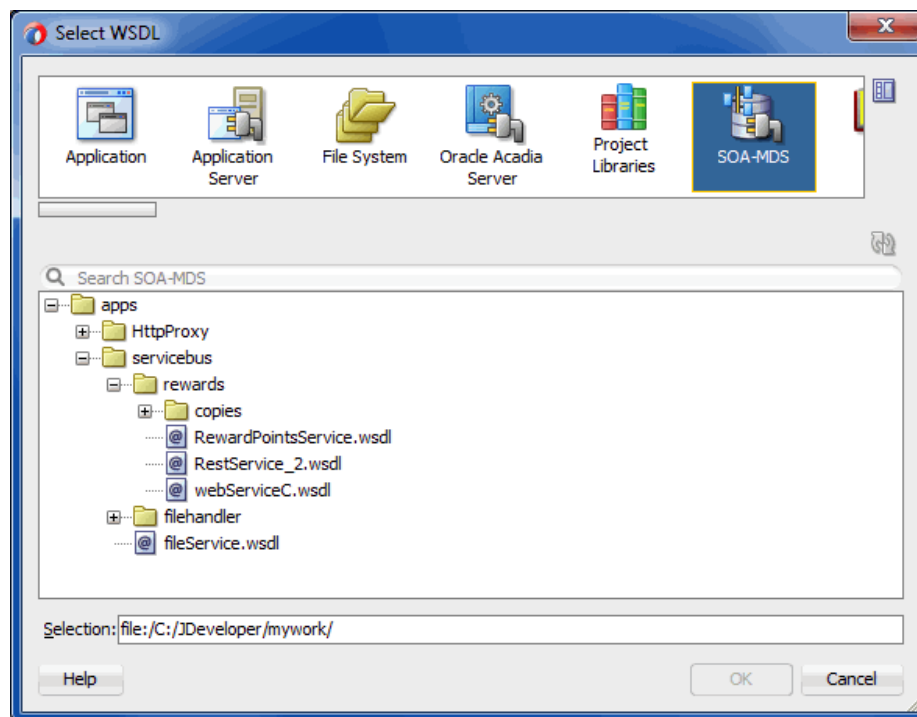
For example, when creating a proxy service based on a WSDL document, click **Select WSDL** to the right of the field.

Figure 45-5 Select WSDL Icon on the Create Pipeline Service Wizard



2. In the upper portion of the Resource Browser, select **SOA-MDS**.

Figure 45-6 Resource Browser



3. In the lower portion of the Resource Browser, expand the folders to select the file you want to use and then click **OK**.

The Import Service Bus Resources wizard appears (because Service Bus copies instead of references repository files).

4. Verify the information for the source file and make any necessary changes. Click **Next**.
5. On the Configuration window, make sure the artifacts you want to import are all selected. Click **Finish**.

The file and its dependencies are imported into the current project, and the file information is populated into the appropriate fields.

6. Continue creating or configuring the Service Bus component.

How to Add MDS Repository Artifacts to a Service Bus Project

Service Bus can only consume supported resource types from the MDS Repository. Any resource recognized as a Service Bus resource can be imported to a Service Bus project. When you import an artifact, Service Bus launches the Import Service Bus Resources wizard to add the resources to the selected project.

Before You Begin

Before you can work with artifacts in the MDS repository, you need to create a connection to the repository from JDeveloper. For instructions, see [How to Create a SOA-MDS Connection](#).

To add repository artifacts to a Service Bus project:

1. If the Resources window is not visible in JDeveloper, click the **Window** menu and select **Resources**.
2. In the Resources window, expand **SOA-MDS** and the repository name (the default name is **SOA_DesignTimeRepository**).
3. Expand the folders until you locate the artifact you want to add to your project.

Tip:

Alternatively, enter a full or partial name in the **Search** field on the Resources window and press **Enter** to search for the artifact.

4. Right-click the artifact, point to **Service Bus**, and select **Import Resource**.

The Import Service Bus Resources wizard appears.

5. Verify the information for the source file and make any necessary changes to the name and location for the imported file. Click **Next**.
6. On the Configuration window, make sure the artifacts you want to import are all selected. Click **Finish**.

The new resource is added to the location you specified and is available for use in Service Bus projects.

How to Create a Business Service from a WSDL File in the MDS Repository

You can generate a Service Bus business service from a WSDL document in the MDS Repository. When you select this option, the Create Business Service wizard appears so you can further define the business service. When the business service is generated,

the WSDL file is imported to the Service Bus project, along with any dependencies like XML schema files.

Before You Begin

Before you can work with artifacts in the MDS repository, you need to create a connection to the repository from JDeveloper. For instructions, see [How to Create a SOA-MDS Connection](#).

To create a business service from a WSDL file in the MDS Repository:

1. If the Resources window is not visible in JDeveloper, click the Window menu and select **Resources**.
2. In the Resources window, expand **SOA-MDS** and the repository name (the default name is **SOA_DesignTimeRepository**).
3. Expand the folders until you locate the WSDL file you want to use for the business service.

Tip:

Alternatively, enter a full or partial name in the **Search** field on the Resources window and press **Enter** to search for the WSDL file.

4. Right-click the WSDL file, point to **Service Bus**, and select **Generate Business Service**.

The Create Business Service wizard appears.

5. Click **Browse** next to the **Service Artifacts Folder** field, and browse to and select the project or folder to which you want to add the WSDL file. Click **Select**.
6. On the Create Business Service wizard, click **Next**.
7. On the Create Service window, accept the default values or make any of the following changes:
 - Modify the default name assigned to the service.
 - Add a description for the business service.
 - Select a different port from the WSDL document.
8. Click **Next**.
9. On the Transport window, accept the default values or make any of the following changes:
 - Select a new transport protocol.
 - Modify the endpoint URI.
10. Click **Finish**.

The new resources are added to the locations you specified.

11. Continue configuring the business service, as described in [Configuring Business Services](#).

How to Create a Business Service from a WADL File in the MDS Repository

You can generate a Service Bus business service from a WADL document in the MDS Repository. When you select this option, the Create REST Binding wizard appears so you can configure the REST service. The business service and WSDL file are generated, and the WADL file is imported to the Service Bus project, along with any dependencies, such as XML schema files.

Before You Begin

Before you can work with artifacts in the MDS repository, you need to create a connection to the repository from JDeveloper. For instructions, see [How to Create a SOA-MDS Connection](#).

To create a business service from a WADL file in the MDS Repository:

1. In the Application Navigator in JDeveloper, open the application to which you want to add the REST business service, and select the project or folder in which it will be located.
2. If the Resources window is not visible, click the Window menu and select **Resources**.
3. In the Resources window, expand **SOA-MDS** and the repository name (the default name is **SOA_DesignTimeRepository**).
4. Expand the folders until you locate the WADL file you want to use for the business service.

Tip:

Alternatively, enter a full or partial name in the **Search** field on the Resources window and press **Enter** to search for the WADL file.

5. Right-click the WADL file, point to **Service Bus**, and select **Generate Business Service**.

The Create REST Binding wizard appears.

6. Enter a name for the REST binding.
7. Enter the **Base URI**, which is the endpoint URI for the business service.
8. To specify that JSON payloads be reordered to match the order of elements in the XML schema, select **Enforce XMLSchema Ordering**.
9. The remaining configuration is based on the selected WADL file. Click **OK**.
The Import Service Bus Resources wizard appears, if there are resources to import.
10. Optionally, specify a new name and import location for the resource to import.
11. Click **Next**.
12. Review the summary of resources to import, and click **Finish**.

The WADL file, WSDL file, and any dependent resources are added to the Resources folder of the selected project. The business service is added to the project.

13. Continue configuring the business service, as described in [Configuring Business Services](#).

How to Expose a WSDL File in the MDS Repository as a REST Service

You can generate REST business and proxy services from a WSDL document in the MDS Repository. When you select this option, the Create Business Service wizard appears, followed by the Create REST Binding wizard. The Service Bus services are generated, and the WSDL and WADL files are imported to the Service Bus project, along with any dependencies, such as XML schema files.

Before You Begin

Before you can work with artifacts in the MDS repository, you need to create a connection to the repository from JDeveloper. For instructions, see [How to Create a SOA-MDS Connection](#).

To expose a WSDL file in the MDS Repository as a REST service:

1. In the Application Navigator in JDeveloper, open the application to which you want to add the REST services, and select the project or folder where they will be located.
2. If the Resources window is not visible, click the Window menu and select **Resources**.
3. In the Resources window, expand **SOA-MDS** and the repository name (the default name is **SOA_DesignTimeRepository**).
4. Expand the folders until you locate the WSDL file you want to expose as a REST service.

Tip:

Alternatively, enter a full or partial name in the **Search** field on the Resources window and press **Enter** to search for the WSDL file.

5. Right-click the WSDL file, point to **Service Bus**, and select **Expose as REST**.

The Create Business Service wizard appears.

6. Click **Browse** next to the **Service Artifacts Folder** field, and browse to and select the project or folder to which you want to add the resources and service. Click **Select**.
7. On the Create Business Service wizard, click **Next**.
8. On the Create Service window, accept the default values or make any of the following changes, and then click **Next**:
 - Modify the default name assigned to the service.
 - Add a description for the business service.
 - Select a different port from the WSDL document.
9. On the Transport window, accept the default values or make any of the following changes:
 - Select a new transport protocol.

- Modify the endpoint URI.

10. Click **Finish.**

The Create REST Binding wizard appears.

- 11.** Optionally, enter a new name and description for the REST service.
- 12.** To specify that JSON payloads be reordered to match the order of elements in the XML schema, select **Enforce XMLSchema Ordering**.
- 13.** To enter a new resource path, click the **Add** icon in the Resources section.
- 14.** If necessary, double-click in the HTTP Verb column of the Operation Bindings section to configure the methods.
- 15.** Click **OK**.
- 16.** If the Localize Files dialog appears, clear the check box if you do not want to maintain the original directory structure, and click **OK**.
- 17.** Continue configuring the business service, as described in [Configuring Business Services](#).
- 18.** Continue configuring the proxy service, as described in [Configuring Proxy Services](#).

Opening the Project Overview File Through a SOA-MDS Connection

If you create a SOA-MDS connection in JDeveloper, expand the connection, and attempt to open the `servicebus.sboverview` file of a Service Bus project or the `overview.xml` files of a SOA composite application from the Resources window, the file may not load correctly. Only open a these files from the Application Navigator.

For information about the Oracle MDS Repository, see *Administering Oracle Fusion Middleware*.

Working with UDDI Registries

This chapter describes how to use Service Bus with Universal Description, Discovery, and Integration (UDDI) registries.

This chapter contains the following sections:

- [UDDI, UDDI Registries, and Web Services](#)
- [Service Bus and UDDI](#)
- [Keeping Services Synchronized](#)
- [Related References](#)
- [Working with UDDI Registry Resources](#)
- [Sharing UDDI Registry Services in JDeveloper](#)
- [Sharing UDDI Registry Services in the Oracle Service Bus Console](#)
- [Sample Business Scenarios for Service Bus and UDDI](#)
- [Mapping Service Bus Proxy Services to UDDI Entities](#)

UDDI, UDDI Registries, and Web Services

UDDI provides a framework in which to classify your business, its services, and the technical details about the services you want to expose. The UDDI Project is an industry initiative that aims to enable businesses to find and carry out transactions with one another quickly, easily, and dynamically. A populated UDDI registry contains cataloged information about businesses, the services that they offer, and communication standards and interfaces they use to conduct transactions. A UDDI registry provides a standards-based foundation infrastructure for locating services, invoking services, and managing metadata about services (security, transport, or quality of service). The UDDI registry can store and provide these metadata using arbitrary categorizations. These categorizations are called taxonomies.

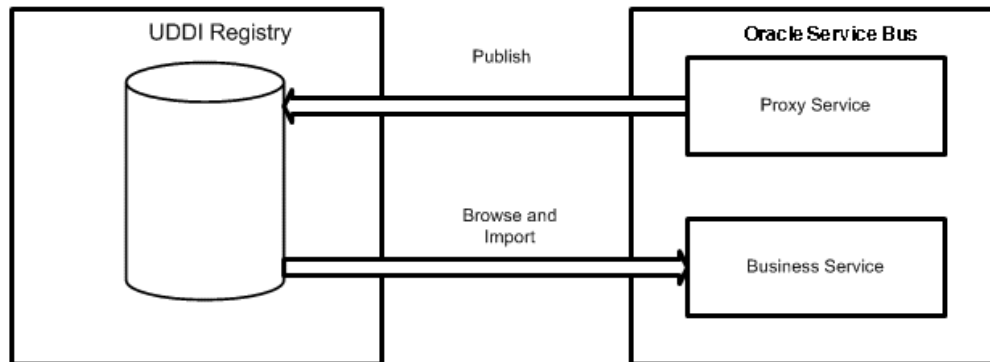
An organization uses UDDI registries to share web services. Using UDDI registries helps companies organize and catalog web services for sharing and reuse in an enterprise or with trusted external partners. The UDDI version 3.0 specification is available at: <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3>

UDDI registries are based on this specification, which provides details on how to publish and locate information about web services using UDDI. The specification does not define runtime aspects of the services (it is only a directory of the services). UDDI provides a framework in which to classify your business, its services, and the technical details about the services you want to expose.

Publishing a service to a registry requires knowledge of the service type and the data structure representing that service in the registry. Certain properties are associated with each registry entry and these property types are defined when the registry is created. You can publish your service to a registry and make it available for other organizations to discover and use. Proxy services developed in Service Bus can be published to a UDDI registry. Service Bus can interact with any UDDI registry that is compliant with version 3.0.

Figure 46-1 illustrates the integration of Service Bus with a UDDI registry.

Figure 46-1 Oracle Service Bus integration with UDDI



The Service Bus web-based interface makes the registry accessible and easy to use. In working with UDDI, Service Bus promotes the reuse of standards-based web services. In this way, Service Bus registry entries can be searched for, discovered, and used by multiple domains. web services and UDDI are built on a set of standards, so reuse promotes the use of acceptable, tested web services and application development standards across the enterprise. The web services and interfaces can be catalogued by type, function, or classification so that they can be discovered and managed more easily.

Basic Concepts of the UDDI Specification

UDDI is based upon several established industry standards, including HTTP, XML, XML Schema Definition (XSD), SOAP, and WSDL. The UDDI specification describes a registry of web services and its programmatic interfaces. UDDI itself is a set of web services. The UDDI specification defines services that support the description and discovery of the following:

- Businesses, organizations, and other web services providers
- The web services they make available
- The technical interfaces that can be used to access and manage those services

Benefits of Using a UDDI Registry with Service Bus

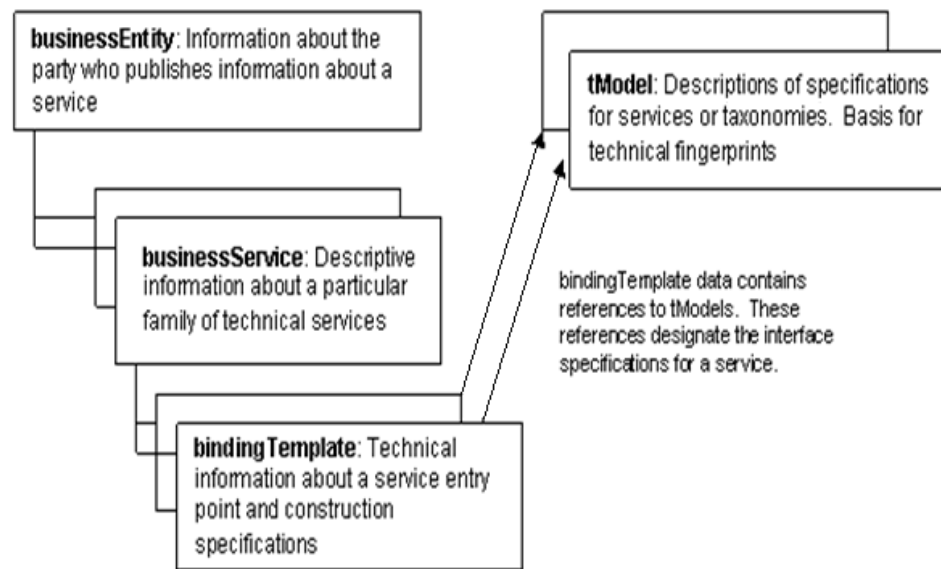
A UDDI registry stores data and metadata about business services. A UDDI registry offers a standards-based mechanism to classify, catalog, and manage web services so that they can be discovered and consumed by other applications. UDDI offers several benefits to IT managers and developers at both design time and runtime, including the following:

- UDDI improves infrastructure management by publishing information about services to the registry and categorizing the services for discovery. This ability of UDDI to categorize a growing portfolio of services makes it easier to manage them and helps you to understand relationships among components, supports versioning, and manages dependencies.
- You can import UDDI services from a registry to configure the parameters required to invoke the web service and the necessary transport and security protocols.
- UDDI promotes the use of standards-based web services and business services development in business applications and provides a link to a library of resources for web services developers. This decreases development time and improves productivity. It also increases the prospect of interoperability between business applications by sharing standards-based resources.
- UDDI provides a user-friendly interface for searching and discovering web services.

Introduction to UDDI Entities

UDDI uses a specific data model to represent entities that define organizations and services. [Figure 46-2](#) shows the relationships between different UDDI entities.

Figure 46-2 UDDI Entities Representing Organizations and Services



[Table 46-1](#) provides a high-level overview of UDDI entities.

Table 46-1 High-Level Description of UDDI Entities

UDDI Entity	Description
Business Entity	An organization or group that owns and provides the services. A business entity can be described by a set of names, descriptions, contact details for the service provider, a set of categories that represent the business entity features, unique identifiers, and discovery URLs.

Table 46-1 (Cont.) High-Level Description of UDDI Entities

UDDI Entity	Description
Business Service	A description of the functionality or resources provided by a business entity. A business service is described by a name, a description, and a set of categories that represent the function of the service. A business service in a UDDI registry does not necessarily represent a web service. The UDDI registry can register arbitrary services, for example EJB, CORBA, and such.
Binding Template	The technical details of how to invoke a business service. A business service can contain one or more binding templates. Binding templates are described by access points representing service endpoints (the endpoint URI and protocol specification), tModel instance information, and categories to reference specific features of the binding template.
tModel	A technical specification; typically a specifications pointer, or metadata about a specification document, describing how services must be represented in the UDDI registry. The description of a service includes a name, a description, an overview document (a reference to a document specifying the purpose of the tModel), a category, and an identifier (to uniquely identify the tModel).

Service Bus and UDDI

Service Bus works with any UDDI registry that is compliant with the version 3.0 implementation of UDDI. Using Service Bus with a UDDI registry, you can do the following:

- Configure Service Bus to work with one or more V3.0-compliant UDDI registries.
- Configure a registry to allow users to publish services and import services.
- Publish information about any proxy service to a registry.
- Search for specific services in a registry or list all services available. You can search on business entity, service name pattern, or both.
- Import business services from a registry.

UDDI Registry URLs

When you configure a UDDI registry in Service Bus, you specify several API endpoint URLs for different types of UDDI registry access. These URLs include the following:

- Inquiry URL: The URL of the Inquiry API endpoint used for locating and importing services.
- Publish URL: The URL of the Publish API endpoint used for publishing services.
- Security URL: The URL of the Security API endpoint used for getting an authentication token so you can publish to the registry
- Subscription URL: The URL of the Subscription API endpoint used for subscribing to registry changes, creating a registry subscription, and listening for changes to imported services.

UDDI Registry Security Configuration

You can make your UDDI registries available in Service Bus by creating a UDDI registry resource, which defines the connection information for the UDDI registry. You can then publish Service Bus proxy services to the registry or import business services from the registry to be used in a proxy service. You specify the UDDI URLs in the resource, and optionally specify security information. When publishing services to most registries, the proxy service configuration must include a valid user name and password for authentication to gain access to the registry.

You can set up registries with multiple user names and passwords allowing different users to have different permissions based on the associated service accounts. In Service Bus, user permissions govern access to the registries, their content, and available functionality.

Authentication Configuration and UDDI Registries

When a proxy service is published to a UDDI registry, the service is converted into a WS business service with the WSDL document. If present, the authentication configuration is also exported to UDDI. When a WSDL-based business service with WSRM policy is imported from an UDDI registry to Service Bus, the service is imported as a WS business service that is automatically configured to use the WS transport.

Transport-level custom authentication tokens are published to the UDDI. The `client-auth` property is present in the `instanceParms` of the HTTP or HTTPS transport attributes whenever authentication is configured. As described in [Transport Attributes](#), the possible values of `client-auth` are `BASIC`, `CLIENT-CERT` and `CUSTOM-TOKEN`. Whenever the value is `CUSTOM-TOKEN`, two additional properties are present: `token-header` and `token-type`.

Note:

Service Bus business service definitions do not support custom token authentication. If you import a service from UDDI that has `client-auth` equal to `CUSTOM-TOKEN`, the service is imported as if it does not have any authentication configuration.

About Publishing Proxy Services to a UDDI Registry

Use the Oracle Service Bus Console or JDeveloper to publish proxy services to a UDDI registry and make it available for other organizations to discover and use. All proxy services developed in Service Bus can be published to a UDDI registry. You can select the business entity under which you want to publish your service and you can publish a number of services at a time.

To do this you must have a user account set up in the UDDI registry. You can publish any proxy service to a UDDI registry and you can select the Business Entity under which a service is to be published. Business Entity Administration (including creation, removal, update, and deletion of entities) is done using the management console provided by the registry vendor. The first time you publish to a registry you must load the tModels to that registry. You do this when you configure the publishing details in the console or JDeveloper. For more information on how to publish to a UDDI registry, see [Publishing Proxy Services to a UDDI Registry](#).

Note:

An error can occur when you attempt to import a service from a UDDI registry if that service was originally published to the registry from a Service Bus cluster in which *any* of the clustered servers uses the localhost address. Specifically, when the service being imported references a resource (WSDL or XSD) which references other resources (WSDL or XSD).

Before you publish services to a UDDI registry from a clustered domain, be sure that none of the servers in the cluster use localhost in the server addresses. Instead, use either the machine name or the IP address.

You can publish local proxy services to a UDDI registry in order to associate them with Service Bus generic proxy services. For example, you might have an any SOAP or any XML generic proxy service that dynamically routes to multiple local transport proxy services with concrete WSDL files. Alternatively, you might have a generic proxy service in Service Bus 1 that dynamically routes to a generic proxy service in Service Bus 2 where the business service is attached. From the UDDI registry, you can get the WSDL file of the local proxy service and the URL of the any SOAP or any XML generic proxy service. Combining the WSDL file and URL creates an effective WSDL file for sending messages to the local proxy service through the generic proxy service.

About Importing Services from a UDDI Registry

You can import services from a UDDI registry as Service Bus business services. When importing a WSDL-based service, if multiple UDDI binding templates are encountered, Service Bus creates a different business service for each binding template.

For information about the security roles required to establish access to UDDI registries in Service Bus, see "Role-Based Access in Oracle Service Bus" in *Administering Oracle Service Bus*. When importing, you select from the list of available registries. In the Oracle Service Bus Console, you can view the complete list of registries on the UDDI Folder Definition Editor. When you import from a registry, you discover services by querying that registry. In JDeveloper, you can view the available registries in the Resources window and browse through the list to discover a service. Entries in registries are unique.

You can import the following business services types from a UDDI registry into Service Bus:

- WSDL over HTTP binding. When multiple UDDI binding templates are present, a business service is created for each binding template.
- SOAP or XML binding over HTTP.
- Services that are categorized as Service Bus services. These are proxy services that are published to a UDDI registry. This feature is primarily used in multi-domain Service Bus deployments where proxy services from one domain need to discover and route to proxy services in another domain.

Services have documents associated with them, and those documents can include a number of other documents (schemas, policies, and so on). On import, the UDDI registry points to the document location based on the inquiry URL of the service. When a document that includes or references other resources is located, all of the referenced information and each included item is added as a separate resource in Service Bus.

About Business Entities and Patterns

Business entity and pattern are the criteria used to search for a service in a registry. Services published by Service Bus have specific tModel keys identifying the services that you use when searching for the service in the registry. The business entity is the highest level of organization in the registry, though you can use other search criteria, such as business, application type, and so on. If you require authentication, then you need a user name and password, which you must get from your system administrator.

Keeping Services Synchronized

You can keep the service definitions in Service Bus automatically synchronized (both ways) with those in UDDI. Services can be automatically published to a UDDI registry after they are created or changed within Service Bus, and business service definitions can be imported from UDDI and automatically updated when the original service is changed in UDDI. Alternatively, you can configure the Oracle Service Bus Console or JDeveloper to prompt you for approval for synchronization when a service changes in the UDDI registry.

Automatic Publishing for Proxy Services

When you configure a proxy service in the Oracle Service Bus Console, you can configure it to be published automatically to a default UDDI registry. This feature is not available in JDeveloper. You must first set up a default registry and configure the proxy service to automatically publish to the default registry. When you activate these changes, the proxy service is published to the default registry. If the UDDI registry is unavailable, the publish action is retried. Any further changes to the proxy service resets the retry attempts. When a proxy service is republished to a UDDI registry, all taxonomies and categorizations, which are defined in UDDI for the proxy service, are preserved.

For instructions, see [How to Specify a Default UDDI Registry Resource](#) and [How to Automatically Publish Proxy Services to a UDDI Registry](#).

Changes to the Default Registry

When you change the default registry, all the proxy services that have auto-publish enabled are published to the new default registry. Synchronization then takes place with the current default registry. When a proxy service is not synchronized, the Oracle Service Bus Console displays an unsynchronized icon.

Note:

When you have a default registry and you import a configuration JAR file that has a default registry set with the same logical name during the import, it is possible that the default registry will have an incorrect value for the business entity. This might result in errors on the Auto Publish Status page if there are any auto-published proxy services. You can correct this situation by selecting the default registry again.

Auto-Publish Synchronization Process

When auto-publish is enabled for a proxy service, you can use the Auto Publish Status page on the Oracle Service Bus Console to view and manage the service synchronization process. This page displays a list of published proxy services along

with their status. If any errors prevent a service from automatically publishing the registry, you can retry publishing them from this page.

If a proxy service changes after being published, you can synchronize the changes using the Admin features in the Oracle Service Bus Console. If auto-publish is enabled, Service Bus automatically publishes any changes to the service in the registry. In addition, the Auto Publish Status dialog shows this service and provides options for publishing the service to the registry.

Automatic Importing of UDDI Services

You can use the auto-import feature to synchronize the business services that were imported from a UDDI registry with the corresponding services in the registry. For instructions, see [How to Automatically Synchronize Imported Services](#).

Note:

Auto-import is available only in the Oracle Service Bus Console, and not in JDeveloper.

When a service is updated, you must re-import the service from the registry to get the most recent version unless auto-import is enabled in the UDDI registry resource. When the **Enable Auto-Import** option is selected, any service that is imported is automatically kept synchronized with the UDDI registry. Any failure that occurs during auto-synchronization is reported on the Auto-Import Status page where you can synchronize it manually.

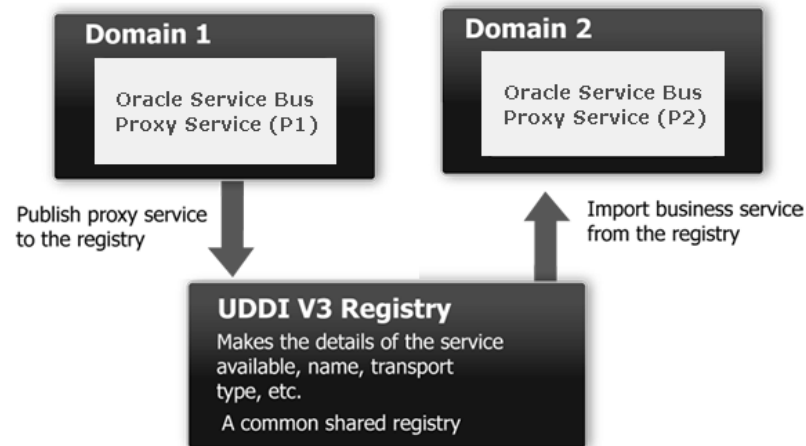
When auto-import is enabled, you can use the Auto Import Status page to view and manage the service synchronization process. You can either synchronize a service with the registry or unlink the service to avoid synchronization on this page.

Synchronization of Imported Services

If the services you have imported from a UDDI registry are changed in the registry change, you can synchronize the services with those in the registry using the Admin features in the Oracle Service Bus Console. If auto-import is enabled and a business service is not unlinked from the registry, Service Bus automatically subscribes to any changes to the service in the registry. The Auto Import Status dialog shows this service and provides options for synchronizing the service or unlinking it from the registry. Under certain circumstances, synchronizing the service might result in semantic validation errors, which could prevent session activation. In that case, unlinking might be a better choice.

When a service is synchronized, the service is updated only with fields that are obtained from UDDI. Other fields in the service definition will preserve their values if modified since last import. This includes policy configurations.

Consider a scenario where you publish services from Domain1 to a registry. You then import these services from the registry into a domain, Domain2 (see [Figure 46-3](#)). Then you make changes to the services in Domain1 and update them in the registry. You can update the services in Domain2 by synchronizing them with the registry using the auto-import feature.

Figure 46-3 Sample Business Case of Cross-Domain Deployment

Unlinking Imported Services

There may be times when you do not want the service in the Oracle Service Bus Console to be synchronized with the corresponding service in the registry. You can avoid synchronization by unlinking the service from the registry. See [How to Unlink an Imported Service From the UDDI Registry](#).

Related References

The following documents provide additional UDDI information:

- Technical Notes can be found at <http://www.oasis-open.org/committees/uddi-spec/doc/tns.htm>. The note on *Using WSDL in a UDDI Registry* is important.
- UDDI product and development tool information is available at the OASIS UDDI Solutions page at <http://uddi.org/solutions.html>.
- The UDDI specifications <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm>

The specification defines the following:

- SOAP APIs that applications use to query and to publish information to a UDDI registry
- XML schema of the registry data model and the SOAP message formats
- WSDL definitions of the SOAP APIs
- UDDI registry definitions (tModels) of various identifier and category systems that may be used to identify and categorize UDDI registrations

Working with UDDI Registry Resources

In order to access a UDDI registry in Service Bus, you need to create and configure a UDDI registry resource, which describes the registry. Publishing a service to a registry requires knowledge of the service type and the data structure representing that service in the registry. A registry entry has certain properties associated with it and these property types are defined when the registry is created. You can publish your service

to a registry and make it available for other organizations to discover and use. Proxy services developed in Service Bus can be published to a UDDI registry.

How to View UDDI Registry Resources in the Oracle Service Bus Console

The Folders Definition Editor for UDDI registries lists all the UDDI registry resources you have created in the current session. Use this page to quickly find and access the UDDI registry resources you have defined.

To view UDDI registries in the console:

1. Expand the **System** project, right-click **UDDI**, and then select **Open**.

The Folder Definition Editor appears with a list of existing UDDI registry resources.

2. To locate specific UDDI registry resources, do the following:
 - If the query fields are not visible above the UDDI table, click **Query by Example** in the table toolbar.
 - Enter the name of the UDDI registry resource you want to find above the **Name** column, and press **Enter**.
You can enter wildcard characters (? for a single character; * for multiple characters) to perform a more general search.
 - To view all UDDI registry resources again, clear the query fields and press **Enter**.
3. To view the configuration for a UDDI registry, click the resource name in the UDDI table.
4. To delete a UDDI registry resource, select the name of the resource in the table and click **Delete**. See [How to Delete a UDDI Registry Resource](#).

How to Create UDDI Registry Resources

When you create a UDDI registry resource, you specify connection information for the remote server, including URLs, security credentials, and whether to automatically synchronize services. After you create a registry, you can then publish Service Bus proxy services to them or import business services from them to be used in a proxy service.

To create a UDDI registry resource:

1. Do one of the following:
 - If you are using JDeveloper, expand the Application Resources panel, right-click **Service Bus System Resources**, point to **New**, and then select **UDDI Registry**.

Note:

To create UDDI registry resources directly in a project, making it a project-level resource, right-click the project, point to **New**, and then select **UDDI Registry**.

- If you are using Oracle Service Bus Console, expand the **System** project, right-click **UDDI**, point to **Create**, and then select **Create UDDI Registry**.

The Create UDDI Registry dialog appears.

2. Enter a name and optional description for the resource, and then click **Finish** or **Create**.

The UDDI Registry Definition Editor appears and the new UDDI registry resource appears in the Systems folder of the Oracle Service Bus Console or in the Service Bus System Resources folder in the Application Resources panel of JDeveloper.

3. Enter the inquiry, publish, subscription, and security URLs for the UDDI registry. For more information, see [UDDI Registry URLs](#).

4. To publish the Service Bus tModels to the registry, select **Load T-Models into Registry**.

This field is only required when publishing proxy services to this registry.

5. To automatically synchronize services with the UDDI registry, select **Enable Auto-Import**.

Any service that is imported with this option selected will be kept in synchrony with the UDDI registry.

Note:

Auto-synchronization is a background process; you cannot reverse it using the session **Undo** function. Undoing an auto-synchronization change is not permanent as the service will be re-synchronized in the next synchronization cycle. If you want an imported service to stay out of synchrony with the UDDI registry, you have to detach the service to avoid further updates from the registry. See [How to Unlink an Imported Service From the UDDI Registry](#).

6. If access to the UDDI registry console requires a user name and password, enter a user name in the **User Name** field, and the associated password in the **Password** and **Confirm New Password** fields.
7. Click **Save**.
8. Test the UDDI URLs by doing one of the following:
 - a. If you are using JDeveloper, click **Test Connection**.
 - b. If you are using the Oracle Service Bus Console, click **Test and Validate UDDI Registry**.
9. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Create a UDDI Registry Resource from a JDeveloper UDDI Connection

In JDeveloper, you can create a UDDI registry resource from an existing UDDI registry connection. Conversely, you can also create a UDDI registration connection from a UDDI registry resource.

To create a UDDI registry resource from a UDDI connection:

1. In the toolbar, click **Window** and then select **Resources** to display the **Resources** window.
2. Expand **IDE Connections** and **UDDI Registry**.
3. Right-click a UDDI connection, point to **Service Bus**, and select **Create Service Bus UDDI Registry Resource**.

The **Create UDDI Service** dialog appears.

4. Enter a name for the resource, or accept the default, and click **Finish**.

The **UDDI Registry Definition Editor** appears with the inquiry URL already populated, based on the URL specified for the UDDI registry connection.

5. Complete the remaining fields, as described in [How to Create UDDI Registry Resources](#).

How to Edit a UDDI Registry Resource

Once you create a UDDI registry resource, you can modify its description and most of the UDDI properties.

To edit a UDDI registry resource:

1. Expand the project and folders containing the resource to edit. This can be any of the following locations:
 - In JDeveloper, the **Service Bus System Resources** folder in the **Application Resources** panel.
 - In JDeveloper, the **Service Bus** project or folder in which the UDDI registry resource is located in the **Application Navigator**.
 - In Oracle Service Bus Console, the **UDDI** folder in the **System** project.
2. Right-click the UDDI registry name, and select **Open**.

The **UDDI Registry Definition Editor** appears.
3. Modify any of the fields described in [How to Create UDDI Registry Resources](#). The online help describes these fields in greater detail.
4. When you are done making changes, click **Save**.
5. Test the UDDI URLs by doing one of the following:
 - a. If you are using JDeveloper, click **Test Connection**.
 - b. If you are using the Oracle Service Bus Console, click **Test and Validate UDDI Registry**.
6. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Specify a Default UDDI Registry Resource

You can designate one of the UDDI registries that has already been configured and activated as the default registry for the domain. To use the auto-publish functionality,

you must first set a default registry. For more information, see [Automatic Publishing for Proxy Services](#).

To specify a default UDDI registry resource:

1. Expand the project and folders containing the resource to edit. This can be any of the following locations:
 - In JDeveloper, the **Service Bus System Resources** folder in the Application Resources panel.
 - In JDeveloper, the Service Bus project or folder in which the UDDI registry resource is located in the Application Navigator.
 - In Oracle Service Bus Console, the **UDDI** folder in the **System** project.
2. Right-click the UDDI registry resource, and select **UDDI Settings**.

The UDDI Settings dialog appears.
3. In the **Default UDDI Registry** list, select the name of the registry you want to set as the default registry.

Note:

The list displays all UDDI registries you have created, but you can only select one that has already been activated.

4. In the **Default Business Entity** list, select the name of the business entity you want to set as the default entity. This is optional.
5. Click **OK**.
6. When you are done making changes, click **Save**.
7. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

How to Delete a UDDI Registry Resource

When you delete a UDDI registry resource, any references to the resource from other Service Bus resources are broken. Before deleting the resource, check for dependencies. In the Oracle Service Bus Console, open the UDDI registry resource in the UDDI Registry Definition Editor and click the **References** icon in the upper right. In JDeveloper, right-click the UDDI registry and select **Explore Dependencies**.

To delete a UDDI registry resource:

1. Expand the project and folders containing the resource to edit. This can be any of the following locations:
 - In JDeveloper, the **Service Bus System Resources** folder in the Application Resources panel.
 - In JDeveloper, the Service Bus project or folder in which the UDDI registry resource is located in the Application Navigator.
 - In Oracle Service Bus Console, the **UDDI** folder in the **System** project.

2. Right-click the UDDI registry resource, and select **Delete**.
3. If you are using JDeveloper, a confirmation dialog displays the number of references for the resource. Click **Show Usages** to view information about the references, and then click **Yes** to confirm that you want to delete the resource.
4. If you are using the Oracle Service Bus Console, click **Activate** to end the session and deploy the configuration to the runtime.

Sharing UDDI Registry Services in JDeveloper

In JDeveloper, you can create business services from services located in a UDDI registry, or you can simply download a service to a Service Bus project.

How to Create a UDDI Registry Connection in JDeveloper

When working with UDDI registries in JDeveloper, you need to create a JDeveloper UDDI registry connection and a Service Bus UDDI registry resource. The registry connection lets you access and browse the registry in the Resources window and in the various selector dialogs that let you browse to and select artifacts for your projects. The registry resource specifies the API endpoint URLs and security information for the registry.

You can create a UDDI registry connection using JDeveloper's New Gallery, or you can create the connection from an existing UDDI registry resource in the Application Resources panel. The following instructions describe how to create the connection from the resource.

Before You Begin:

Before you can work with a registry in JDeveloper, you must have an account with that registry. Service Bus supports interoperability with UDDI registries that are compliant with the version 3.0 specification.

To create a UDDI connection in JDeveloper:

1. If it does not already exist, create the UDDI registry resource, as described in [How to Create UDDI Registry Resources](#).
2. In the Application Resources panel, expand **Service Bus System Resources**.
3. Right-click the UDDI registry resource for which you want to create a connection, point to **Service Bus**, and click **Create UDDI Connection**.

The new connection is created based on the configuration of the UDDI registry resource.

4. To display the Resources window and access the connection, click **Window** in the toolbar and then select **Resources**. Expand **IDE Connections** and **UDDI Registry**.
5. To modify the connection properties and test the connection, right-click the connection and select **Properties**.

The Edit UDDI Registry Connection wizard appears, where you can modify the inquiry endpoint URL and the view, and you can test the connection.

How to Create a Business Service from a UDDI Registry Service

In JDeveloper, you can create business services from services stored in a UDDI registry. You cannot publish services to the UDDI registry from JDeveloper. When you perform these steps, the business service is created in the location you specify in the current application.

To create a business service from a UDDI registry service:

1. If the Resources window is not visible, click **Window** and then select **Resources**.
2. In the Resources panel, expand **IDE Connections** and **UDDI Registry**.
3. Locate the UDDI registry containing the web service you want to access, and browse through the nodes until you find the service.

Tip:

Alternatively, you can perform a search in the Resources window for the service to use.

4. Under the service name, expand **Binding Templates**, until you see the binding you want to use.
5. Right-click the binding, point to **Service Bus**, and then select **Generate Business Service**.

The Consume Service wizard appears. The service WSDL file and endpoint are automatically configured based on the service you selected.

6. By the **Service Artifacts Folder** field, click **Browse** to navigate through the current application and select a project or folder in which to create the business service.

For more information at any time, press **F1** or click **Help** from within the Create Business Services wizard.

7. Click **Next**.
8. On the Create Service page, enter a name for the business service in the **Service Name** field.
9. Optionally add a description and update the location on the file system (this location must be somewhere within the application folder). The WSDL information is automatically configured.
10. Click **Next**.
11. On the Transport page, specify the transport to use for the business service. The available options vary depending on the type of service selected.
12. Update the endpoint URI if necessary.
13. Click **Finish**.

The business service is created, and the Business Service Definition Editor appears so you can finish configuring the service. For more information, see [Creating and Configuring Business Services](#).

How to Download a Service From a UDDI Registry

If you want to use a specific service in a Service Bus project, you can download the service and any related files to the project.

To download a service from a UDDI registry:

1. If the Resources window is not visible, click **Window** and then select **Resources**.
2. In the Resources panel, expand **IDE Connections** and **UDDI Registry**.
3. Locate the UDDI registry containing the web service you want to access, and browse through the nodes until you find the service.

Tip:

Alternatively, you can perform a search in the Resources window for the service to use.

4. Under the service name, expand **Binding Templates**, until you see the binding to use.
5. Right-click the binding, point to **Service Bus**, and then select **Download**.

The Import Service Bus Resources wizard appears. The resource type and source URL are automatically configured based on the service you selected.

6. In the **Resource Name** field, enter a new name for the service, or accept the existing name.

For more information at any time, press **F1** or click **Help** from within the Import Service Bus Resources wizard.

7. By the **Import Location** field, click **Browse** to navigate through the current application and select a project or folder in which to download the service.
8. Click **Next**.
9. Verify the resources to be downloaded and then click **Finish**.

The service is added to the project or folder you specified, and the WSDL Editor appears.

Sharing UDDI Registry Services in the Oracle Service Bus Console

The Oracle Service Bus Console provides several options for publishing and importing from UDDI registries, including manual procedures and automatic processing.

Note:

If you need to unpublish a service from a registry, this is done from the UDDI registry.

Before You Begin:

Before you can publish to or import from a registry, you must have an account with that registry. Service Bus supports interoperability with UDDI registries that are compliant with the version 3.0 specification.

Publishing Proxy Services to a UDDI Registry

You can publish any Service Bus proxy service to a UDDI registry so it is available for other organizations to discover and use. When you publish a service, you can optionally select the business entity under which the service is published and you can publish a number of services at one time. You can only publish from the Oracle Service Bus Console, and not JDeveloper.

Note:

If the service is not successfully published it can be re-published. To re-publish a service, select the service on the Auto-Publish Status page and click **Publish**.

If the **Publish to Registry** option is enabled, the proxy services are published as soon as they are created or edited *and* the session is activated. You can use the **Publish to Registry** option with all proxy services, except those using the Local Transport.

How to Automatically Publish Proxy Services to a UDDI Registry

You can automatically publish proxy services to the default UDDI registry that you configure for a session. In order to enable automatic publishing, you need to enable auto-publishing on the service and you need to define the default UDDI registry so Service Bus knows which UDDI registry to publish services to.

To automatically publish a proxy service to a UDDI registry:

1. Specify a default UDDI registry to which proxy services will publish.
For instructions, see [How to Specify a Default UDDI Registry Resource](#).
2. In the Oracle Service Bus Console's Project Navigator, expand the project and folders containing the proxy service to configure.
3. Right-click the proxy service name, and select **Open**.
4. Under UDDI on the General page, select **Auto Publish to Registry**.

Note:

if there is no UDDI section, there is no default UDDI registry specified. For more information, see [How to Specify a Default UDDI Registry Resource](#).

5. Click **Save**.

For more information about editing proxy services, see [Configuring Proxy Services](#).

6. Click **Activate** to end the session and deploy the configuration to the runtime.
7. To verify that the proxy service was published, click the Admin tab and click **Auto-Publish Status**.

The proxy service should appear in the list of published services.

How to Manually Publish a Proxy Service to a UDDI Registry

You can only publish services to a UDDI registry when you are not in a session. Exit your session to enable UDDI publishing and access the registries list.

To manually publish a proxy service to a UDDI registry:

1. Right-click a folder or project, point to **Export**, and then select **Publish to UDDI**.

The Publish to UDDI dialog appears.

2. In the **Registry Name** field, select the UDDI registry to which the services will be published.
3. In the **Business Entity** field, select the business entity in the UDDI registry under which the services will be classified.
4. In the Proxy Services table, specify which proxy services to export to the registry.

By default, all proxy services in the session are selected. Clear the check boxes for any services you do *not* want to export.

5. Click **Publish**.

The Publish Summary page appears, and indicates whether the services were published successfully. It also lists any messages generated during the publishing process.

6. To publish another set of proxy services, click **Publish Another**. Otherwise, click **Close**.

How to Import Resources from a UDDI Registry

You can import the following business service types from a UDDI registry into the Oracle Service Bus Console:

- WSDL services over HTTP transport.
- Service Bus proxy services that are published to a UDDI registry. This feature is primarily used in multi-domain Service Bus deployments where proxy services from one domain need to discover and route to proxy services in another domain.

You can import multiple resources from a UDDI registry at one time.

Before you begin:

In order to access the UDDI registry to import resources from, you must create a UDDI resource in Service Bus. For more information, see [How to Create UDDI Registry Resources](#).

To import resources from a UDDI registry in the console:

1. Do one of the following:
 - Right-click the folder or project where you want to import the resource, point to **Import**, and then select **From UDDI**.
 - Right-click **All Projects**, point to **Import**, and then select **From UDDI**. On the Destination page of the Import from UDDI dialog, select the project or folder where you want to import the service and then click **Next**.

The Import from UDDI dialog appears with the Service page displayed.

2. In the **Registry Name** field, select the UDDI registry from which the services will be imported.
3. In the **Business Entity** field, select the business entity in the UDDI registry under which the services are located. Select **All** to search all business entities in the registry.

4. In the **Service Name** field, enter the name of a service to import.

Searches accept wildcard characters ("*" for multiple characters and "%" for a single character).

5. Click **Search**.

A list of matching business services appears in the Business Services table. If you are unable to find a specific service, it may be because you do not have the security permissions to view its records.

6. In the Business Services table, select the business services to import from the registry.

7. Click **Next**.

8. If a selected business service has multiple binding templates, the Binding Template page appears. For each listed service, select one binding template to use to create the business service.

Note:

If a selected service has multiple binding templates, each binding template results in a business service. The Binding Template page lets you further narrow your selection among the binding templates you want to import.

9. Click **Next**.

The Review page appears, displaying a summary of the resources to be imported. A warning message is displayed for any resource that cannot be imported.

10. Verify the list of resources selected to be imported, and then click **Import**.

The selected resources appear in the Project Navigator. The Import from UDDI wizard displays a summary of the import along with any errors in the imported resources.

11. To import another set of business services, click **Import Another**. Otherwise, click **Close**.

The import process can result in dependency issues. To view and resolve any resulting conflicts, click the Conflicts tab at the bottom of the console.

How to Automatically Synchronize Imported Services

The Auto-Import Status page lets you synchronize changes to a service with those present in the registry. Any service imported while automatic import is enabled is kept synchronized with the UDDI registry. Upon any changes to a service in the

registry, Service Bus provides notification of the change on the Auto-Import Status page, which lists all out-of-sync services.

To enable automatic import for a UDDI registry:

1. In the Oracle Service Bus Console's Project Navigator, expand the System project and UDDI folder.
2. Right-click the UDDI registry resource name, and select **Open**.
3. Select **Enable Auto-Import**.
4. Click **Save**.

Any services imported from a UDDI registry are kept synchronized with those in the UDDI registry. You can view the status of synchronized services on the Auto-Import Status page.

How to Manually Synchronize an Imported Service

When automatic import is enabled and there are any failures during auto-synchronization, the errors are reported on the Auto-Import Status page. After fixing the errors, you can synchronize the services manually.

To synchronize a service with the UDDI registry:

1. In the Oracle Service Bus Console, click the Admin tab and then click **Auto-Import Status**.

The Auto-Import Status dialog appears.

2. In the list of services, select the check box next to the service you want to synchronize to the UDDI registry.
3. Click the **Synchronize** icon above the table.

How to Unlink an Imported Service From the UDDI Registry

When you do not want the service in the Oracle Service Bus Console synchronized with the corresponding service in the registry, you can avoid synchronization by detaching it from the registry.

Unlinking a business service from the UDDI registry cannot be undone. You have to re-import the service manually to link them back up.

To unlink an imported service from the UDDI registry:

1. In the Oracle Service Bus Console, click the Admin tab and then click **Auto-Import Status**.

The Auto-Import Status dialog appears. Services are shown on this page only when there is a change in the original service in the registry. Not every service is available on this page.

2. In the list of services, select the check box next to the service you no longer want to be linked to the UDDI registry.
3. Click the **Unlink** icon above the table.
4. On the Warning dialog that appears, click **OK** to unlink the service, or click **Cancel** if you no longer want to unlink the service.

- Click **Close** on the Auto-Import Status dialog.

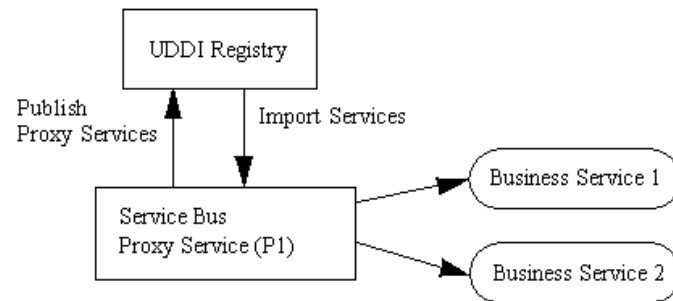
Sample Business Scenarios for Service Bus and UDDI

The following are two sample business scenarios that highlight the benefit of using UDDI.

Basic Proxy Service Communication with a UDDI Registry

This scenario illustrated using Service Bus to import services from a registry and then publish Service Bus proxy services back to the registry. See [Figure 46-4](#).

Figure 46-4 Proxy Service Communication with a UDDI Registry

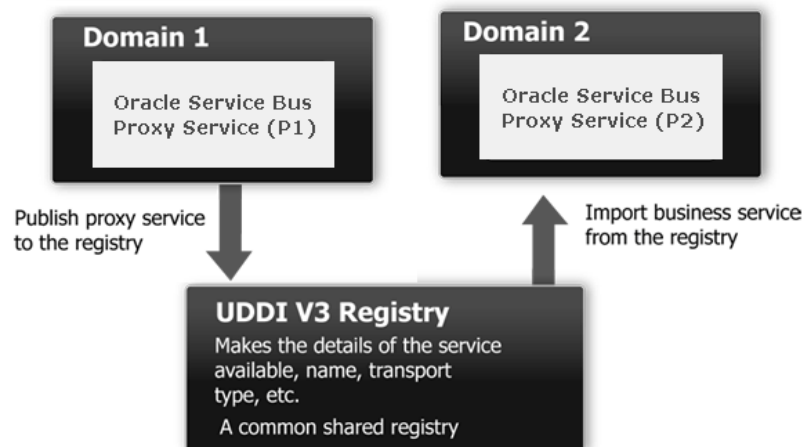


Service Bus imports business services from a UDDI registry. Proxy services are configured to communicate with the business services in the pipeline. The proxy services can then be published back to the registry and made available for use by other domains.

Cross-Domain Deployment in Service Bus

This scenario shows cross-domain deployment using Service Bus. In this scenario, a Service Bus application in one domain requires access to a Service Bus service in another domain at runtime. See [Figure 46-5](#).

Figure 46-5 Sample Business Case of Cross-Domain Deployment



An instance of Service Bus is deployed in each of two domains. The Service Bus proxy service (P1) is configured in domain (D1). The Service Bus proxy service (P2) in domain (D2) requires access to proxy service (P1). As the domains cannot

communicate directly with each other, P2 in D2 cannot use P1 in D1. The Service Bus import and export feature does not support runtime discovery of services in different domains, but publishing the service to a UDDI registry allows the discovery and use of a service in any domain. Once P1 is made available in the UDDI registry it can be invoked at runtime (for example, get a stock quote) and imported as a business service in another Service Bus proxy service.

When importing and exporting from different domains you should have network connectivity. A proxy service might reference schemas located in the repository of a different domain, in which case you need HTTP access to the domain to import it using the URL. In the absence of connectivity an error message is returned.

Mapping Service Bus Proxy Services to UDDI Entities

Service Bus proxy service attributes must be mapped to the data model supported by the UDDI registry to allow a proxy service to be published as a UDDI business entity. [Table 46-2](#) shows the service types, message types, and transports relevant to the UDDI registry mapping for a proxy service.

Table 46-2 Proxy Service Attributes and Service Types

Service Type	Message Content Type	Transports
WSDL	SOAP or XML (with attachment)	HTTP, JMS, Local, SB, WS
Transport Typed	SOAP or XML	JEJB
Any SOAP	Untyped SOAP (with attachment)	HTTP, JMS, Local, SB
Any XML	Untyped XML (with attachment)	Email, File, FTP, HTTP, JMS, Local, MQ, SB, SFTP, Tuxedo
Messaging	Binary, Text, MFL, XML (schema)	Email, File, FTP, HTTP, JMS, Local, MQ, SFTP, Tuxedo

Note:

Optional parts are listed in parentheses. Messaging services can have different content for requests and responses, or can have no response at all (one-way messages). Email, File, SFTP, and FTP should be one-way.

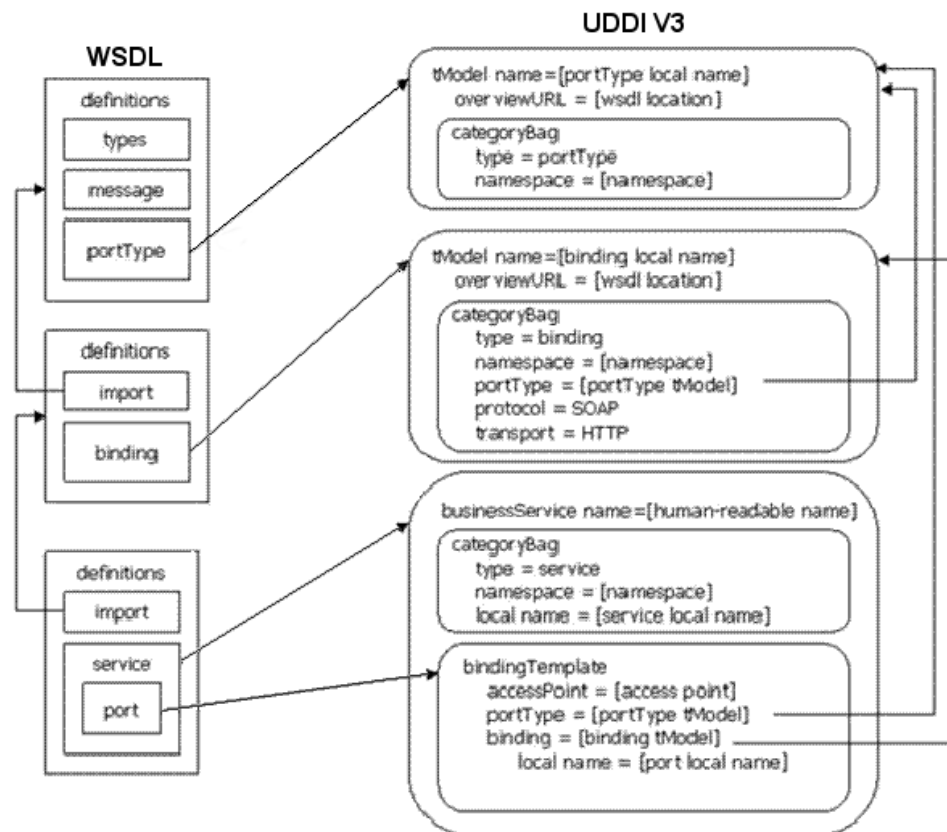
Proxy services have attributes in common and also attributes that are specifically defined by the transport protocols used by the service and the type of service. Each proxy service can deliver messages of a certain type.

The primary relevant entities in UDDI include the following:

- `businessService`: Represents the service as a whole and contains high-level general information about the service.
- `bindingTemplate`: Contains information for accessing the service.
- `tModels`: Supplies the individual attributes for categorizing and defining the service.

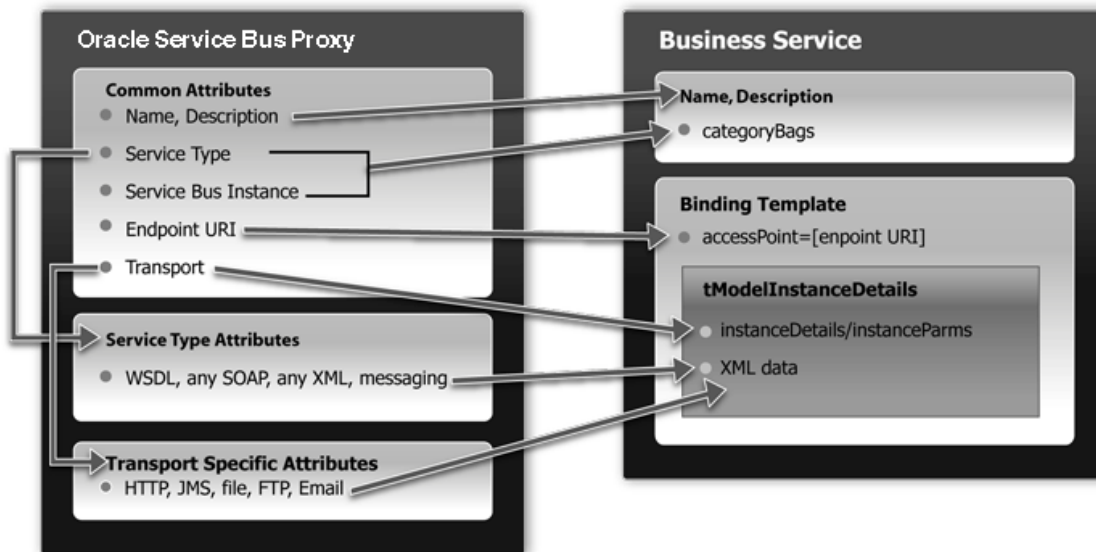
[Figure 46-6](#) shows how WSDL-based services are mapped to UDDI business entities.

Figure 46-6 WSDL Service to UDDI Mapping



The technical note on *Using WSDL in a UDDI registry, version 2.0.2*, at <http://www.oasis-open.org/committees/uddi-spec/doc/tns.htm>, is used as the basis for publishing WSDL-based proxy services to the UDDI registry. This document is also used as a reference point for publishing non-WSDL based services. The document and the base UDDI specification describe the canonical technical models (tModels) used to describe UDDI entities. To publish Service Bus proxy services as entities in the UDDI registry, you must provide additional canonical tModels to support some of the constructs specific to Service Bus. Not all attributes of a proxy service are useful when searching for a service, for example, service type and transport details. These attributes do not categorize the service. tModels are configuration details of the service once it has been discovered. These configuration details are mapped to the business service binding template `tmodelinstanceDetails` section. Other attributes specifically identify a service and can be used as the search criteria for the service. These attributes are mapped using keyed references to tModels with values in the `categoryBag` of the binding template.

An example of how Service Bus maps to UDDI is shown in Figure 46-7.

Figure 46-7 Service Bus to UDDI Mapping

UDDI Mapping Details for a Service Bus Proxy Service

Service Bus high-level proxy service information maps to the business service as follows:

- The Name and Description map to `businessService` elements.
- There is a special `keyedReferenceGroup` for Service Bus properties. An example of a key is `uddi:bea.com:attributes:oracleservicebus`.
- Service Bus type (WSDL, SOAP, XML, and Mixed) and instance are mapped to `keyedReferences` in the service category. An example of a key is `uddi:bea.com:serVICetype`.
- A Service Bus instance maps to a `keyedReference` in the Service Bus `keyedReferenceGroup` (Name = "OracleServiceBus", Values = URL of the Service Bus instance).

This instance serves two purposes:

- To indicate that this service is in fact hosted by a Service Bus server.
- To contain the URL of the Service Bus instance.

The following example shows a mapping of high-level proxy service information to a business service.

Example - Sample Proxy Service to Business Service Mapping

```
<keyedReferenceGroup tModelKey="uddi:bea.com:servicebus:properties">
  <keyedReference tModelKey="uddi:bea.com:servicebus:serVICetype"
    keyName="Service Type"
    keyValue="SOAP" />
  <keyedReference tModelKey="uddi:bea.com:servicebus:instance"
    keyName="Service Bus Instance"
    keyValue="http://F0002.amer.bea.com:7001" />
</keyedReferenceGroup>
```

Note:

The key for the `businessService` created when a proxy service is published is a publisher assigned key name. It is derived from the Service Bus domain name, the path of the proxy service, and the proxy service name. It takes the following form:

```
uddi:bea.com:servicebus:<domainname>:<path>:<servicename>
```

For example, AnonESBAn, a Service Bus domain, contains a project named Proxy, which contains a folder named Accounting, which in turn contains a proxy service called PayoutProxy. When PayoutProxy is published to UDDI, its `businessService` is created with the following key:

```
uddi:bea.com:servicebus:AnonESB:Proxies:Accounting:PayoutProxy
```

Service Bus detailed proxy service information maps into the binding template as follows:

- The Endpoint URI maps to the access point.
- The Marker tModel for each transport maps to `tModelInstanceDetails`.
 - Transport tModels for HTTP, JMS, File, FTP, Email. New tModels are packaged with Service Bus to support JMS and File transports.
 - Detailed Service Bus configuration information maps to `instanceParms`.
- The Marker tModel for each service type maps to the `tModelInstanceDetails`. This includes the following:
 - Protocol tModels for WSDL, any SOAP, any XML, and Messaging. New tModels are packaged with Service Bus to support anySOAP, anyXML, and Messaging.
 - WSDL maps using WSDL to UDDI technology note.
 - Messaging has detailed configuration information that maps to `InstanceParms`.

The following example shows a detailed information mapping to the binding template.

Example - Sample Detailed Mapping to the Binding Template

```
<bindingTemplate bindingKey="uddi:" serviceKey="uddi:">
  <accessPoint useType="endPoint">file:///c:/temp/in3</accessPoint>
  <tModelInstanceDetails>
    <tModelInstanceInfo tModelKey="uddi:uddi.org:transport:file">
      <InstanceDetails>
        <InstanceParms><ALSBInstanceParms xmlns="http://www.bea.com/wli/sb/uddi">
          <property name="fileMask" value="*.*"/>
          <property name="sortByArrival" value="false"/> </ALSBInstanceParms>
        </InstanceParms>
      </InstanceDetails>
    </tModelInstanceInfo>
    <tModelInstanceInfo tModelKey="uddi:bea.com:servicebus:protocol:
      messagingservice">
      <InstanceDetails>
```

```

<InstanceParms><ALSBIInstanceParms xmlns="http://www.bea.com/wli/sb/uddi">
  <property name="requestType" value="XML"/>
  <property name="RequestSchema" value="http://domain.com:7001
    /sbresource?SCHEMA%2FDJS%2FOAGProcessPO"/>
  <property name="RequestSchemaElement"
    value="PROCESS_PO"/>
  <property name="responseType" value="None"/></ALSBIInstanceParms>
</InstanceParms>
</InstanceDetails>
</tModelInstanceInfo>
</tModelInstanceDetails>
</bindingTemplate>

```

Transport Attributes

Each of the transport types in the `uddi:uddi.org:transport:*` group has a different set of detailed metadata. See [Table 46-2](#). This metadata provides the configuration details of the transport for the proxy service. It is neither useful for characterizing the service nor useful in querying the service. However, after the service has been discovered, this data is needed to access the service. The metadata is represented by an XML string and is located in the `instanceParms` field in `tModelInstanceInfo`.

If you are mapping a proxy service that uses the HTTP transport, and as part of the HTTP configuration you need to describe some configuration details, including the required client authorization and the request and response character encoding. The following example provides an example of what must appear in the `bindingTemplate tModelInstanceDetails`.

Example - Example of `tModelInstanceDetails`

```

<tModelInstanceDetails>
  <tModelInstanceInfo tModelKey="uddi:uddi.org:transport:http">
    <instanceDetails>
      <instanceParms>
        <ALSBIInstanceParms xmlns="http://www.bea.com/wli/sb/uddi">
          <property name="client-auth" value="basic"/>
          <property name="request-encoding" value="iso-8859-1"/>
          <property name="response-encoding" value="utf-8"/>
          <property name="Scheme" value="http"/>
        </ALSBIInstanceParms>
      </instanceParms>
    </instanceDetails>
  </tModelInstanceInfo>
</tModelInstanceDetails>

```

Note:

For each transport, the service endpoint is always stored in the `bindingTemplate accessPoint` field.

The `client-auth` property is present in the `instanceParms` of the HTTP or HTTPS transport attributes whenever authentication is configured. The possible values for `client-auth` are `basic`, `client-cert`, and `custom-token`. Whenever the value is `custom-token`, two additional properties are present: `token-header` and `token-type`.

Because Service Bus business service definitions do not support custom token authentication in this release, if you import a service from UDDI that has a value of `custom-token` for `client-auth`, the service is imported as if it does not have any authentication configuration.

Table 46-3 is organized by transport type and lists the `tModelKey` and `instanceParms` used by each of the transports.

Table 46-3 Transport Attributes

Transport	tModelKey	InstanceParms
Email ¹	uddi:uddi.org:transport:smtp	<ul style="list-style-type: none"> Attachment Supported [Boolean] Request Encoding
File	uddi:uddi.org:transport:file	<ul style="list-style-type: none"> File Mask Sort by Arrival [Boolean] Request Encoding
FTP	uddi:uddi.org:transport:ftp	<ul style="list-style-type: none"> File Mask Sort by Arrival [Boolean] Transfer Mode [Text, Binary] Request Encoding
HTTP	uddi:uddi.org:transport:http	<ul style="list-style-type: none"> Client Authentication [None, Basic, Client Certificate (HTTP only), and Custom Token] Request Encoding Response Encoding
JEJB	uddi:uddi.org:transport:jejb	<ul style="list-style-type: none"> URI EJB Spec Version Client JAR Home Interface (not published for EJB 3.0) Remote Interface (business interface for EJB 3.0) Method Names
JMS	uddi:uddi.org:transport:jms	<ul style="list-style-type: none"> Destination Type [Queue, Topic] Response Required, Response URI Response Message Type [Bytes, Text] Request Encoding Response Encoding
Local	uddi:uddi.org:transport:local	None
MQ	uddi:bea.org:transport:mq	<ul style="list-style-type: none"> Response Required Response URI Response Correlation Pattern
SB	uddi:bea.org:transport:sb The URI scheme is <code>sb</code> when <code>use ssl</code> is false; <code>sbs</code> when <code>use ssl</code> is true.	None

Table 46-3 (Cont.) Transport Attributes

Transport	tModelKey	InstanceParms
SFTP	uddi:bea.org:transport:sftp	<ul style="list-style-type: none"> • File Mask • Sort by Arrival [Boolean] • Request Encoding • Authentication Mode
Tuxedo	uddi:bea.org:transport:tuxedo	<ul style="list-style-type: none"> • Response Required • Access Point ID • Buffer Type • Buffer Subtype • Classes Jar • Field Table Classes • View Classes
WS	uddi:uddi.org:transport:http WS uses the HTTP tModelKey	None

- ¹ The accessPoint in the Binding Template for an Email transport uses the standard mailto URL format: `mailto:name@some_server.com`. This is different from the one configured for the proxy service in Service Bus, which is a URL oriented toward reading email. It is not possible to derive this mailto URL from the proxy service definition as the server name is not known. For example, if the proxy service is defined to read from a POP3 server, it might be defined with a URL such as `mailfrom:pop3.bea.com`. When publishing such a proxy service, a dummy server is added. In the above example, the published URL will take the form `mailto:some_name@some_server.com`.

Service Type Attributes

Table 46-4 provides a high-level description of each of the service types.

Table 46-4 Service Type Attributes

Service	Description
WSDL	WSDL-based proxies map to UDDI based on the <i>Using WSDL in a UDDI Registry, version 2.0.2</i> technical note at URL: http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v202-20040631.htm .
Any SOAP	A simple marker protocol in the tModel in the <code>bindingTemplate</code> <code>tModelInstanceDetails</code> , as well as in the <code>categoryBag</code> , defines the Any Soap attributes.
Any XML	A simple marker protocol tModel within the <code>bindingTemplate</code> <code>tModelInstanceDetails</code> , as well as in the <code>categoryBag</code> defines the Any XML attributes.

Table 46-4 (Cont.) Service Type Attributes

Service	Description
Messaging Services	<p>A simple marker protocol tModel in the bindingTemplate tModelInstanceDetails, defines the messaging services attributes. Unlike the other service types, messaging services have additional configuration information associated with them, which provides detail about the request and response messages. The configuration details are represented as XML data in the InstanceParms data for the following tModel reference in the tModelInstanceInfo:</p> <ul style="list-style-type: none"> • Input message format (XML, Text, Binary, MFL) • URL of input message schema in Service Bus (optional, if input message is XML) • URL of input message MFL in Service Bus (if input message is MFL) • Output message format (none, XML, Text, Binary, MFL) • URL of output message schema in Service Bus (optional, if output message is XML) • URL of output message MFL in Service Bus (if output message is MFL)

Canonical tModels Supporting Service Bus Services

The Service Bus to UDDI mapping provides a number of canonical tModels that are used to represent Service Bus metadata and relationships. These tModels must be registered in the UDDI registry to support this mapping. You can create the tModels in the UDDI registry under the administrator ID.

Table 46-5 through Table 46-8 provide a summary of the tModels.

Table 46-5 CategorizationGroup tModel Types

Name	Description
bea-com:servicebus:properties	Describes very specific attributes of a Service Bus service. In the data model it is used in the business service categoryBag.

Table 46-6 Categorization tModel Types

Name	Value	Description
bea-com:servicebus:serviceType	WSDL, SOAP, XML, Messaging Service	Describes the service type of the Service Bus service.
bea-com:servicebus:instance	URL of Service Bus Administration Console	Describes the service instance in Service Bus responsible for publishing the service to UDDI.

Table 46-7 Transport tModel Types

Name	Description
uddi-org:jms	Describes the type of transport used by the service. A reference to it is found in the accessPoint attribute of the business service binding template.
uddi-org:file	Describes the type of transport used to invoke the service. A reference to it is found in the accessPoint attribute of the business service binding template.

Table 46-8 Protocol tModel Types

Name	Description
bea-com:servicebus:anySoap	Describes the type of protocol used to access the service. It designates services that have a SOAP message but not defined by a WSDL file or schema. The message body content is determined dynamically by the application.
bea-com:servicebus:anyXML	Describes the type of protocol used to access the service. It designates services having an XML message but not defined by a WSDL file or schema. The message body content is determined dynamically by the application.
bea-com:servicebus:messagingService	Describes the type of protocol used to access the service. It designates services where the request message can be any XML (with or without schema), text, binary, or MFL and whose response message can be any of the above or none. The message body content is determined dynamically by the application.

Mapping Example

The following example is a sample of the mapping for a Messaging Service, configured with JMS transport, the request being XML with a schema and the response being a text message.

Example - Sample Messaging Service Mapping

```
<businessService
  serviceKey="uddi:bea.com:servicebus:Domain:Project:JMSMessaging"
  businessKey="uddi:9cb77770-57fe-11da-9fac-6cc880409fac"
  xmlns="urn:uddi-org:api_v3">
  <name>JMSMessagingProxy</name>
  <bindingTemplates>
    <bindingTemplate
      bindingKey="uddi:4c401620-5ac0-11da-9faf-6cc880409fac"
      serviceKey="uddi:bea.com:servicebus:
        Domain:Project:JMSMessaging">
      <accessPoint useType="endPoint">
        jms://server.com:7001/weblogic.jms.XAConnectionFactory/
          ReqQueue
      </accessPoint>
    </bindingTemplate>
  </bindingTemplates>
  <tModelInstanceDetails>
    <tModelInstanceInfo tModelKey="uddi:uddi.org:transport:jms">
```

```

<instanceDetails>
  <instanceParms>
    <ALSBInstanceParms
      xmlns="http://www.bea.com/wli/sb/uddi">
        <property name="is-queue" value="true"/>
        <property name="request-encoding"
          value="iso-8859-1"/>
        <property name="response-encoding"
          value="utf-8"/>
        <property name="response-required"
          value="true"/>
        <property name="response-URI"
          value="jms://server.com:7001/
            .jms.XAConnectionFactory/
            RespQueue"/>
        <property name="response-message-type"
          value="Text"/>
        <property name="Scheme" value="jms"/>
      </ALSBInstanceParms>
    </instanceParms>
  </instanceDetails>
</tModelInstanceInfo>
<tModelInstanceInfo
  tModelKey="uddi:bea.com:servicebus:
    protocol:messaging-service">
  <instanceDetails>
    <instanceParms>
      <ALSBInstanceParms xmlns=
        "http://www.bea.com/wli/sb/uddi">
        <property name="requestType" value="XML"/>
        <property name="RequestSchema"
          value="http://server.com:7001/
            sbresource?SCHEMA%2FDJS%2FOAGProcessPO"/>
        <property name="RequestSchemaElement"
          value="PROCESS_PO_007"/>
        <property name="responseType" value="Text"/>
      </ALSBInstanceParms>
    </instanceParms>
  </instanceDetails>
</tModelInstanceInfo>
</tModelInstanceDetails>
</bindingTemplate>
</bindingTemplates>
<categoryBag>
<keyedReferenceGroup tModelKey="uddi:bea.com:servicebus:properties">
  <keyedReference tModelKey="uddi:bea.com:servicebus:servicetype"
    keyName="Service Type"
    keyValue="Mixed" />
  <keyedReference tModelKey="uddi:bea.com:servicebus:instance"
    keyName="Service Bus Instance"
    keyValue="http://cyberfish.bea.com:7001" />
</keyedReferenceGroup>
</categoryBag>
</businessService>

```


Part VIII

Security

The chapters in this part describe how to secure Service Bus and the messages it handles.

The chapters include:

- [Understanding Oracle Service Bus Security](#)
- [Oracle Service Bus Security FAQ](#)
- [Securing Business and Proxy Services](#)
- [Configuring Message-Level Security for Web Services](#)
- [Configuring Transport-Level Security](#)
- [Securing Oracle Service Bus with Oracle Web Services Manager](#)
- [Securing Oracle Service Bus Proxy and Business Services with WS-Policy](#)
- [Using SAML with Oracle Service Bus](#)
- [Configuring Custom Authentication](#)
- [Defining Message-Level Security with .Net 2.0](#)

Related Information

Service Bus uses the Oracle WebLogic security framework as the foundation for higher level security services, including authentication, identity assertion, authorization, role mapping, auditing, and credential mapping. In addition to this document, the following documents provide information about Oracle WebLogic security:

- *[Understanding Security for Oracle WebLogic Server](#)*
- *[Administering Web Services](#)*
- *[Securing a Production Environment for Oracle WebLogic Server](#)*
- *[Administering Security for Oracle WebLogic Server](#)*
- *[Securing Resources Using Roles and Policies for Oracle WebLogic Server](#)*

Understanding Oracle Service Bus Security

This chapter provides an overview of the Service Bus security model and its features, including inbound and outbound security.

Service Bus supports open industry standards for ensuring the integrity and privacy of communications and to ensure that only authorized users can access resources in a Service Bus domain. It uses the underlying WebLogic security framework as building blocks for its security services.

Service Bus uses Oracle Platform Security Services (OPSS) and Oracle Web Services Manager (OWSM) as the building blocks for higher-level security services including authentication, identity assertion, authorization, role mapping, auditing, and credential mapping. In order to configure Service Bus access security, you must first configure Oracle WebLogic Server security. Service Bus uses OWSM to provide a policy framework to manage and secure web services consistently across your organization.

Note:

Oracle Web Services Manager (OWSM) is the Web Services security mechanism used by Oracle Service Bus. All newly created resources, such as business services and proxy services, use OWSM policies for security. WLS 9 policies are deprecated, and cannot be used to configure security for a new Service Bus resource.

However, you can import resources already configured with WLS 9 policies into your Service Bus project. You cannot edit or modify these WLS 9 policies, but you can replace them with OWSM policies.

This chapter includes the following sections:

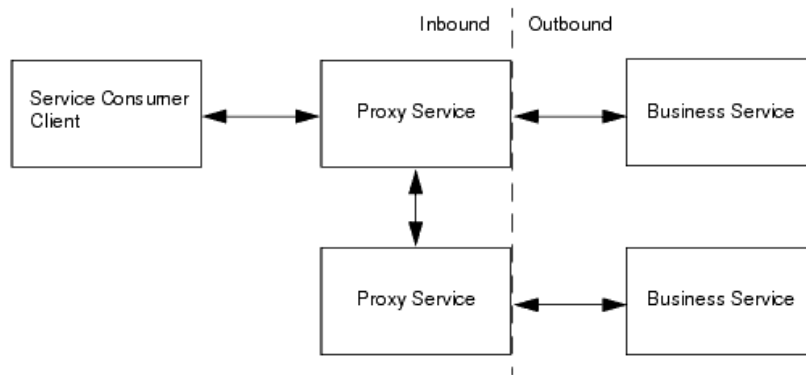
- [Inbound Security](#)
- [Outbound Security](#)
- [Options for Identity Propagation](#)
- [Administrative Security](#)
- [Configuring the Oracle WebLogic Security Framework: Main Steps](#)
- [Context Properties Are Passed to Security Providers](#)
- [Using Security Providers](#)

Inbound Security

Inbound security ensures that Service Bus proxy services handle only the requests that come from authorized clients. (By default, any anonymous or authenticated user can connect to a proxy service.) It can also ensure that no unauthorized user has viewed or modified the data as it was sent from the client.

Proxy services can have two types of clients: service consumers and other proxy services. [Figure 47-1](#) illustrates that communication between proxy services and their clients is secured by inbound security, while communication between proxy services and business services is secured by outbound security.

Figure 47-1 Inbound and Outbound Security



You set up inbound security when you create proxy services and you can modify it as your needs change. For outward-facing proxy services (which receive requests from service consumers), consider setting up strict security requirements such as two-way SSL over HTTPS.

If a proxy service uses public key infrastructure (PKI) technology for digital signatures, encryption, or SSL authentication, create a service key provider to provide private keys paired with certificates. For more information, see [Working with Service Key Providers](#).

For each proxy service, you can configure the following inbound security checks:

- **Transport-level security** applies security checks as part of establishing a connection between a client and a proxy service. The security requirements that you can impose through transport-level security depend on the protocol that you configure the proxy service to use.

For example, for proxy services that communicate over the HTTP protocol, you can require that all clients authenticate against a database of users that you create in Fusion Middleware Control. You then create an access control policy that specifies conditions under which authenticated users are authorized to access the proxy service.

Service Bus also supports client-specified custom authentication tokens for inbound transport-level requests.

For information about configuring transport-level security for each supported protocol, see [Configuring Transport-Level Security](#).

- **Custom Authentication for message-level security.** Service Bus supports client-specified custom authentication credentials for inbound transport-level and

message-level requests. The custom authentication credentials can be in the form of a custom token, or a user name and password.

For information on configuring custom authentication transport- and message-level security, see [Configuring Custom Authentication](#).

- **Message-level security** (for proxy services that are web services) is part of the WS-Security specification. It applies security checks before processing a SOAP message or specific parts of a SOAP message.

Part of the configuration for message-level security can be embedded in the WSDL document and WS-Policy document that are associated with the web service. These documents specify whether SOAP messages must be digitally signed and encrypted and which web service operations can be invoked only by authorized users.

Message protection involves encrypting the message for message confidentiality and signing the message for message integrity. OWSM predefined policies and any policy you create using one of the message-protection assertion templates provide the options for message confidentiality, message integrity, or both.

If a proxy service or business service uses a WS-Policy statement to secure access to one or more of its operations, and if you have configured the service as an active intermediary (as opposed to a pass-through service), you use the Oracle Service Bus Console to create a message-level access control policy. The policy specifies conditions under which users, groups, or security roles are authorized to invoke the protected operations.

For more information about configuring message-level security, see [Configuring Message-Level Security for Web Services](#).

Outbound Security

Outbound security secures communication between a proxy service and a business service. Most of the tasks that you complete for outbound security are for configuring proxy services to comply with the transport-level or message-level security requirements that business services specify.

For example, if a business service requires user name and password tokens, you create a service account, which either directly contains the user name and password, passes along the user name and password that was contained in the inbound request, or provides a user name and password that depend on the user name that was contained in the inbound request. For more information, see [Working with Service Accounts](#).

If a business service requires the use of PKI technology for digital signatures, or SSL authentication, you create a service key provider, which provides private keys paired with certificates. For more information, see [Working with Service Key Providers](#).

Options for Identity Propagation

A key group of decisions that you must make when designing security for Service Bus is how to handle (propagate) the identities that clients provide. You can configure Service Bus to do any of the following:

- Authenticate the credentials that clients provide
- Perform authorization checks
- Pass client credentials to business services unchanged

- Map client credentials to a different set of credentials that a business service can authenticate and authorize
- Bridge between security technologies

[Table 47-1](#) describes the decisions that affect how Service Bus propagates client identities to business services.

Table 47-1 Options for Identity Propagation

Decision	Description
Which type of credentials do you require clients to provide?	<p>For transport-level security, Service Bus adapts to your existing security requirements. Clients of Service Bus can supply user name and password tokens, SSL certificates, or any other type of custom authentication token that is supported by an Identity Assertion provider that you configure.</p> <p>For message-level security, Service Bus supports the Username Token, X.509 Token, any other type of custom authentication token that is supported by an Authentication or Identity Assertion provider that you configure, and SAML Token profiles (see Using Security Providers).</p> <p>If you are establishing security requirements for a new business service that uses Web Services Security, Oracle recommends that you require clients to provide SAML tokens. SAML is the emerging standard for propagating user identities within web services. See Using SAML with Oracle Service Bus.</p>
Do you require Service Bus to authenticate clients or to simply pass the client-supplied credentials to business services for authentication?	<p>When you require clients to authenticate with Service Bus, you add an additional layer of security. In general, the more security layers you add, the more secure you make a domain.</p> <p>To enable Service Bus to authenticate users, you must create user accounts in Fusion Middleware Control. If your set of users is very large, you must consider whether maintaining a large database of user accounts is worth the effort.</p>
If Service Bus authenticates clients that provide X.509 tokens or SAML tokens, which Service Bus user maps to the tokens?	<p>Oracle recommends that you require clients to authenticate with Service Bus and that you modify the default access-control policies to allow (authorize) only specific, authenticated users access to your proxy services.</p> <p>To authenticate and authorize clients who supply X.509 certificates, SAML tokens, or other types of credentials other than user names and passwords, you must configure an Identity Assertion provider that maps the client's credential to a Service Bus user. Service Bus will use this user name to establish a security context for the client.</p>

Table 47-1 (Cont.) Options for Identity Propagation

Decision	Description
<p>If Service Bus authenticates clients that provide custom authentication tokens, which Service Bus user maps to the tokens?</p>	<p>Oracle recommends that you require clients to authenticate with Service Bus and that you modify the default access-control policies to authorize only specific, authenticated users to have access to your proxy services.</p> <p>To authenticate and authorize clients who supply custom authentication tokens other than user names and passwords, you must configure an Identity Assertion provider that maps the client's credential to a Service Bus user. Service Bus will use this user name to establish a security context for the client.</p>
<p>If Service Bus authenticates clients that provide user name and password tokens, decide whether you want to:</p> <ul style="list-style-type: none"> • Pass the client's user name and password to the business service • Map the client's user name to a new user name and password and pass the new credentials to the business service 	<p>If a custom user name/password token is used, as described in Understanding Custom Authentication Tokens, then the user name and password in the custom token can be used for outbound HTTP basic or outbound WS-Security Username Token authentication if a pass-through service account is used.</p> <p>If you pass the client-supplied user name and password to the business service, then clients are responsible for maintaining the credentials that the business service requires. If the business service changes its security requirements, then you must notify each client to make corresponding changes.</p> <p>If you expect a business service to change its requirements frequently, then consider mapping the credentials that clients supply to the credentials that the business service requires. The more clients for a business service, the more work will be required to maintain this credential mapping.</p>

[Table 47-2](#) describes all combinations of the requirements that you can impose for inbound and outbound transport-level security.

Table 47-2 Combinations of Transport-Level Security Requirements

This Inbound Requirement...	Can Be Used With This Outbound Requirement...	How to Configure
Client supplies user name and password in the HTTP header and Service Bus authenticates the client.	Pass the client's credentials in an HTTP header.	<ol style="list-style-type: none"> <li data-bbox="976 405 1373 611">1. Configure inbound HTTP security. See Configuring Inbound HTTP Security: Main Steps. Be sure to add the client's user name to the Service Bus Security Configuration module. <li data-bbox="976 636 1373 747">2. Configure outbound HTTP security. See Configuring Outbound HTTP Security: Main Steps. Be sure to create a pass-through service account and attach the account to the business service.
Same as previous requirement.	Map the client's credentials to a different Service Bus user and pass the new credentials in an HTTP header.	<ol style="list-style-type: none"> <li data-bbox="976 909 1373 1115">1. Configure inbound HTTP security. See Configuring Inbound HTTP Security: Main Steps. Be sure to add the client's user name to the Service Bus Security Configuration module. <li data-bbox="976 1140 1373 1251">2. Configure outbound HTTP security. See Configuring Outbound HTTP Security: Main Steps. Be sure to create a user-mapping service account and attach the account to the business service.

Table 47-2 (Cont.) Combinations of Transport-Level Security Requirements

This Inbound Requirement...	Can Be Used With This Outbound Requirement...	How to Configure
Client supplies user name and password in the HTTP header and Service Bus does not authenticate the client.	Pass the client's credentials in an HTTP header.	<ol style="list-style-type: none"> 1. Configure inbound HTTP security. See Configuring Inbound HTTP Security: Main Steps. Be sure to configure the proxy service for HTTP, no authentication or HTTPS, one-way SSL, no authentication. 2. Configure outbound HTTP security. See Configuring Outbound HTTP Security: Main Steps. Be sure to configure the business service for HTTP basic authentication or HTTPS, one-way SSL, basic authentication. Also create a pass-through service account and attach the account to the business service.
Client supplies custom authentication token in the HTTP header. Service Bus authenticates the client.	Map the client's credentials to a different Service Bus user and pass the new credentials in an HTTP header.	<ol style="list-style-type: none"> 1. Configure inbound HTTP security. See Configuring Inbound HTTP Security: Main Steps. Be sure to add the client's user name to the Service Bus Security Configuration module. 2. Configure outbound HTTP security. See Configuring Outbound HTTP Security: Main Steps. Be sure to create a user-mapping service account and attach the account to the business service.
Any form of local authentication (HTTP or HTTPS basic, HTTPS client certificate with credential mapping)	Pass the client's credentials to an EJB over RMI. The EJB container authenticates the user.	Create a pass-through service account and attach the account to the business service. See Working with Service Accounts .

[Table 47-3](#) describes all combinations of the requirements that you can impose for inbound and outbound message-level security. In some cases, the inbound requirement for *transport-level* security affects the requirements that you can impose for outbound message-level security.

Table 47-3 Combinations of Message-Level Security Requirements

This Inbound Requirement...	Can Be Used With This Outbound Requirement...	How to Configure
Client supplies user name and password, or custom authentication token, in the HTTP header and Service Bus authenticates the client.	Pass the client's credentials in a SOAP header.	<ol style="list-style-type: none"> 1. Configure inbound HTTP security. See Configuring Inbound HTTP Security: Main Steps. Be sure to create a user account with the client's user name. 2. Create a pass-through service account and attach the account to the business service. See Working with Service Accounts.
Same as previous requirement.	Map the client's credentials to a different Service Bus user and pass the new credentials in a SOAP header.	<ol style="list-style-type: none"> 1. Configure inbound HTTP security. See Configuring Inbound HTTP Security: Main Steps. Be sure to create a user account with the client's user name. 2. Create a user-mapping service account and attach the account to the business service. See Working with Service Accounts.
Same as previous requirement.	Map the client credentials to a SAML token. Service Bus asserts the user identity.	<ol style="list-style-type: none"> 1. Configure inbound HTTP security. See Configuring Inbound HTTP Security: Main Steps. Be sure to create a user account with the client's user name. 2. Configure a SAML credential mapping provider. See Mapping Identity to a SAML Token.
Client supplies custom user name and password, or custom authentication token, in the message header or body and Service Bus authenticates the client.	Pass the client's credentials in a SOAP header.	<ol style="list-style-type: none"> 1. Configure an Authentication or Identity Assertion provider to handle the custom token or user name and password. Be sure to create a user account with the client's user name. 2. Create a pass-through service account and attach the account to the business service. See Working with Service Accounts.

Table 47-3 (Cont.) Combinations of Message-Level Security Requirements

This Inbound Requirement...	Can Be Used With This Outbound Requirement...	How to Configure
Same as previous requirement.	Map the client's credentials to a different Service Bus user and pass the new credentials in a SOAP header.	<ol style="list-style-type: none"> 1. Configure an Authentication or Identity Assertion provider to handle the custom token or user name and password. Be sure to create a user account with the client's user name. 2. Create a user-mapping service account and attach the account to the business service. See Working with Service Accounts.
Same as previous requirement.	Map the client credentials to a SAML token. Service Bus asserts the user identity.	<ol style="list-style-type: none"> 1. Configure an Authentication or Identity Assertion provider to handle the custom token or user name and password. Be sure to create a user account with the client's user name. 2. Configure a SAML credential mapping provider. See Mapping Identity to a SAML Token.
Client supplies user name and password in the HTTP header and Service Bus does not authenticate the client.	Pass the client's credentials in a SOAP header.	<ol style="list-style-type: none"> 1. Configure inbound HTTP security. See Configuring Inbound HTTP Security: Main Steps. Be sure to configure the proxy service for HTTP, no authentication or HTTPS, one-way SSL, no authentication. 2. Configure outbound HTTP security. See Configuring Outbound HTTP Security: Main Steps. Be sure to configure the business service for HTTP basic authentication or HTTPS, one-way SSL, basic authentication. Also create a pass-through service account and attach the account to the business service.

Table 47-3 (Cont.) Combinations of Message-Level Security Requirements

This Inbound Requirement...	Can Be Used With This Outbound Requirement...	How to Configure
Client supplies a certificate as part of HTTPS client certificate authentication (two-way SSL) and Service Bus authenticates the client.	Map the client credentials to a SAML token. Service Bus asserts the user identity.	<ol style="list-style-type: none"> 1. Configure inbound HTTP security. See Configuring Inbound HTTP Security: Main Steps. 2. Configure a SAML credential mapping provider. See Mapping Identity to a SAML Token.
An active intermediary proxy service enforces Web-Services Security with the User Name Token Profile.	Encode the credentials as a user name and password token in the SOAP message.	Create an active intermediary proxy service with a WS-Policy statement that requires passwords (not password digests). See Creating an Active Intermediary Proxy Service: Main Steps .
Same as previous requirement.	Encode the credentials as a SAML token in the SOAP message.	<ol style="list-style-type: none"> 1. Create an active intermediary proxy service with a WS-Policy statement that requires passwords. See Creating an Active Intermediary Proxy Service: Main Steps. 2. Configure a SAML credential mapping provider. See Mapping Identity to a SAML Token.
An active intermediary proxy service enforces Web-Services Security with the X.509 Token Profile.	Encode the credentials as a SAML token in the SOAP message.	<ol style="list-style-type: none"> 1. Create an active intermediary proxy service with a WS-Policy statement that requires digital signatures and optionally requires authentication with an X.509 token. See Creating an Active Intermediary Proxy Service: Main Steps. 2. Configure a SAML credential mapping provider. See Mapping Identity to a SAML Token.

Table 47-3 (Cont.) Combinations of Message-Level Security Requirements

This Inbound Requirement...	Can Be Used With This Outbound Requirement...	How to Configure
An active intermediary proxy service enforces Web-Services Security with the SAML Token Profile.	Generate a new SAML token in the outbound SOAP message.	<ol style="list-style-type: none"> 1. Create an active intermediary proxy service with a WS-Policy statement that requires a SAML token. See Authenticating SAML Tokens in Proxy Service Requests. 2. Configure a SAML credential mapping provider. See Mapping Identity to a SAML Token.
A pass-through proxy service, which can pass user names and passwords, X.509 tokens, or SAML tokens.	A business service that uses either the User Name Token Profile, the X.509 Token Profile, or the SAML Token Profile.	<ol style="list-style-type: none"> 1. Create a pass through proxy service. See Creating an Active Intermediary Proxy Service: Main Steps. 2. Create a business service that enforces one of the token profiles. See Configuring Business Service Message-Level Security: Main Steps or Configuring SAML Pass-Through Identity Propagation.

For inbound Tuxedo requests, you can configure any of the following security requirements:

- Encode the client's credentials in an outbound call to a Tuxedo service.
- Encode the client's credentials in an outbound SOAP message as either a user name token or a SAML token.
- Map the client's credentials to a different Service Bus user and pass the new credentials in an outbound HTTP header.
- Map the client's credentials to a different Service Bus user and pass the new credentials to an EJB over RMI. The EJB container authenticates the user.

For information about using Tuxedo with Service Bus, see [Using the Tuxedo Transport](#).

Using a Service Account with Business Service when Attaching OWSM Policies

The following out-of-the-box OWSM policies support using service accounts:

- `oracle/**_username_token**_client_policy`
- `oracle/wss11_saml_token_identity_switch_with_message_protection_client_policy`

- `oracle/**_saml*_**_client_policy` (The `subject.precedence` property has to be set to `false`)

The following out-of-the-box OWSM policy assertions support service accounts:

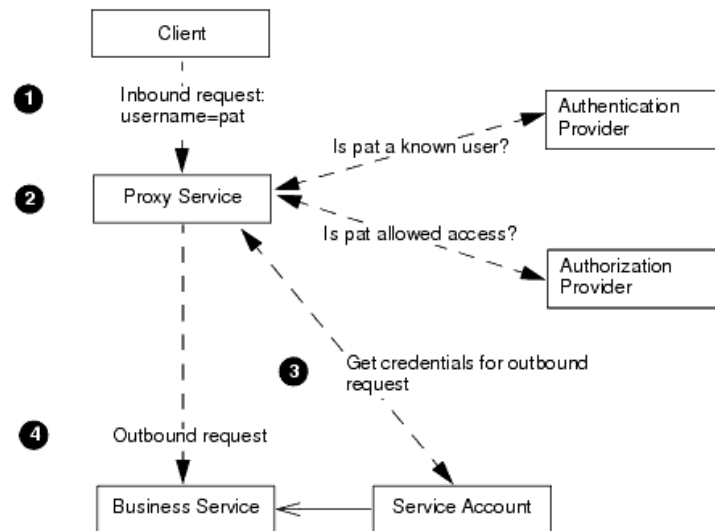
- `oracle/**_username_token_**_client_template`
- `oracle/**_saml*_**_client_template` (The `subject.precedence` property has to be set to `false`)

Example: Authentication with a User Name Token

Figure 47-2 illustrates how user identities flow through Service Bus when you configure Service Bus as follows:

- Require clients to provide user names and passwords in their requests
You can require web services clients to provide credentials at the transport level, the message level, or both. If you require clients to provide credentials at both levels, Service Bus uses the message-level credentials for identity propagation and credential mapping.
- Authenticate clients

Figure 47-2 How Service Accounts Are Used



The illustration begins with the inbound request and ends with the outbound request:

1. A client sends a request to a proxy service. The request contains the user name and password credentials.

Clients can send other types of tokens for authentication, such as an X.509 certificate or a custom authentication token. To authenticate and authorize clients who supply X.509 certificates, SAML tokens, or other types of credentials other than user names and passwords, you must configure an Identity Assertion provider that maps the client's credential to a Service Bus user. Service Bus will use this user name to establish a security context for the client.

2. The proxy service asks the domain's authentication provider if the user exists in the domain's authentication provider store.

If the user exists, the proxy service asks the domain's authorization provider to evaluate the access control policy that you have configured for the proxy service.

3. If the proxy service's access control policy allows the user access, the proxy service processes the message. As part of generating its outbound request to a business service, the proxy service asks the business service to supply the user name and password that the business service requires.

The business service asks its service account for the credentials. Depending on how the service account is configured, it does one of the following:

- Requires the proxy service to encode a specific (static) user name and password.
 - Requires the proxy service to pass along the user name and password that the client supplied.
 - Maps the user name that was returned from the authentication provider to some other (remote) user name, then requires the proxy service to encode the remote user name.
4. The proxy service sends its outbound request with the user name and password that was returned from the service account.

Administrative Security

To secure access to administrative functions, such as creating proxy services or business services, Service Bus provides security roles with pre-defined access privileges. See "Understanding Oracle Service Bus Application Security" in *Administering Oracle Service Bus* for more information on these roles.

A security role is an identity that can be dynamically conferred upon a user or group at runtime. You cannot change the access privileges for these administrative security roles, but you can change the conditions under which a user or group is in one of the roles.

The Service Bus roles have permission to modify only Service Bus resources; they do not have permission to modify WebLogic Server or other resources on WebLogic Server. When assigning administrative users to roles, assign at least one user to the WebLogic Server Admin role. The WebLogic Server security roles are described in "Roles" in *Administering Oracle Service Bus*.

Access Control Policies

Access control determines who has access to the resources in Service Bus. An access control policy specifies conditions under which users, groups, or roles can access a proxy service. For example, you can create a policy that always allows users in the GoldCustomer role to access a proxy service and that allows users in the SilverCustomer role to access the proxy service only after 12pm on weeknights.

An access control policy is an association between a WebLogic resource and one or more users, groups, or security roles. A security policy protects the WebLogic resource against unauthorized access. Access control policies are boolean expressions assigned to specific resources. When there is an attempt to access the resource, the expression is evaluated. The expression consists of one or more conditions joined by boolean operators, such as a role (operator) and access time (8 am to 5 pm). For more information about access control policies, see "Security Fundamentals" in *Understanding Security for Oracle WebLogic Server*.

Service Bus relies on WebLogic Server security realms to protect its resources. Each security realm consists of a set of configured security providers, users, groups, security roles, and (access control) security policies. To access any resources belonging to a realm, a user must be assigned a security role defined in that realm, as described in "Configuring Oracle Service Bus Administrative Security" in *Administering Oracle Service Bus*. When a user attempts to access an Service Bus resource, WebLogic Server authenticates and authorizes the user by checking the security role assigned to the user in the relevant security realm and relevant security policy.

Note:

Only a WebLogic Server administrator can define security policies or edit security roles in Fusion Middleware Control.

For all proxy services, you can create a transport-level policy, which applies a security check when a client attempts to establish a connection with the proxy service. Only requests from users who are listed in the transport-level policy are allowed to proceed.

For proxy services that are WS-Security active intermediaries, or that implement message-level custom authentication, you can also create a message-level policy. This type of policy applies a security check when a client attempts to invoke one of the secured operations. Only users who are listed in the message-level policy are allowed to invoke the operation.

You can view and configure security for users, groups, and roles in Fusion Middleware Control.

Configuring Proxy Service Access Control

You can configure transport-level access control for all proxy services. You can also configure access control at the message-level for any WS-Security active intermediary proxy service, or for any proxy service that implements message-level custom authentication. To configure access control, you must assign an access control policy to the proxy service, either at the transport-level or message-level (or both).

The default transport-level and message-level access control policy for all proxy services is to allow access to all requests. You must assign an access control policy to the proxy service to protect it.

You configure transport-level and message-level access control policies in the Oracle Service Bus Console, as described in [Configuring Service Bus Client Access Security](#).

Access Control Policy Management

Access control policies are persisted in authorization providers, and there is a reference to them in the Service Bus repository.

Access control policies are managed within a Service Bus design session and not outside the session, as was the case in releases prior to 3.0. Because the changes are made within a session, you can commit or discard the changes as with other resources.

Although ACLs can be managed from the Oracle Service Bus Console, you can change policies outside Service Bus. However, changing policies outside of Service Bus can make the reference in Service Bus out-of-date and invalid.

Therefore, for consistent management, either completely manage ACLs outside of Service Bus sessions (using the authorization provider MBeans or third-party authorization provider tools) or completely manage them from within Service Bus

sessions. Any combination of the two approaches can result in an inconsistent view of policies.

Service Bus manages access control policy only for proxy services. You must manage access control policy management for other server resources, such as JMS queues, JNDI entries, EJBs, applications, WebLogic Server instances, data sources, and so forth from the Oracle WebLogic Server Administration Console.

Note:

When you clone a service, ACLs are also cloned in the session. If the user commits the session, ACLs on the service will be committed to the authorization provider. Therefore, when you clone a service you need to decide if you want the clone to have the same ACLs as the original. If you do not want this, then make sure to edit the ACLs of the clone.

Deleting a Proxy Service

Deleting a proxy service deletes all of the ACLs referenced by the proxy from the repository controlled by Service Bus, as well as from the appropriate authorization provider.

Deleting the Access Control Policy Assigned to a Proxy Service

You can also delete the access control policies assigned to a service without deleting the service. To do this:

1. Create a session.
2. From the **View a Proxy Service > Security** tab, use the edit Transport Access Control option to delete the policies.
3. Commit the session.

Moving or Renaming a Proxy Service

Renaming a proxy service correctly moves all of the policies. You need only rename or move the service in a Service Bus session.

Renaming a Proxy Service Operation

When an operation is renamed, the existing operation is transparently deleted and a new operation is created.

However, when an operation name is changed by changing the WSDL file, Service Bus considers any policies for the old operation to be invalid, removes the reference from the Service Bus repository, and deletes the policies from the appropriate authorization provider.

In this case Service Bus does not know that the old operation is renamed to the new operation, and it does not add anything new for the new operation. That is, Service Bus considers that there are no policies for this new operation.

You need to add the appropriate policy to the new operation manually. You can do this in the same session as the rename of operation, after the rename is done.

Configuring the Oracle WebLogic Security Framework: Main Steps

Many of the initial configuration tasks for Service Bus security require you to work in the Oracle WebLogic Server Administration Console to configure the WebLogic security framework. After these initial tasks, you can complete most security tasks from the Oracle Service Bus Console.

To configure the WebLogic security framework for Service Bus:

1. If you plan to use SSL as part of transport-level security, do the following:
 - a. In the Oracle WebLogic Server Administration Console, configure identity and trust. See "Configuring Identity and Trust" and "Important Information Regarding Cross-Domain Security Support" in *Administering Security for Oracle WebLogic Server*.
 - b. In the Oracle WebLogic Server Administration Console, configure SSL. See "Configuring SSL" in *Administering Security for Oracle WebLogic Server*.

Oracle recommends the following for your SSL configuration:

- If you configure two-way SSL, you must choose between two modes: *Client Certificate Requested But Not Enforced* or *Client Certificates Requested and Enforced*. Oracle recommends that whenever possible you choose *Client Certificate Requested and Enforced*. For more information, see "Secure Sockets Layer (SSL)" in *Understanding Security for Oracle WebLogic Server*.
 - In a production environment, make sure that Host Name Verification is enabled. See "Using Host Name Verification" in "Configuring SSL" in *Administering Security for Oracle WebLogic Server*.
2. In the Oracle WebLogic Server Administration Console, configure authentication providers, which your proxy services use for inbound security.

[Table 47-4](#) describes the authentication providers that are commonly configured for Service Bus. For a description of all authentication providers that you can configure, see "Security Providers" in *Administering Security for Oracle WebLogic Server*.

Table 47-4 Authentication Providers

If You Require Clients to Provide...	Configure...
Simple user names and passwords	The WebLogic Authentication provider and use Fusion Middleware Control to enter the user names and passwords of the clients that you want to allow access. Note: As described in Configuring Authentication Providers , Oracle recommends that you use the default WebLogic Authentication provider for all WebLogic Server and Service Bus administrative accounts. See "Creating Oracle Service Bus Users" in <i>Administering Oracle Service Bus</i> .

Table 47-4 (Cont.) Authentication Providers

If You Require Clients to Provide...	Configure...
X.509 tokens for inbound HTTPS and two-way SSL authentication	All of the following: <ul style="list-style-type: none"> • The WebLogic Identity Assertion provider, which can validate X.509 tokens but does not by default. Make sure that you enable this provider to support X.509 tokens. In addition, enable this provider to use a user name mapper. See "Identity Assertion and Tokens" in <i>Understanding Security for Oracle WebLogic Server</i>. • WebLogic CertPath Provider, which completes and validates certificate chains by using trusted Certificate Authority based checking.
Custom authentication and user name/password tokens for inbound HTTP and message-level authentication	An Identity Assertion provider, possibly user-written or from a third-party, that can validate the token type. Make sure that you enable this provider to support the token.
X.509 tokens for inbound Web Services Security X.509 Token Authentication	If any of your proxy services or business services are web services that use abstract WS-Policy statements, you must also configure the following: In the Web Service Security configuration named <code>__SERVICE_BUS_INBOUND_WEB_SERVICE_SECURITY_MBEAN__</code> add the <code>UseX509ForIdentity</code> property and set it to <code>true</code> . See "Use X.509 Certificates to Establish Identity" in the <i>Oracle WebLogic Server Administration Console Online Help</i> .
SAML tokens	All of the following: <ul style="list-style-type: none"> • WebLogic SAML Identity Assertion Provider V2, which authenticates users based on Security Assertion Markup Language 1.1 (SAML) assertions. • WebLogic SAML Credential Mapping Provider V2, which maps Service Bus users to remote users.

3. If needed, in the Oracle WebLogic Server Administration Console, configure one or more Identity Assertion providers to handle the token types, such as X.509 or custom token types, for which you require support. For a description of all Identity Assertion providers that you can configure, see "Security Providers" in *Administering Security for Oracle WebLogic Server*.
4. If you plan to create proxy services or business services that require WS-Security digital signatures on inbound requests, enable the Certificate Registry provider, which is a Certification Path provider that validates inbound certificates against a list of certificates that you register.

See "Configure Certification Path Providers" in the *Oracle WebLogic Server Administration Console Online Help*.
5. If you configure message-level security (in inbound requests or outbound requests) to require user name and password tokens, and if you want messages to provide a password digest instead of cleartext passwords, do the following:

- a. In the Oracle WebLogic Server Administration Console, find the two Web Service Security configurations that Service Bus provides and set the value of the `UsePasswordDigest` property to `true`.

The Service Bus Web Service Security configurations are named:

`__SERVICE_BUS_INBOUND_WEB_SERVICE_SECURITY_MBEAN__` and
`__SERVICE_BUS_OUTBOUND_WEB_SERVICE_SECURITY_MBEAN__`

For information on setting the values in Web Service Security configurations, see "Use a Password Digest in SOAP Messages" in the *Oracle WebLogic Server Administration Console Online Help*.

- b. For each authentication provider that you configured, in the Oracle WebLogic Server Administration Console, select the Password Digest Enabled check box.
 - c. For each Identity Assertion provider that you configured, in the Oracle WebLogic Server Administration Console set `wsse:PasswordDigest` as one of the active token types.
6. If you plan to create a service key provider (which passes key-certificate pairs in outbound requests), use the Oracle WebLogic Server Administration Console to configure a PKI credential mapping provider. In any WebLogic Server domain that hosts Service Bus, you can configure at most one PKI credential mapping provider.

A PKI credential mapping provider maps Service Bus service key providers to key-pairs that can be used for digital signatures and encryption (for Web Services Security) and for outbound SSL authentication. For more information, see "Configuring a PKI Credential Mapping Provider" in *Administering Security for Oracle WebLogic Server*.

You store the key-pairs that the PKI credential mapping provider uses in a keystore. You can store the PKI credential mappings in WebLogic Server's identity keystore or in a separate keystore. Configure each WebLogic Server instance to have access to its own copy of each keystore. All entries referred to by the PKI credential mapper must exist in all keystores (same entry with the same alias). For information about configuring keystores in WebLogic Server, see "Identity and Trust" in *Understanding Security for Oracle WebLogic Server*.

Note:

When you create a Service Bus domain, by default the domain contains a user name/password credential mapping provider, which you can use if you need credential mapping for user names and passwords. In addition to this user name/password credential mapping provider, you can add one PKI credential mapping provider. A Service Bus domain can contain at most one user name/password credential mapping provider, one PKI credential mapping provider, and multiple SAML credential mapping providers.

7. If you want to enable security auditing, do the following:
 - a. In the Oracle WebLogic Server Administration Console, configure an auditing provider. See "Configuring the WebLogic Auditing Provider" in *Administering Security for Oracle WebLogic Server*.

- b. To enable auditing of events related to WS-Security, when you start each Service Bus server, include the following Java option in the server's startup command:

```
-Dcom.bea.wli.sb.security.AuditWebServiceSecurityErrors=true
```

Service Bus supports the auditing of security events but it does not support configuration auditing, which emits log messages and generates audit events when a user changes the configuration of any resource within a domain or invokes management operations on any resource within a domain. See "Enabling Configuration Auditing" in *Administering Security for Oracle WebLogic Server*.

- 8. If you have not already done so, in the Oracle WebLogic Server Administration Console, activate your changes. If you have made changes that require you to restart WebLogic Server, the Administration Console will indicate that a restart is required. If you see such a message, restart all WebLogic Server instances that host Service Bus so your modifications to the security providers will be in effect for the remaining configuration steps.

Context Properties Are Passed to Security Providers

Context properties provide a way (the `ContextHandler` interface) to pass additional information to the WebLogic Security Framework so a security provider can obtain contextual information beyond what is provided by the arguments to a particular provider method. A `ContextHandler` is a high-performing WebLogic class that obtains additional context and container-specific information.

Service Bus makes use of the `ContextHandler` interface and passes several context properties to the security framework for transport-level and message-level authentication, transport-level and message-level access control, and credential mapping.

This section describes the situations in which Service Bus-specific context properties are used.

Context Properties for HTTP Transport-Level Authentication

When an HTTP proxy service is configured for authentication, the HTTP transport provider passes a Service Bus implementation of the WebLogic Server `ContextHandler`. There is no user configuration required for this feature.

The `ContextHandler` properties in [Table 47-5](#) are passed at runtime, under the following conditions:

- To Authentication providers, if the proxy is configured for HTTP basic authentication.
- To Identity Assertion providers, if the proxy is configured for client certificate identity assertion.
- To Identity Assertion providers, if the proxy is configured for HTTP custom token identity assertion.

Table 47-5 ContextHandler Properties for HTTP Transport Authentication

Property Name	Type	Property Value
com.bea.contextelement. .alsb.service-info	com.bea.wli.sb.service s.ServiceInfo	An instance of <code>ServiceInfo</code> that contains information about the proxy service.
com.bea.contextelement. .alsb.transport.endpoint	com.bea.wli.sb.transpo rts.TransportEndPoint	This is the HTTP or HTTPS endpoint.
com.bea.contextelement. .alsb.transport.http.h ttp-request	javax.servlet.http.Htt pServletRequest	This is the <code>HttpServletRequest</code> object.
com.bea.contextelement. .alsb.transport.http.h ttp-response	javax.servlet.http.Htt pServletResponse	This is the <code>HttpServletResponse</code> object.

ContextHandler Properties for Access Control and Custom Authentication

The ContextHandler properties for access control and message-level custom authentication are passed at runtime, under the following conditions:

- To Authentication providers when performing message-level custom user name/password authentication.
- To Identity Assertion providers when performing message-level custom token identity assertion.
- To Authorization providers when performing transport-level or message-level access control.

Table 47-6 ContextHandler Properties for Custom Authentication and Access Control

Property Name	Type	Property Value
com.bea.contextelement. .alsb.router.ProxyService	java.lang.String	The service name (full-name; for example <code>/myproject/myfolder/svc-a</code>).
com.bea.contextelement. .alsb.router.ServiceUri	java.net.URI	The base URI from which the message was received.
com.bea.contextelement. .alsb.router.inbound.TransportPr ovider	java.lang.String	The Id of the transport provider that received this message.

Table 47-6 (Cont.) ContextHandler Properties for Custom Authentication and Access Control

Property Name	Type	Property Value
<code>com.bea.contextelement.alsb.router.inbound.request.MessageId</code>	<code>java.lang.String</code>	This is the transport provider-specific message identifier. Ideally it should uniquely identify the message among other messages going through the Service Bus runtime. However, Service Bus does not depend on the message Id being unique. The message Id is added to the message context and thus visible in the pipeline.
<code>com.bea.contextelement.alsb.router.inbound.request.CharacterEncoding</code>	<code>java.lang.String</code>	Character encoding used in the message payload, or null.
<code>com.bea.contextelement.wli.Message</code>	<code>java.io.InputStream</code>	The request message as an input stream.

Additional Transport-Specific Context Properties

In addition to the properties in [Table 47-7](#), other transport-specific properties may be present. For each transport request-header (see the transport SDK), a property with the name

```
com.bea.contextelement.alsb.router.inbound.request.headers.<provider-id>.<header-name>
```

is present, where `provider-id` is the transport provider id, and `header-name` is one of the request-headers declared in the provider's schema file. The type and semantics of these properties are transport-specific. For HTTP proxy services, the message-level security properties in [Table 47-3](#) are also available.

Table 47-7 Additional ContextHandler Properties for HTTP Proxy Services

Property Name	Type	Property Value
<code>com.bea.contextelement.alsb.router.inbound.request.metadata.http.relative-URI</code>	<code>java.lang.String</code>	The relative URI of the request.
<code>com.bea.contextelement.alsb.router.inbound.request.metadata.http.query-string</code>	<code>java.lang.String</code>	The query string that is contained in the request URL after the path.

Table 47-7 (Cont.) Additional ContextHandler Properties for HTTP Proxy Services

Property Name	Type	Property Value
com.bea.contextelement.alsb.router.inbound.request.metadata.http.client-host	java.lang.String	The fully qualified name of the client that sent the request.
com.bea.contextelement.alsb.router.inbound.request.metadata.http.client-address	java.lang.String	The Internet Protocol (IP) address of the client that sent the request.

Administrator-Supplied Context Properties for Message-Level Authentication

Both custom user name/password authentication and custom token authentication allow users (who are in the IntegrationAdmin or IntegrationDeployer roles) to pass additional context information to the security provider in the Context Properties field on the Security tab.

You can configure additional context properties by entering the **Property Name** as a literal string, and the **Value Selector** as a valid XPath expression. (XPath expressions can also be literal strings.)

The XPath expression is evaluated at runtime against the same message part that is used for the custom token or custom user name/password. That is, the **Value Selector** XPath expressions are evaluated against the header for SOAP-based proxy services, and against the body for non-SOAP-based proxy services.

Security Provider Must Have Knowledge of the Property Name

A ContextHandler is essentially a name/value list and, as such, it requires that a security provider know what names to look for. Therefore, for both transport- and message-level custom authentication, the XPath expressions are evaluated only if an Authentication provider or Identity Assertion provider asks for the value of one of these properties.

This means that your configured Authentication or Identity Assertion provider must explicitly know which property names to request through the ContextHandler.getValue(propertyName) method. The only way to satisfy this requirement is for you, or a third party, to write a custom Authentication or Identity Assertion provider.

The following example shows how to get the HttpServletRequest property from a provider that you write.

Example - Getting the HttpServletRequest Property

```

:
Object requestValue =
handler.getValue("com.bea.contextelement.alsb.transport.http.http-request");
if ((requestValue == null) || (!(requestValue instanceof HttpServletRequest)))
return;
HttpServletRequest request = (HttpServletRequest) requestValue;
log.println(" " + HTTP_REQUEST_ELEMENT + " method: " + request.getMethod());
log.println(" " + HTTP_REQUEST_ELEMENT + " URL: " + request.getRequestURL());

```

```
log.println(" " + HTTP_REQUEST_ELEMENT + " URI: " + request.getRequestURI());  
return;
```

If the security provider does not need the value of the user-defined property, then the XPath expression is not evaluated.

WebLogic Server Administrative Channel is Supported

This release of Service Bus can use the WebLogic Server administrative channel.

As described in "Understanding Network Channels" in *Administering Server Environments for Oracle WebLogic Server*, a WebLogic Server network channel is a configurable resource that defines the attributes of a network connection to WebLogic Server.

You can configure a particular type of network channel, called an administrative channel, to isolate "administration" and application ("business") traffic in a WebLogic domain. The administrative channel is a secured channel that accepts only SSL connections.

In Service Bus, business traffic is comprised of all messages sent to and from Service Bus proxy services and business services. SSL business traffic must use the default WebLogic Server secure network channel.

Administration traffic is comprised of all communication with the Oracle WebLogic Server Administration Console, Oracle Service Bus Console, internal traffic within a cluster, and traffic between administration scripts and admin or Managed Servers.

When an administrative channel is enabled in a domain, all of the administration traffic in that domain must go through that channel. Otherwise, the administration traffic also uses the default WebLogic Server secure network channel.

[Using the Administrative Channel: Main Steps](#) describes using the administrative channel.

Using the Administrative Channel: Main Steps

Complete the following steps to use the administrative channel:

1. Close any open browser connections to the Oracle Service Bus Console for the domain.

As soon as you activate the administrative channel in WebLogic Server, the Oracle Service Bus Console for the domain becomes unavailable at the current URL. The Help system also becomes unavailable.

2. Enable the domain-wide administration port in the Oracle WebLogic Server Administration Console (which configures an administrative channel on your behalf), or explicitly create an administrative channel. Both of these tasks are described in "Configuring Network Resources" in *Administering Server Environments for Oracle WebLogic Server*.

The domain-wide administration port control is located on the Domain > Configuration > General page. The default administration port is 9002.

Be sure to activate the change.

3. Open a browser connection to the new URL for the Oracle Service Bus Console for the domain.

The URL is `https://hostname:9002/servicebus` if you enabled the domain-wide administration port and accepted the default port number.

4. Revise any startup scripts that refer to the old URL. If you are using the Windows graphical interface to launch the Oracle Service Bus Console for the domain, revise the shortcut property to reflect the new URL.

Using Security Providers

This section provides instructions on using security providers with Service Bus.

This section includes the following topics:

- [Configuring Authentication Providers](#)
- [Using a Custom Authorization Provider to Protect Service Bus Resources](#)
- [About Errors When Using Security Provider Policies](#)

Configuring Authentication Providers

Check the provided WebLogic Server authentication providers to see if one meets your needs. WebLogic Server includes a broad array of Authentication providers, including the following:

- The WebLogic Authentication provider accesses user and group information in WebLogic Server's embedded LDAP server. This is the default out-of-the-box authentication provider. This provider is not optimized for use with very large numbers of users.
- *LDAP Authentication providers* access external LDAP stores. You can use an LDAP Authentication provider to access any LDAP server. WebLogic Server provides LDAP Authentication providers already configured for Open LDAP, Oracle Directory Server Enterprise Edition, Microsoft Active Directory, and Novell NDS LDAP servers.
- *RDBMS Authentication providers* access external relational databases. WebLogic Server provides three RDBMS Authentication providers: SQL Authenticator, Read-only SQL Authenticator, and Custom RDBMS Authenticator.
- The *SAML Authentication provider*, which authenticates users based on Security Assertion Markup Language 1.1 (SAML) assertions.

See "Improving the Performance of WebLogic and LDAP Authentication Providers" in *Administering Security for Oracle WebLogic Server* for guidance on improving the performance of these authentication providers.

As described in "Why Customize the Default Security Configuration" in *Administering Security for Oracle WebLogic Server*, you may want to use an Authentication provider that accesses a database other than WebLogic Server's embedded LDAP server. For example, you might want to use a different authentication provider for the majority of user accounts, but continue to use the default authentication provider (embedded LDAP) for Service Bus and WebLogic Server administrative user accounts.

Using the WebLogic Authentication provider for all WebLogic Server and Service Bus administrative user accounts provides reliable access in the event of a network or database problem. Oracle recommends that you use the default WebLogic Authentication provider for all WebLogic Server and Service Bus administrative accounts for this reason.

If one of the bundled Authentication providers meets your needs, see "Configuring Authentication Providers" in *Administering Security for Oracle WebLogic Server* for instructions on how to configure this Authentication provider in the Oracle WebLogic Server Administration Console.

If none of the Authentication providers included in WebLogic Server suits your needs, you (or a third-party) must first write a custom Authentication provider and then use the Oracle WebLogic Server Administration Console to add that provider to the security realm, as described in the following steps:

Note:

Only a broad overview of the required tasks is included here. You will need to consult the WebLogic Server documentation to actually complete the tasks.

To add a provider to a security realm:

1. "Create Runtime Classes Using the Appropriate SSPIs" (in *Developing Security Providers for Oracle WebLogic Server*).
2. "Generate an MBean Type Using the WebLogic MBeanMaker"
3. "Configure the Custom Authentication Provider Using the Administration Console"

Using a Custom Authorization Provider to Protect Service Bus Resources

You can use Service Bus resources with custom Authorization providers, but those providers must understand the type and format of the Service Bus resources.

There are three possible resource objects for Service Bus that an Authorization provider must be able to detect and handle:

- [ALSBProxyServiceResource Object](#)
- [ProjectResourceV2 Object](#)
- [ConsoleResource Object](#)

These resource objects are described in the sections that follow.

WebLogic Authorization Provider Usage Information

This section briefly describes the WebLogic Server authorization provider SSPI. See "Authorization Providers" in *Developing Security Providers for Oracle WebLogic Server* for complete information.

You protect resources by binding access control policies to resources using the Oracle Service Bus Console, third-party tools or scripts. The WebLogic Server Security Service Provider Interface (SSPI) requires containers, such as Service Bus, to implement the Resource SPI. These implementations represent concrete resources.

The Authorization provider database contains a map from resource to policy. When an attempt is made to access a resource, the container calls the runtime SSPI to get an access control decision. The container passes a resource instance indicating which resource is being accessed.

An Authorization provider has one method, `getAccessDecision()`. The `getAccessDecision()` method obtains the implementation of the `AccessDecision`

SSPI. The `AccessDecision` SSPI itself has one method, `isAccessAllowed()`. `isAccessAllowed` has five parameters, one of which is the `Resource` object for which access is being requested.

`isAccessAllowed` determines if the requestor should be allowed to access the named resource. To do this, the Authorization provider must find the right access control policy to evaluate. The provider must first look for a policy bound to the resource passed in. The lookup can use either the `getId()` or `toString()` method as a lookup key, as described in "Looking Up WebLogic Resources in a Security Provider's Runtime Class" in *Developing Security Providers for Oracle WebLogic Server*. If no policy is found, the Authorization provider must then get the parent resource and look again. This process is repeated until a policy is found or the parent is null, in which case no policy is found. When no policy is found, `isAccessAllowed` must return false.

This algorithm allows you to create coarse-grained policies that protect all proxy services in a given project or folder, all resources in a project, or all Service Bus proxy services in a Service Bus domain. More specific, finer-grained policies take precedence over coarse-grained policies.

Note:

The Oracle Service Bus Console user interface does not provide pages for protecting proxy services at the folder, project or domain level.

ALSBProxyServiceResource Object

The `ALSBProxyServiceResource` object is used for transport-level and message-level access control to Service Bus proxy services. The `ALSBProxyServiceResource` resource extends `weblogic.security.service.ResourceBase`, which itself implements `weblogic.security.spi.Resource`.

`ALSBProxyServiceResource` implements the following methods, as described in `weblogic.security.spi.Resource`:

getType() – Returns the type, where type is "<alsb-proxy-service>"

getKeys() – Returns up to four key-value properties: `path`, `proxy`, `action`, and `operation`. The properties are defined as follows:

- `path` is the full-name of the proxy service. For example, `path=project/folder1/folder2`
- `proxy` is the name of the proxy service. For example, `proxy=myProxy`
- `action` is one of two values, `invoke` or `wss-invoke`. For example, `action=invoke`

The `action` attribute is used to distinguish between transport-level and message-level access control. `invoke` is used for transport-level access control. `wss-invoke` is used for message-level access control; that is, access control on WS-Security active intermediaries or proxies with custom message-level authentication. The `operation` attribute is only allowed when `action` is `wss-invoke`.

- `operation` is the name of the operation to invoke, and is used only when `action` is `wss-invoke`. For example, `operation=processPO`. The `operation` attribute is only allowed when `action` is `wss-invoke`.

An `ALSBProxyServiceResource` has from 1 to 4 keys. The following table explains how the various combinations protect proxy services. The most specific policies take precedence.

If the Resource Contains These Keys	A Policy Bound to the Resource Protects:
path	The policy protects all proxy services in the given path
path and proxy	The policy protects all access to the given proxy service (transport-level as well as message-level)
path, proxy, and action	If action="invoke": The policy is the transport-level policy to the given proxy If action="wss-invoke": The policy is the message-level policy to the given proxy (for all operations)
path, proxy, action="wss-invoke", and operation	The policy is a message-level policy for the given proxy and operation

getPath() – Gets the path (project and folders) to the proxy service. This is the path where the proxy service exists within the Service Bus configuration framework.

getProxyServiceName() – Gets the name of the proxy service. For example, `proxy=myProxy`.

getAction() – Gets one of two values, `invoke` or `wss-invoke`. For example, `action=invoke`.

getOperation() – Gets the name of the operation to invoke, and is used only when action is `wss-invoke`. For example, `operation=processPO`.

makeParent() – Creates a new `ALSBProxyServiceResource` object that represents the parent of the current `ALSBProxyServiceResource` resource. `makeParent()` uses the path of the proxy service to create the parent.

ALSBProxyServiceResource Examples

The following examples show various uses of the `ALSBProxyServiceResource` object.

- Using `ALSBProxyServiceResource` for transport-level access control for proxy `project/folder/myProxy`:
`type=<alsb-proxy-service>, path=project/folder, proxy=myProxy, action=invoke`
- Using `ALSBProxyServiceResource` for message-level access control for operation `processPO` on proxy `project/folder/myProxy`:
`type=<alsb-proxy-service>, path=project/folder, proxy=myProxy, action=wss-invoke, operation=processPO`
- Using the parentage hierarchy for an `ALSBProxyServiceResource`, from fine-grained to coarse-grained:
`type=<alsb-proxy-service>, path=myProject/f1/f2, proxy=myProxy, action=wss-invoke, operation=foo`
`type=<alsb-proxy-service>, path=myProject/f1/f2, proxy=myProxy, action=wss-invoke`

```
type=<alsb-proxy-service>, path=myProject/f1/f2, proxy=myProxy
type=<alsb-proxy-service>, path=myProject/f1/f2
type=<alsb-proxy-service>, path=myProject/f1
type=<alsb-proxy-service>, path=myProject
type=<alsb-project>, project-name=myProject
type=<alsb-proxy-service>
```

ProjectResourceV2 Object

The `ProjectResourceV2` is the root resource for all `ALSBProxyServiceResource` objects in a given project. `ProjectResourceV2` extends `ResourceBase`.

Setting an access control policy on a `ProjectResourceV2` provides a coarse-grained access control policy for all proxy services in the given project that do not have more specific policies.

`ProjectResourceV2` has the following methods:

getType() – Returns the type, where type is "`<alsb-project>`".

getKeys() – Returns the key, where key is "`project-name`".

getName() – Gets the name of the `ProjectResourceV2` object.

makeParent() – There is no parent for an `ProjectResourceV2` object. This method therefore returns the object name that was used to create the `ProjectResourceV2` object, or null if `ProjectResourceV2` does not exist.

ConsoleResource Object

The `com.bea.wli.security.resource.ConsoleResource` object is used for access control to the Oracle Service Bus Console. However, we do not recommend that you set access control policies for `ConsoleResource` objects using a custom Authorization provider. This is because these policies are subject to change in future Service Bus releases.

We instead recommended that even if you need to use a custom Authorization provider, you also continue to use the WebLogic Server XACML Authorization provider to maintain the policies for the `ConsoleResource` object. In this case of two Authorization providers, you must also configure an Adjudication provider.

About Errors When Using Security Provider Policies

If you are using a plug-in security provider with WebLogic Server to store policies for use with Service Bus, you may encounter an error that says Service Bus cannot determine whether or not required policies are available.

An error message like that does not necessarily mean the policies do not exist, or that you have a connection or configuration problem with the security provider. Service Bus uses a WebLogic Server SSPI to read policies that security providers can implement. However, the SSPI read functionality is optional. It is possible that a security provider does not allow read access by not implementing this SSPI. In such a case, Service Bus cannot reliably determine whether or not the security provider contains the required policies, even when the required policies could very well exist in the security provider.

To determine whether or not such a warning indicates a real problem, try creating or modifying resources in the Oracle Service Bus Console. Also, try securing a proxy service with an access control policy and test it. For information on configuring an access control policy on a proxy service, see [Configuring Service Bus Client Access Security](#). If you can successfully create or manipulate resources as well as test a

secured proxy service while using the security provider, then the security provider is configured correctly and you can safely ignore the error message.

Oracle Service Bus Security FAQ

This chapter provides answers to frequently asked questions about Service Bus security.

This chapter includes the following sections:

- [How are Service Bus and WebLogic Server Security related?](#)
- [What is Transport-Level Security?](#)
- [What is Web Services Security?](#)
- [What is Web Service Policy?](#)
- [What are Web Service Policy assertions?](#)
- [Are Access Control Policy and Web Service Policy the same?](#)
- [What is Web Services Security Pass-Through?](#)
- [What is a Web Services Security Active Intermediary?](#)
- [What is outbound Web Services Security?](#)
- [What is SAML?](#)
- [What is the Certificate Lookup And Validation Framework?](#)
- [Does Service Bus support identity propagation in a proxy service?](#)
- [Is it possible to customize the format of the subject identity in a SAML assertion?](#)
- [Is single sign-on supported in Service Bus?](#)
- [Are security errors monitored?](#)
- [Can I configure security for MBeans?](#)

How are Service Bus and WebLogic Server Security related?

Service Bus leverages the WebLogic Security Framework. The details of this framework are described in "WebLogic Security Framework" in "WebLogic Security Service Architecture" in *Understanding Security for Oracle WebLogic Server*. Before configuring security in Service Bus, you must configure an WebLogic Server security realm and other server configurations (such as SSL) in WebLogic Server, as described in [Configuring the Oracle WebLogic Security Framework: Main Steps..](#)

What is Transport-Level Security?

Transport-level security refers to the transport protocols that secure the connection over which messages are transported. An example of transport-level security is HTTPS (HTTP over SSL). SSL provides point-to-point security, but does not protect the message when intermediaries exist in the message path. For more information, see [Configuring Transport-Level Security](#).

What is Web Services Security?

Web Services Security (WS-Security) is an OASIS standard that defines interoperable mechanisms to incorporate message-level security into SOAP messages. WS-Security supports message integrity and message confidentiality. It also defines an extensible model for including security tokens in a SOAP envelope and a model for referencing security tokens from within a SOAP envelope. WS-Security token profiles specify how specific token types are used within the core WS-Security specification. Message integrity is achieved through the use of XML digital signatures; message confidentiality is accomplished through the use of XML encryption. WS-Security allows you to specify which parts of a SOAP message are digitally signed or encrypted. Service Bus supports WS-Security over HTTP (including HTTPS) and JMS.

Note:

Oracle Web Services Manager (OWSM) is the Web Services security mechanism used by Oracle Service Bus. All newly created resources, such as business services and proxy services, use OWSM policies for security. WLS 9 policies are deprecated, and cannot be used to configure security for a new Service Bus resource.

However, you can import resources already configured with WLS 9 policies into your Service Bus project. You cannot edit or modify these WLS 9 policies, but you can replace them with OWSM policies.

What is Web Service Policy?

The Web Services Policy Framework (WS-Policy) provides a general-purpose model and corresponding syntax to describe and communicate the policies of a web service. WS-Policy defines a base set of constructs that can be used and extended by other web service specifications to describe a broad range of service requirements, preferences, and capabilities. See the note in [What is Web Services Security?](#) for important information about WLS9 web service policies. For more information, see [Securing Oracle Service Bus Proxy and Business Services with WS-Policy](#).

What are Web Service Policy assertions?

The Web Services Policy Assertions Language (WS-PolicyAssertions) specifies a set of common message policy assertions that can be specified within a security policy. The specification defines general messaging-related assertions for use with WS-Policy. Separate specifications describe the syntax and semantics of domain-specific assertions for security assertions and reliable-messaging assertions. See the note in [What is Web Services Security?](#) for important information about WLS9 web service policies.

Are Access Control Policy and Web Service Policy the same?

No. Access control policy is a boolean expression that is evaluated to determine which requests to access a particular resource (such as a proxy service, web application, or EJB) are granted and which should be denied access. Typically access control policies are based on the *roles* of the requestor. WS-Policy is metadata about a web service that complements the service definition (WSDL). WS-Policy can be used to express a requirement that all service clients must satisfy, such as, all requests must be digitally signed by the client.

What is Web Services Security Pass-Through?

In a WS-Security pass-through scenario, the client applies WS-Security to the request and/or response messages. The proxy service does not process the security header, instead, it passes the secured request message untouched to a business service. Although Service Bus does not apply any WS-Security to the message, it can route the message based on values in the header. After the business service receives the message, it processes the security header and acts on the request. The business service must be configured with WS-Policy security statements. The secured response message is passed untouched back to the client. For example, the client encrypts and signs the message and sends it to the proxy service. The proxy service does not decrypt the message or verify the digital signature, it simply routes the message to the business service. The business service decrypts the messages and verifies the digital signature, and then processes the request. The response path is similar. This is sometimes called a passive intermediary.

What is a Web Services Security Active Intermediary?

In an active intermediary scenario, the client applies WS-Security to the request and/or response messages. The proxy service processes the security header and enforces the WS-Security policy. For example, the client encrypts and signs the message and sends it to the proxy service, the proxy decrypts the message and verifies the digital signature, then routes the message. Before the proxy service sends the response back to the client, the proxy signs and encrypts the message. The client decrypts the message and verifies the proxy's digital signature.

What is outbound Web Services Security?

Outbound WS-Security refers to security between Service Bus proxy services and business services. It includes both the request and response between business applications and proxy services. For more information, see [About Message-Level Security](#).

What is SAML?

SAML (Security Assertion Markup Language) is an OASIS standards-based extensible XML framework for exchanging authentication and authorization information, allowing single sign-on capabilities in modern network environments.

Is it possible to customize the format of the subject identity in a SAML assertion?

By default, the subject identity within an outbound SAML token is the same as the inbound user name. The format of the subject identity can be customized by writing a custom SAML name mapper-provider. For more information, see "Configuring a SAML Credential Mapping Provider" in *Administering Security for Oracle WebLogic Server*.

What is the Certificate Lookup And Validation Framework?

The Certificate Lookup and Validation (CLV) providers complete certificate paths and validate X509 certificate chains. The two types of CLV providers are:

CertPath Builder—receives a certificate, a certificate chain, or certificate reference (the end certificate in a chain or the Subject DN of a certificate) from a web service or application code. The provider looks up and validates the certificates in the chain.

CertPath Validator—receives a certificate chain from the SSL protocol, a web service, or application code and performs extra validation, such as revocation checking.

At least one CertPath Builder and one CertPath Validator must be configured in a security realm. Multiple CertPath Validators can be configured in a security realm. If multiple providers are configured, a certificate or certificate chain must pass validation with all the CertPath Validators for the certificate or certificate chain to be valid. WebLogic Server provides the functionality of the CLV providers in the WebLogic CertPath provider and the Certificate Registry. For more information see "The Certificate Lookup and Validation Process" in "WebLogic Security Service Architecture" in *Understanding Security for Oracle WebLogic Server*.

Does Service Bus support identity propagation in a proxy service?

Yes, Service Bus supports two methods for propagating identities:

- By generating SAML assertions in conformance with the Web Services Security. This is done by setting a SAML holder-of-key or sender-vouches WS-Policy on the business service routed to by the proxy.
- If a business service requires user name and password tokens, you can configure the business service's service account to pass through the user credentials from the original client request. See [Working with Service Accounts](#).

Is single sign-on supported in Service Bus?

Strictly speaking single sign-on (SSO) is not applicable to Service Bus messaging scenarios for several reasons. First, Service Bus is stateless; there is no notion of a session or conversation among multiple parties. Second, Service Bus clients are typically other enterprise software applications, not users behind a web browser. Therefore, it is acceptable to require that these clients send credentials such as user name and password on every request, provided that the communication is secured by means such as SSL or WS-Security. However, SSO between the Oracle Service Bus Console and the Oracle WebLogic Server Administration Console is supported. For more information, see "Single Sign-On" in "Security Fundamentals" in *Understanding Security for Oracle WebLogic Server*.

Are security errors monitored?

Only WS-Security errors are monitored by the Service Bus monitoring framework. Transport-level security errors such as SSL handshake errors, transport-level authentication and transport-level access control are not monitored. However, it is possible to configure an Auditor provider to audit transport-level authentication and authorization. For more information, see *Monitoring and Managing Security Policies* in *Administer Oracle Service Bus*.

Can I configure security for MBeans?

Service Bus includes two managed beans (MBeans) that configure such runtime behavior as which types of credentials are available to abstract WS-Policy statements. By default, only users in the Admin and Deployer security roles can modify these MBeans, however you can change these defaults.

Securing Business and Proxy Services

This chapter describes how to attach policies to business services and proxy services in Service Bus applications. Policies apply security to the delivery of messages.

This chapter includes the following sections:

- [Introduction to Policies](#)
- [Security and Security Policies for Business and Proxy Services](#)
- [Attaching and Configuring Policies in JDeveloper](#)
- [Attaching and Configuring Policies in the Oracle Service Bus Console](#)
- [Configuring Service Bus Client Access Security](#)
- [Hiding Personally Identifiable Information in Messages](#)

Introduction to Policies

Oracle Fusion Middleware uses a policy-based model to manage and secure web services across an organization. Policies apply security to the delivery of messages, and can be managed by both developers in a design-time environment and system administrators in a runtime environment.

Policies are comprised of one or more assertions. A policy assertion is the smallest unit of a policy that performs a specific action. Policy assertions are executed on the request message and the response message, and the same set of assertions is executed on both types of messages. The assertions are executed in the order in which they appear in the policy.

[Table 49-1](#) describes the supported policy categories.

Table 49-1 Supported Policy Categories

Category	Description
Message Transmission Optimization Mechanism (MTOM)	Ensures that attachments are in MTOM format. This format enables binary data to be sent to and from web services. This reduces the transmission size on the wire.
Security	Implements the WS-Security 1.0 and 1.1 standards. They enforce authentication and authorization of users, identity propagation, and message protection (message integrity and message confidentiality).
Management	Logs request, response, and fault messages to a message log. Management policies can also include custom policies.
Personally Identifiable Information (PII)	Encrypts and decrypts certain fields to protect personally identifiable information.

Note:

JDeveloper displays two additional categories of policies, Reliability and Addressing. Service Bus does not currently support these policies. In the Oracle Service Bus Console, PII and MTOM policies are grouped in the Security category.

Within each category there are one or more policy types that you can attach. When looking at the list of policies, you can click an information icon to see a description of each policy.

Security and Security Policies for Business and Proxy Services

You can secure access to proxy and business services using Oracle Web Services Manager (OWSM) policies. You can also define transport-level and message-level security in the proxy service configuration, and transport-level security in the business service configuration. For information about OWSM policies, see [Securing Oracle Service Bus with Oracle Web Services Manager](#).

A service provider is required if the proxy service routes messages to HTTPS services that require client certificate authentication and may be required in some message-level security scenarios. A service account can be created to provide authentication when connecting to a business service. It acts as an alias resource for the required user name and password pair. WebLogic Server can be used to directly manage security credentials for a business service requiring credential-level validation.

Security Policies in Service Bus

You can attach OWSM policies to a proxy or business service with a service type of WSDL Web Service, Messaging Service, Any SOAP Service, or Any XML Service. In order for OWSM policies to be used with non-SOAP WSDL Web Service, Messaging Service, or Any XML Service proxy services, the protocol must be HTTP. For WSDL-based services, OWSM policies are bound by reference and not inlined in the effective WSDL file. OWSM policies support a variety of industry standards, including WS-Security 1.1, SAML 2.0, and KerberosToken Profile.

In previous versions, Service Bus accepted security policies from the WSDL file and from policies predefined in WebLogic Server. These policies are replaced by OWSM policies in 12c. When you import projects from previous versions that use WSDL-defined or WLS policies, the policies display as read-only and cannot be modified. The information appears in the proxy or business service configuration so you can update the service to OWSM policies.

Policy Overrides

Certain OWSM policies let you configure override values for runtime properties. If you are configuring a proxy service in the Oracle Service Bus Console with OWSM policies, policy override options appear below any attached policies that support overrides. In JDeveloper, the Edit icon brings up a dialog where you can configure overrides. For more information, see [Securing Oracle Service Bus with Oracle Web Services Manager](#).

Security Settings

Service Bus provides additional security features for business and proxy services, like specifying custom authentication for access to the service, transport-level security, and, for proxy services only, message-level security. You can find additional information about the specific settings in the online help provided for the security and policies pages. For more information about these options, see the following chapters:

- [Configuring Message-Level Security for Web Services](#)
- [Configuring Transport-Level Security](#)
- [Configuring Custom Authentication](#)

Global Policies

When you apply OWSM policies to a service in JDeveloper or the Oracle Service Bus Console, you assign them directly to that service. You can also assign policies to multiple JCA, REST, and SOAP services in a Service Bus project using global policy sets in Fusion Middleware Control. For more information, see "Global Policies" in *Administering Oracle Service Bus*. For information about global policy attachments and policy sets, see "Global Policy Attachments Using Policy Sets" in *Understanding Oracle Web Services Manager*.

Service Accounts in Business Services

If any of a business service's WS-Policies specify authentication, you can select a service account to specify credentials when making an outbound request. A proxy service that routes to this business service uses this service account to authenticate to the business service. Service account credentials are supported for the following OWSM policies:

- `oracle/**_username_token_**_client_policy`
- `oracle/wss11_saml_token_identity_switch_with_message_protection_client_policy`
- `oracle/**_saml*_**_client_policy` (only by setting the `subject.precedence` property to false)

Service account credentials can also be used for the following OWSM policy assertions:

- `oracle/**_username_token_**_client_template`
- `oracle/**_saml*_**_client_template` (only by setting `subject.precedence` property to false)

Note:

If both a service account and the `csf-key` override are specified for a business service, the `csf-key` credentials take precedence.

Security-Related Validation for Active Proxy Services

When you use the Oracle Service Bus Console to activate a session that contains changes to an active proxy service, Service Bus validates the changes to ensure that you have created all of the credentials that the proxy service's static endpoints require. If a session contains a change to the key-pair bindings of a service key provider, Service Bus validates the change against all of the proxy services that use the service key provider. For example, if you remove the encryption key-pair, Service Bus reports a validation error for any proxy service that references the service key provider and whose endpoint requires encryption.

The following criteria determine when Service Bus performs this security-related validation and the actions that it takes during validation:

- If a proxy service specifies a static route and operation, Service Bus determines which credentials the static route and operation require. If the proxy service is missing the required credentials, Service Bus will not commit the session until you add the missing credentials.
- If a proxy service specifies a static route but the operation is passed through from the inbound request, Service Bus determines which credentials the static route and each of the route's operations require. If the proxy service is missing the required credentials, Service Bus issues a validation warning but allows you to commit the session.
- If a proxy service specifies a dynamic route and operation, Service Bus cannot validate the security requirements and you risk the possibility of runtime errors. For information about dynamic routing, see [Using Dynamic Routing](#).

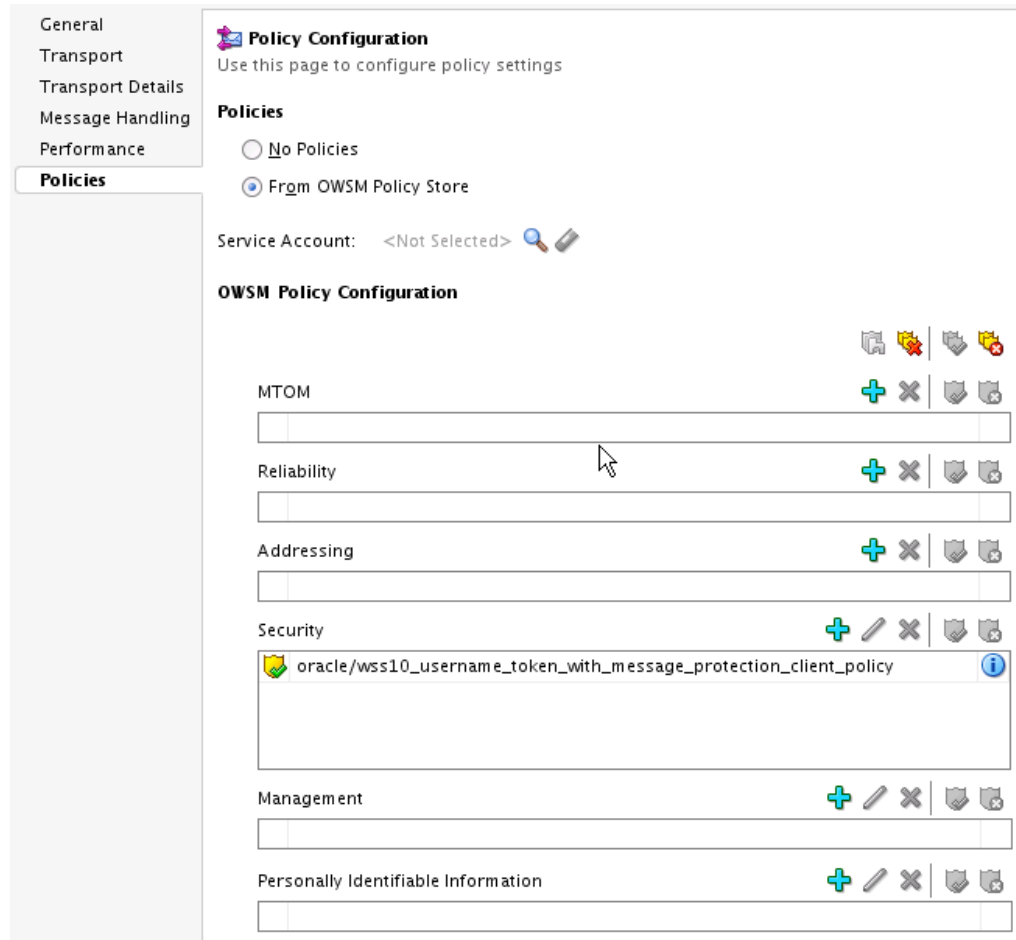
Attaching and Configuring Policies in JDeveloper

In JDeveloper, you can attach policies for testing security in a design-time environment. When your application is ready for deployment to a production environment, you can attach or detach runtime policies in Oracle Enterprise Manager Fusion Middleware Control. For more information about runtime management of policies, see "Monitoring and Managing Security Policies" in *Administering Oracle Service Bus*.

You can only attach OWSM policies to business and proxy services with specific configurations. Depending on the service type and protocol, some policy options may not be available. For information about supported configurations, see [Security Policies in Service Bus](#). For information about when service accounts are used, see [Service Accounts in Business Services](#).

For services created in previous versions of Service Bus, if the service is created from a WSDL file that includes WS-Policy attachments, the policies are displayed read-only on the service's Policies page.

The following image shows the Policies page for business services in JDeveloper. This image shows all categories, but the actual categories displayed depend on the service type and protocol of the service.

Figure 49-1 Policy Configuration Page for Business Services in JDeveloper

How to Attach Oracle Web Services Manager Policies in JDeveloper

When you attach policies to a proxy or business service in JDeveloper, those policies are not validated until they are deployed to the WebLogic Server. For more information about OWSM, see [Securing Oracle Service Bus with Oracle Web Services Manager](#).

Note:

If the service was upgraded from a previous version and includes WLS 9 policies, you can view but not edit those policies. These policies are deprecated. Use the steps in this section to update the policies in the upgraded services to OWSM policies.

To attach Oracle Web Services Manager Policies in JDeveloper;

1. In the Application Navigator, locate the business or proxy service you want to edit and double-click the service's file.

The Business or Proxy Service Definition Editor appears.

2. Click the **Policies** tab.

3. On the Policies page, select **From OWSM Policy Store** in the list of available policy binding models.

The available categories appear. These depend on the service type of the proxy or business service.

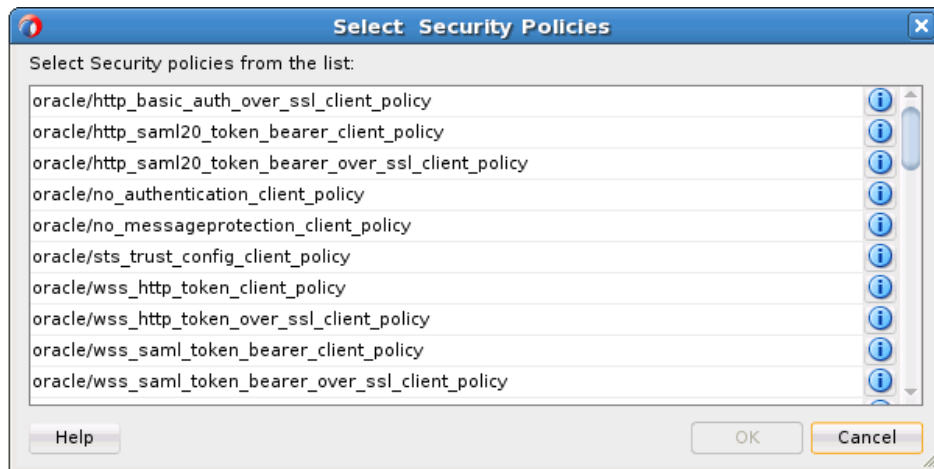
4. In the category of the policy you want to add, click **Add a * Policy**.

A dialog appears with a list of policies you can select. The dialog for Security policies is shown below.

Note:

If there is only one policy available in the chosen category, the Select * Policies dialog does not appear; instead the available policy is populated directly into the select policies table.

Figure 49-2 Select Security Policies Dialog in JDeveloper



5. If the Select * Policies dialog appeared, do the following:
 - a. To view information about a specific policy, click the information icon to the right of the policy name.
 - b. Select the policies you want to attach.
Use the **Ctrl** and **Shift** keys to select multiple policies.
 - c. Click **OK**.

The policy is added to the relevant category on the definition editor.

6. To temporarily disable a policy, select the policy and then click **Disable selected policy** above the table containing the policy. To temporarily disable all policies, click **Disable all policies**.
7. To re-enable a policy, select the policy and then click **Enable selected policy** above the table containing the policy. To re-enable all policies, click **Enable all policies**.

8. To remove a policy added in error, select the policy and then click **Remove selected policies** for that category. Click **Remove all policies** to remove all attached policies.
9. To view a description and additional information for a policy, click **Show Details** next to that policy.
10. If you are attaching policies to a business service, optionally browse to and select a service account from the **Service Account** field.
11. When you are done configuring policies, click **Save**.

How to Define Override Values for a Policy in JDeveloper

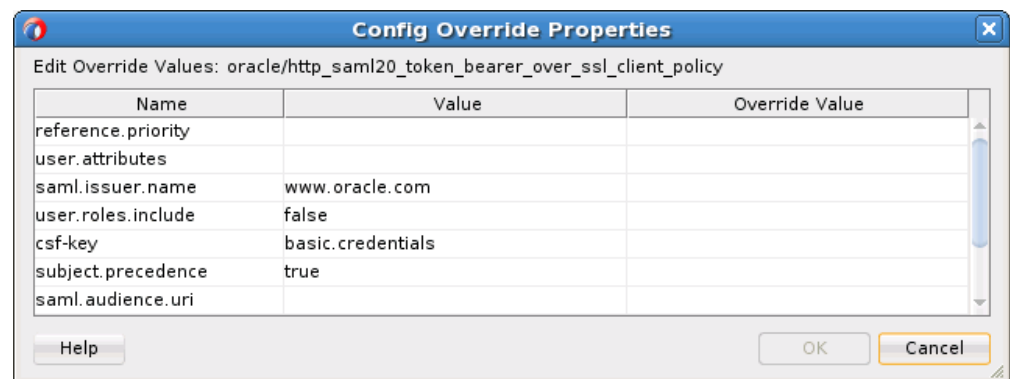
Your environment may include services that use the same policies. However, each service might have specific policy requirements, which you can specify using override properties. Not all policies allow override values.

To define override values for a policy in JDeveloper:

1. Select the policy and then click **Edit Config Override Properties**.

The Config Override Properties dialog appears.

Figure 49-3 Config Override Properties Dialog for OWSM Policies in JDeveloper



2. In the **Override Value** column, enter a value to override the default value listed in the **Value** column for each property you want to configure.
3. Click **OK**.
4. When you are done configuring override values, click **Save**.

How to Configure Custom Authentication for Proxy Services in JDeveloper

Custom authentication lets you specify custom user name and password combinations or custom tokens. You may need to specify the custom user name and password or token in XPath format. The format for both is similar in that you specify XPath expressions that enable Service Bus to locate the necessary information. The root of these XPath expressions is as follows:

- Use `soap-env:Envelope/soap-env:Header` if the service binding is AnySOAP or WSDL-SOAP.
- Use `soap-env:Body` if the service binding is not SOAP based.

All XPath expressions must be in a valid XPath format. The XPath expressions must use the XPath "declare namespace" syntax to declare any namespaces used, as follows:

```
declare namespace
ns='http://webservices.mycompany.com/MyExampleService';)
```

Note:

Not all fields and tasks described below are available for all service types. The configuration depends on the service type and policy configuration of the service.

You can also configure custom authentication for proxy and business services at the transport level. For more information, see [Configuring Custom Authentication Transport-Level Security](#).

Configuring Proxy Service Custom Authentication in JDeveloper

To configure proxy service custom authentication in JDeveloper:

1. In the Application Navigator, locate the proxy service you want to edit and double-click the service's file.

The Proxy Service Definition Editor appears.

2. Click the **Security** tab.
3. Do one of the following:
 - To specify the XPaths to the user name and password, select **Custom User Name and Password**. Use the Expression Editor to define the XPath for the user name and password.
 - To specify a token, select **Custom Token**, select a token type, and then use the Expression Editor to define the XPath to the token.

Note:

REST proxy services do not currently support message-level custom token authentication.

4. Optionally, you can specify context properties to pass additional information to the context provider.

For more information, see [Context Properties Are Passed to Security Providers](#). For more information about custom authentication, see [Configuring Custom Authentication](#).

5. When you are done configuring the security settings, click **Save**.

How to Specify a Service Key Provider for a Proxy Service in JDeveloper

A service key provider contains Public Key Infrastructure (PKI) credentials that proxy services use for decrypting inbound SOAP messages and for outbound authentication and digital signatures. The service key provider resource used by the proxy service must be created before you can perform this step. For more information, see [Working with Service Key Providers](#).

To specify a service key provider for a proxy service in JDeveloper:

1. In the Application Navigator, locate the proxy service you want to edit and double-click the service's file.

The Proxy Service Definition Editor appears.

2. Click the **Security** tab.
3. Click the **Browse** or **Search** icon next to the **Service Key Provider** field to locate and select a service key provider to use.
4. When you are done configuring the security settings, click **Save**.

How to Specify Web Services Policy Enforcement in JDeveloper

When a proxy service passes through the security header without processing it, it is known as a passive intermediary. For more information about web services security pass-through, see [What is Web Services Security Pass-Through?](#)

To web services policy enforcement in JDeveloper:

1. In the Application Navigator, locate the proxy service you want to edit and double-click the service's file.

The Proxy Service Definition Editor appears.

2. Click the **Security** tab.
3. Do one of the following:
 - If the proxy service should not process the security header, select **Passive Security Intermediary**.
 - If the proxy service should process the security header, clear the **Passive Security Intermediary** check box.
4. When you are done configuring the security settings, click **Save**.

Attaching and Configuring Policies in the Oracle Service Bus Console

You can only attach OWSM policies to business and proxy services with specific configurations. Depending on the service type and protocol, some policy options may not be available. For information about supported configurations, see [Security Policies in Service Bus](#).

For services created in previous versions of Service Bus, if the service is created from a WSDL file that includes WS-Policy attachments, the policies are displayed read-only on the service's Policies page.

The following image shows the Policies page for business services in the Oracle Service Bus Console. This image shows all categories, but the actual categories displayed depend on the service type and protocol of the service.

Figure 49-4 Policy Configuration Page for Proxy Services in the Oracle Service Bus Console

Proxy Service Definition ◀ 🔍 ▶ 🔔 🗨 ▶ 🔗

Configuration **Security** SLA Alert Rules

Policies

No Policies

From OWSM Policy Store

Service Level Policies 🔗 ✖

Name	Category	Status	Description
oracle/http_basic_auth_over_ssl_service_policy	Security	Enabled	🗨

Policy Overrides

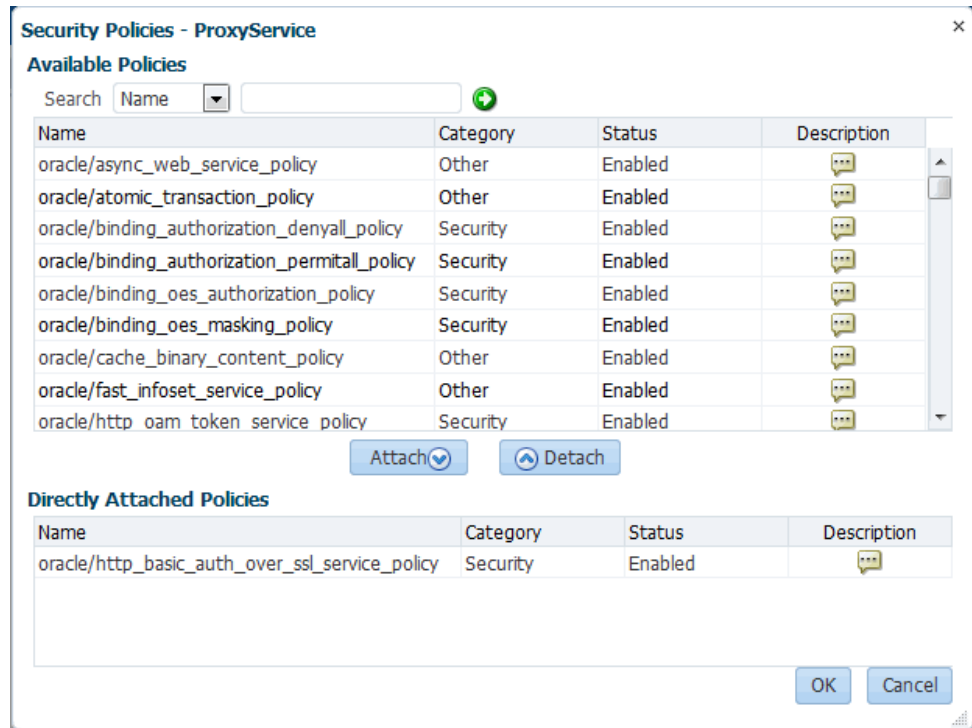
Policy Name	Property	Default Value	Override Value
oracle/http_basic_auth_over_ssl_service_policy	reference.priority	[No Policy Default]	

How to Attach Oracle Web Services Manager Policies in the Console

For more information about OWSM, see [Securing Oracle Service Bus with Oracle Web Services Manager](#).

To attach Oracle Web Services Manager policies in the console:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.
2. In the Project Navigator, locate the business or proxy service and click the service name.
The Business or Proxy Service Definition Editor appears.
3. Click the **Security** tab, and then make sure **Policies** is selected.
4. On the Policies page, select **From OWSM Policy Store** in the list of available policy binding models.
5. In the Service Level Policies table, click **Attach Policies**.
The Security Policies dialog appears, as shown below.

Figure 49-5 Security Policies Dialog in the Oracle Service Bus Console

6. Do the following to perform a search for policies to attach:
 - a. Select a type and enter the name of either the category or the policy to find.
 - b. Click **Search**.
 - c. When you find the policy to attach, select it in the results list and then click **Attach**.
 - d. You can attach multiple policies. When you are done, click **OK**.
7. For business services only: To select a service account that contains credentials for the business service, click **Browse** next to the **Service Account** field, and then browse to and select the service account to use.

Note:

The service account resource must already be created in Service Bus in order to select it here.

8. When you are done configuring policies, click **Save**.
9. To activate the changes in the runtime, click **Activate**.

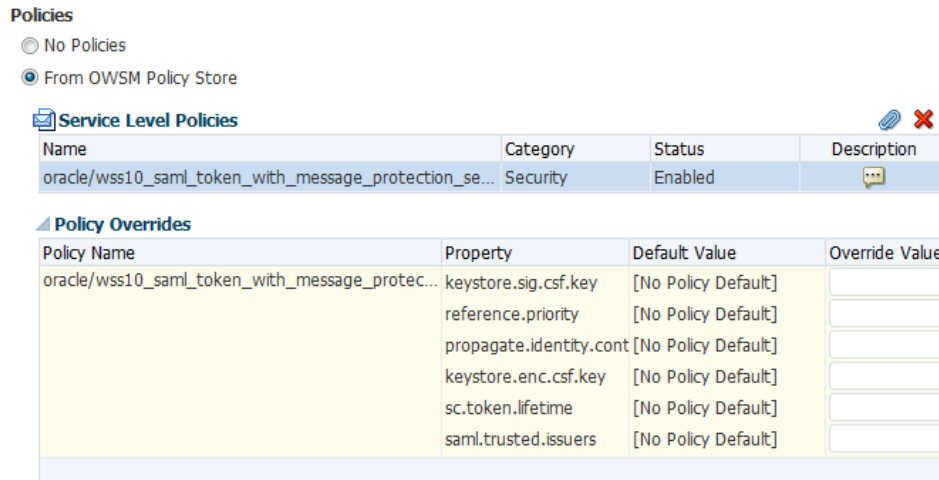
How to Define Override Values for a Policy in the Console

Your environment may include services that use the same policies. However, each service might have specific policy requirements, which you can specify using override properties. Not all policies allow override values.

To define override values for a policy in the console:

1. After you attach a policy, the policy appears in the Policy Overrides section if it allows you to specify override values. Locate the policy you want to configure in the Policy Overrides table.

Figure 49-6 Policy Overrides Table in the Oracle Service Bus Console



2. In the **Override Value** column, enter a value to override the default value listed in the **Override Value** column for each property you want to configure.
3. When you are done configuring override values, click **Save**.
4. To activate the changes in the runtime, click **Activate**.

How to Configure Custom Authentication for a Proxy Service in the Console

Custom authentication lets you specify custom user name and password combinations or custom tokens. You may need to specify the custom user name and password or token in XPath format. The format for both is similar in that you specify XPath expressions that enable Service Bus to locate the necessary information. The root of these XPath expressions is as follows:

- Use `soap-env:Envelope/soap-env:Header` if the service binding is AnySOAP or WSDL-SOAP.
- Use `soap-env:Body` if the service binding is not SOAP based.

All XPath expressions must be in a valid XPath format. The XPath expressions must use the XPath "declare namespace" syntax to declare any namespaces used, as follows:

```
declare namespace
ns='http://webservices.mycompany.com/MyExampleService';)
```

Note:

Not all fields and tasks described below are available for all service types. The configuration depends on the service type and policy configuration of the service.

You can also configure custom authentication for proxy and business services at the transport level. For more information, see [Configuring Custom Authentication Transport-Level Security](#).

Configuring Proxy Server Custom Authentication in the Console

To configure proxy server custom authentication in the console:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.
2. In the Project Navigator, locate the proxy service you want to edit and click the proxy service name.

The Proxy Service Definition page appears.

3. Click the **Security** tab, and then select **Security Settings**.
4. Do one of the following:
 - To specify the XPath to the user name and password, select **Custom User Name and Password**. Use the Expression Editor to define the XPath for the user name and password.
 - To specify a token, select **Custom Token**, select a token type, and then use the Expression Editor to define the XPath to the token.
5. Optionally, you can specify context properties to pass additional information to the context provider. For more information, see [Context Properties Are Passed to Security Providers](#). For more information about custom authentication, see [Configuring Custom Authentication](#).
6. When you are done configuring the security settings, click **Save**.
7. To activate the changes in the runtime, click **Activate**.

How to Specify a Service Key Provider for a Proxy Service in the Console

A service key provider contains Public Key Infrastructure (PKI) credentials that proxy services use for decrypting inbound SOAP messages and for outbound authentication and digital signatures. The service key provider resource used by the proxy service must be created before you can perform this step. For more information, see [Working with Service Key Providers](#).

To specify a service key provider for a proxy service in the console:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.
2. In the Project Navigator, locate the proxy service you want to edit and click the proxy service name.

The Proxy Service Definition page appears.

3. Click the **Security** tab, and then select **Security Settings**.
4. To specify a service key provider, click the **Browse** or **Search** icon to locate and select a service key provider to use.
5. When you are done configuring the security settings, click **Save**.
6. To activate the changes in the runtime, click **Activate**.

How to Specify Web Services Policy Enforcement in the Console

When a proxy service passes through the security header without processing it, it is known as a passive intermediary. For more information about web services security pass-through, see [What is Web Services Security Pass-Through?](#)

To specify web services policy enforcement in the console:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.
2. In the Project Navigator, locate the proxy service you want to edit and click the proxy service name.

The Proxy Service Definition page appears.

3. Click the **Security** tab, and then select **Security Settings**.
4. Do one of the following:
 - If the proxy service should not process the security header, select **Passive Security Intermediary**.
 - If the proxy service should process the security header, clear the **Passive Security Intermediary** check box.
5. When you are done configuring the security settings, click **Save**.
6. To activate the changes in the runtime, click **Activate**.

Configuring Service Bus Client Access Security

Client access to proxy services is defined directly in the service configuration in the Oracle Service Bus Console. When you create or manage a proxy service, you can view and update client access to the service from the Security Settings page on the Security tab. If both transport authentication and message-level authentication exist, the message-level subject identity is propagated.

How To Configure Transport-Level Access Policies

Configure transport-level security policies for a proxy service on the Security Settings tab of the Proxy Service Definition Editor in the Oracle Service Bus Console. This page provides access to the policy editor.

When a proxy service is activated, Service Bus generates and deploys a thin web application. Service Bus relies on WebLogic Server for server-side SSL support, including session management, client certificate validation and authentication, trust management and server SSL key/certificate manipulation.

For more information about defining transport-level security for various Service Bus transports, see [Configuring Transport-Level Security](#).

Before you can configure transport-level access policies, described in [Configuring Transport-Level Access Policies](#), you must enable HTTP URL links to open the policy editor, as described in [Enabling HTTP URL Links to Open the Policy Editor](#).

Enabling HTTP URL Links to Open the Policy Editor

To enable HTTP URL links to open the policy editor:

1. Log in to Fusion Middleware Control as a user with administrator privileges.
2. In the Target Navigator, expand **SOA** and click **service-bus**.
3. In the Service Bus menu, select **Security > Application Policies**.
4. In the **Application Stripe** field of the Application Policies page, select **Service_Bus_Console**.
The **Create** button is activated.
5. Click **Create** above the table.
6. In the Grantee section of the Create Application Grant page, click **Add**.
7. On the Add Principal dialog, do the following:
 - a. In the **Type** field, select **Application Role**.
 - b. Click **Search**.
 - c. Select the `MiddlewareAdministrator` role and click **OK**.
8. In the Permissions section of the Create Application Grant window, click **Add**.
9. Do the following on the Add Permission dialog:
 - a. To search by Java class, select **Permissions** and then select `oracle.soa.osb.console.common.permissions.OSBPermission` in the **Permission Class** field.
 - b. Click **Search**.
 - c. In the search results list, select `AdminOnlyTaskAccess` and click **Continue**.
 - d. In the **Permission Actions** field, select `update`. This also selects `All`.
 - e. Click **Select**.
The new permissions appears in the Permissions table.
10. When you are done granting permissions, click **OK** on the Create Application Grant window. After this is done you can complete the next task, configuring transport-level access policies.

Configuring Transport-Level Access Policies

To configure transport-level access policies:

1. Log in to the Oracle Service Bus Console as a user with administrator privileges. Only users with administrator privileges can modify security configuration data.
2. If you are not in an active session, click **Create** or **Edit** to start or restart a session.
3. In the Project Navigator, locate and open the proxy service whose transport-level access you want to configure.
4. Click the **Security** tab and then the **Security Settings** finger tab.
5. Click the link in the **Transport Access Control** field.

The policy editor appears.

6. In the **Authorization Providers** field, select an authorization provider. Oracle recommends that you select the XACMLAuthorizer.
7. Add policy conditions using any of the instructions in [How to Add Policy Conditions](#).
8. When you have finished entering conditions in the Policy Conditions section, click **Save**.

How to Configure Message-Level Access Policies

Configure message-level security policies for a proxy service on the Security Settings tab of the Proxy Service Definition Editor in the Oracle Service Bus Console. This page provides access to the policy editor. You can configure access policies at the operation level as well.

For more information about defining transport-level security, see [Configuring Message-Level Security for Web Services](#).

To configure message-level access policies:

1. Log in to the Oracle Service Bus Console as a user with administrator privileges. Only users with administrator privileges can modify security configuration data.
2. If you are not in an active session, click **Create** or **Edit** to start or restart a session.
3. In the Project Navigator, locate and open the proxy service whose transport-level access you want to configure.
4. Click the **Security** tab and then the **Security Settings** finger tab.
5. Click a link in the **Message Access Control** field.

Note:

You can define access control at the message or operation level, depending on which you select in this field.

The policy editor appears.

6. In the **Authorization Providers** field, select an authorization provider. Oracle recommends that you select the XACMLAuthorizer.

Note:

Service Bus has deprecated support for the WebLogic Default Authorization provider. Instead, Oracle recommends that you use the WebLogic XACML Authorization provider.

7. Add policy conditions using any of the instructions in [How to Add Policy Conditions](#).
8. When you have finished entering conditions in the Policy Conditions section, click **Save**.

How to Add Policy Conditions

You can define multiple conditions under which users, groups, or roles can invoke the secured operations. Conditions can be based on things like groups or roles, the date or time of access, context elements (for transport-level policies), and so on.

To add policy conditions:

1. Access the policy editor for an access control policy. See [How To Configure Transport-Level Access Policies](#) or [How to Configure Message-Level Access Policies](#).
2. In the policy editor, under Policy Conditions, click **Add Condition**.
The Choose a Predicate page appears.
3. Select a predicate from the list.
4. Click **Next**. Depending on what you chose as the condition predicate, perform one of the steps shown in [Table 49-2](#).

At any time you can click **Back** to discard your changes and return to the previous page or click **Cancel** to discard the changes and return to the Proxy Service Definition Editor.

Table 49-2 Condition Predicate Options

If You Selected...	Complete These Steps...
Role	<p>For transport-level security, this condition applies only if the proxy service uses a protocol that enables a client to supply credentials.</p> <ol style="list-style-type: none"> a. In the Role Argument Name field, enter the application role to which you want to grant access. b. Click Add. c. Repeat steps 1 and 2 until you have finished adding roles. You can click Remove to remove the arguments from the list. d. Click Finish.

Table 49-2 (Cont.) Condition Predicate Options

If You Selected...	Complete These Steps...
Group	<p>For transport-level security, this condition applies only if the proxy service uses a protocol that enables a client to supply credentials.</p> <ol style="list-style-type: none"> <li data-bbox="695 411 1377 642">a. In the Group Argument Name field, enter the group to which you want to grant access. If you have not already created the group that you entered in this field, you can do so after you finish creating access control policies. See "Creating Oracle Service Bus Groups" in <i>Administering Oracle Service Bus</i>. If you do not create this group, then no one will be granted access. <li data-bbox="695 663 867 695">b. Click Add. <li data-bbox="695 716 1377 810">c. Repeat steps 1 and 2 until you have finished adding arguments. You can click Remove to remove the arguments from the list. <li data-bbox="695 831 889 863">d. Click Finish.
User	<p>For transport-level security, this condition applies only if the proxy service uses a protocol that enables a client to supply credentials.</p> <ol style="list-style-type: none"> <li data-bbox="695 1031 1377 1262">a. In the User Argument Name field, enter the user to which you want to grant access. If you have not already created the user that you entered in this field, you can do so after you finish creating access control policies. See "Creating Oracle Service Bus Users" in <i>Administering Oracle Service Bus</i>. If you do not create this user, then no one will be granted access. <li data-bbox="695 1283 867 1314">b. Click Add. <li data-bbox="695 1335 1377 1430">c. Repeat steps 1 and 2 until you have finished adding arguments. You can click Remove to remove the arguments from the list. <li data-bbox="695 1451 889 1482">d. Click Finish.
Access occurs on specified days of the week	<ol style="list-style-type: none"> <li data-bbox="695 1556 1377 1619">a. In the Day of week field, enter the full name of the day of the week. <li data-bbox="695 1640 1377 1776">b. In the GMT offset field, enter the time ahead of GMT in the format GMT+h:mm, or behind GMT in the format GMT-h:mm. For example, Eastern Standard Time in the USA is GMT-5:00. <li data-bbox="695 1797 889 1829">c. Click Finish.

Table 49-2 (Cont.) Condition Predicate Options

If You Selected...	Complete These Steps...
Access occurs between specified hours	<ul style="list-style-type: none"> a. In the Starting Time field, enter the earliest permissible time in the format <code>hh:mm:ss AM PM</code>. For example, enter <code>12:45:00 AM</code>. b. In the Ending Time field, enter the latest permissible time in the format <code>hh:mm:ss AM PM</code>. c. In the GMT offset field, enter the time ahead of GMT in the format <code>GMT+h:mm</code>, or behind GMT in the format <code>GMT-h:mm</code>. For example, Eastern Standard Time in the USA is <code>GMT-5:00</code>. d. Click Finish.
Access occurs before or Access occurs after	<ul style="list-style-type: none"> a. In the Date field, enter a date in the format <code>m/d/yy</code>. For example, enter <code>1/1/04</code>. You can add an optional time in the format <code>h:mm:ss AM PM</code>. For example, you can enter <code>1/1/04 12:45:00 AM</code>. b. In the GMT offset field, enter the time ahead of GMT in the format <code>GMT+hh:mm</code>, or behind GMT in the format <code>GMT-hh:mm</code>. For example, Eastern Standard Time in the USA is <code>GMT-5:00</code>. c. Click Finish.
Access occurs on a specified day of the month, Access occurs before a specified day of the month, or Access occurs after a specified day of the month	<ul style="list-style-type: none"> a. In the The day of the month field, enter the ordinal number of the day within the current month with values in the range from -31 to 31. Negative values count back from the end of the month, so the last day of the month is specified as -1. 0 indicates the day before the first day of the month. b. In the GMT offset field, enter the time ahead of GMT in the format <code>GMT+hh:mm</code>, or behind GMT in the format <code>GMT-hh:mm</code>. For example, Eastern Standard Time in the USA is <code>GMT-5:00</code>. c. Click Finish.

Table 49-2 (Cont.) Condition Predicate Options

If You Selected...	Complete These Steps...
Context element defined	<p>Note: This applies only to transport-level security. A context element is a parameter and value pair that a container such as a web container can optionally provide to a security provider. Context elements are not available for message-level access control policies. For possible values, see Context Properties Are Passed to Security Providers.</p> <ol style="list-style-type: none"> In the Context element name field, enter the name of the context element. Click Finish.
Context element's value equals a string constant	<p>This applies only to transport-level security. See the note for Context element defined above for information about context elements.</p> <ol style="list-style-type: none"> In the Context element name field, enter the name of the context element for which to evaluate the value. In the String Value field, enter the string value that you want to compare. Click Finish.
Context element's value is greater than a numeric constant, Context element's value equals a numeric constant, or Context element's value is less than a numeric constant	<p>This applies only to transport-level security. See the note for Context element defined above for information about context elements.</p> <ol style="list-style-type: none"> In the Context element name field, enter the name of the context element for which to evaluate the value. In the Numeric Value field, enter a numeric value. Click Finish.
Deny access to everyone, Allow access to everyone or Server is in development mode	Click Finish .

- Repeat the above steps to add expressions based on different policy conditions. When you add multiple conditions, an operator list appears, and you can select to join the conditions by either AND or OR.
- Perform any of the following steps to modify the conditions you defined.
 - To change the order of the selected expression, select the check box associated with the condition, then click **Move Up** and **Move Down**.
 - To group policy conditions, select the check box associated with those conditions, and then click **Combine**. This allows you to create conditions such


```
as Role: Administrator OR (Role: Developer AND Access
occurs after: 12/1/13, GMT-5:00).
```

- To ungroup combined policy conditions, select the check box associated with those conditions, and then click **Uncombine**.
- To make a condition negative, select the check box associated with the condition, then click **Negate**. For example, NOT Group Operators excludes the Operators group from the policy.
- To delete a selected expression, select the check box associated with the condition, then click **Remove**.

Hiding Personally Identifiable Information in Messages

You can encrypt and decrypt fields of a message to protect sensitive data (known as personally identifiable information (PII)) in Service Bus pipelines. This feature provides for the obfuscation of certain fields (for example, SSNs) to prevent this data from appearing in administration consoles in clear text.

Messages are encrypted coming into Service Bus through a proxy service and then decrypted on the way out through a business service. Messages outside Service Bus can be protected with other message protection policies (WS-Security/SSL).

The following example shows an example of an unencrypted message. The PII fields are name and driversLicense.

Example - Unencrypted Message

```
<person>
  <name>John</name>
  <driversLicense>B1234</driversLicense>
  <ssn>123-456-789</ssn>
</person>
```

The following example shows an example of the encrypted message with the name and driversLicense fields in encrypted format.

Example - Encrypted Message

```
<person>
  <name>John</name>
  <driversLicense>encrypted:fdslj[lmsfwer09fsn;keyname=pii-csf-key</driversLicense>
  <ssn>encrypted:gdf45md%mfds103k;keyname=pii-csf-key</ssn>
</person>
```

The encryption format is as follows:

```
encrypted:<CIPHER_TEXT>;keyname:<CSF_KEY_NAME>
```

Note:

If both a PII policy and authorization policy are attached to a service, the authorization policy is executed before the PII policy. This is because the PII policy may encrypt the field used for authorization.

If the authorization policy is attached to a service and it requires an already-encrypted field, authorization fails.

How to Hide Personally Identifiable Information

- You must decrypt PII when an encrypted message leaves the service. If you attach a PII policy to a proxy service and do not attach a PII policy to its target service, PII in the outbound message are not decrypted. This is not a recommended practice.
- PII encrypted in one Service Bus service cannot be decrypted in another Service Bus service.

Hiding Personally Identifiable Information Using JDeveloper

To hide personally identifiable information using JDeveloper:

1. In the Application Navigator, locate the business or proxy service you want to edit and double-click the service's file.

The Business or Proxy Service Definition Editor appears.

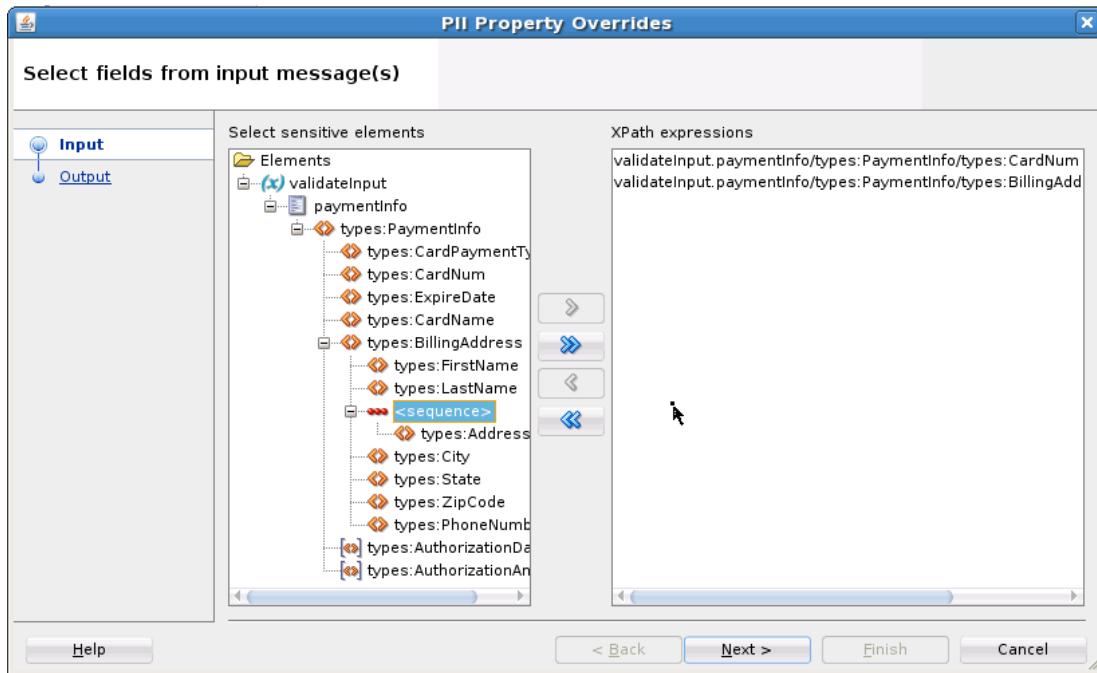
2. Click the **Policies** tab.
3. On the Policies page, select **From OWSM Policy Store** in the list of available policy binding models.
4. In the category of the policy you want to add, click **Add a * Policy**.

The policy is added.

5. Select the policy in the Personally Identifiable Information section, and click the **Edit** icon.

The PII Property Overrides dialog appears with the Select fields from input message(s) page displayed).

6. In the **Select sensitive elements** pane, expand the tree list, select a field whose value you want to hide, and then click the right arrow to move it to **XPath Expressions**.

Figure 49-7 PII Property Overrides Dialog

7. Repeat the above step for each field to encrypt.
8. Click **Next**.
The Select fields from output message(s) page appears.
9. Repeat steps 6 through 8 to select fields in the output message.
10. Select the CSF key. The key used for encryption and decryption is based on the password retrieved from this CSF key.

Hiding Personally Identifiable Information Using the Console

To hide personally identifiable information using the console:

1. If you have not already done so, click **Create** to create a new session or click **Edit** to enter an existing session.
2. In the Project Navigator, locate the business or proxy service and click the service name.
The Business or Proxy Service Definition Editor appears.
3. Click the **Security** tab, and then make sure **Policies** is selected.
4. On the Policies page, select **From OWSM Policy Store** in the list of available policy binding models.
5. In the Service Level Policies table, click **Attach Policies**.
The Security Policies dialog appears.
6. Do the following to select the policy:
 - a. Perform a search for the `oracle/pii_security_policy` policy, or look through the list for the policy.

Configuring Message-Level Security for Web Services

This chapter describes how to configure message-level security in Service Bus.

Message-level security applies security checks to a SOAP message after a web services client establishes a connection with a Service Bus proxy service or business service and before the proxy service or business service processes the message.

Message-level security is categorized as follows:

- **Inbound** message-level security applies to messages between clients and Service Bus proxy services. It applies security to both the request from the client and the response message back to the client.

You can think of this as proxy service security.

- **Outbound** message-level security applies to messages between Service Bus proxy services and SOAP-HTTP or SOAP-JMS business services. It applies security to both the request and the response.

You can think of this as business service security.

This chapter includes the following sections:

- [About Message-Level Security](#)
- [Message-Level Access Control Policies for Proxy Services](#)
- [Configuring Proxy Service Message-Level Security](#)
- [Configuring Business Service Message-Level Security: Main Steps](#)
- [Using the Service Identity Certificate Extensions](#)
- [Examples of Custom WS-Policy Statements](#)
- [Disabling Outbound WS-Security](#)

For instructions on configuring message-level security for proxy services, see [Configuring Service Bus Client Access Security](#).

Note:

The implementation of message-level security includes proxy services that have been configured with message-level custom authentication (either custom token or user name/password).

The message-level security mechanisms described in this section work alone or in concert with the message-level custom authentication mechanism, which is described in [Configuring Custom Authentication](#). See [Combining WS-Security with Custom User Name/Password and Tokens](#) for information about using both types of security.

About Message-Level Security

Service Bus supports message-level security for SOAP messages that are sent over the HTTP (including HTTPS) or JMS protocols. Usually you use message-level security in addition to the transport-level security that these protocols offer. You can require web services clients to provide credentials at the transport level, the message level, or both levels. If you require clients to provide credentials at both levels, Service Bus uses the message-level credentials for proxy service authentication and authorization.

To express the message-level security requirements for a proxy service or business service that is a web service, you use the Web Services Policy (WS-Policy) framework.

With message-level security, a proxy service or business service specifies which of its operations are secured and which of the following security measures a web services client must apply to its SOAP messages, which contain requests to invoke operations:

- **Authentication**
Requires a client to present an identity that can be compared with user accounts in the domain's authentication provider.
- **Message integrity through digital signatures**
Establishes the identity of the client that is requesting to invoke an operation and guarantees that no intermediary has altered the request. Also guarantees that the return values of the operation are returned to the client without being altered by an intermediary.
- **Message confidentiality through XML encryption**
Encrypts the request and the return value in the response and guarantees that no intermediary has viewed the request or the response.

All of these security measures require a client to encode security tokens in its SOAP messages, and the proxy service or business service specifies which types of security tokens it requires to be encoded in the SOAP messages.

Sample Sequence of Actions in Message-Level Security

To send a SOAP message to a proxy service that requires message-level security, the following actions occur:

1. A web services client generates a SOAP header and adds the header to the SOAP message envelope. The header includes digital signatures, security tokens, and other constructs.

2. When the proxy service processes the secured envelope, it decrypts the message, which removes the security header.
3. The proxy service then verifies that the message conforms to its security requirements. For example, the proxy service confirms that the required message parts were signed and/or encrypted and that the required tokens are present with the required claims.
4. The entire process is repeated in reverse for the response from the proxy service to the client.

Message-Level Access Control Policies for Proxy Services

While message integrity and message confidentiality guarantee that intermediaries do not view or modify messages, and while message authentication requires clients to prove that they are known users, they do nothing to specify which known users are allowed (authorized) to invoke proxy service operations.

To limit access to authorized users, you use the Oracle Service Bus Console to create message-level access control policies. These policies allow a proxy service to process only those SOAP messages from authorized clients.

Configuring Proxy Service Message-Level Security

You can configure a proxy service to support one of the following techniques for inbound message-level security:

- **Active-Intermediary**

The proxy service processes the header in the client's SOAP messages and enforces the message-level access control policy on the messages.

For example, a client encrypts and signs its SOAP message and sends it to a proxy service. The proxy service decrypts the message and verifies the digital signature, then routes the message. Before the proxy service sends the response back to the client, the proxy service signs and encrypts the message. The client then decrypts the message and verifies the proxy service's digital signature.

- **Pass-Through**

Instead of processing the header in the client's SOAP messages, the proxy service passes the message untouched to a business service. Although the proxy service does not process the secured sections of the SOAP message, it can route the message based on values in the header. When the business service receives the message, it processes the security header and acts on the request. Note that the business service must use the Web Services Policy (WS-Policy) framework to describe which of its operations are secured with message-level security. The business service sends its response to the proxy service, and the proxy service passes the response untouched to the client.

For example, the client encrypts and signs the message and sends it to the proxy service. The proxy service does not decrypt the message or verify the digital signature; it simply routes the message to the business service. The business service decrypts the messages and verifies the digital signature, and then processes the request. The response path is similar.

Creating an Active Intermediary Proxy Service: Main Steps

To create a proxy service to act as an active intermediary:

1. In a text editor or IDE, create a WSDL document to define the proxy service:
 - If you plan to bind the policies directly from the Oracle Service Bus Console, the WSDL file does not need to have policy statements.
 - If you want the policy to be WSDL-based, attach one or more Web Services Policy (WS-Policy) statements to the WSDL document, including one or more of the predefined policies.
2. In the Oracle Service Bus Console, import the WSDL document into the Service Bus WSDL repository and resolve any WSDL dependencies.

For more information about WSDL files, see [Working with WSDL Documents](#).
3. If you have not already configured the WebLogic security framework to support Service Bus, do one or more of the following depending on whether the WS-Policy of any of the operations in the proxy service contains security policy assertions that secure requests from clients to the proxy service:
 - If you want operation request policies to require authentication with a WS-Security X.509 certificate token, configure the Web Service Security configuration named `__SERVICE_BUS_INBOUND_WEB_SERVICE_SECURITY_MBEAN__`. See [Configuring the Oracle WebLogic Security Framework: Main Steps](#).
 - If you want operation request policies to require authentication with a WS-Security Username/Password token with password digest, make sure to enable password digests. See [Configuring the Oracle WebLogic Security Framework: Main Steps](#).
 - If you want operation request policies to require the use of SAML tokens, you must configure a SAML asserting party for this proxy service. See [Authenticating SAML Tokens in Proxy Service Requests](#).
 - If you want operation request policies to require digital signatures, register the accepted client signature verification certificates in the WebLogic Server Certificate Registry. See [Configuring the Oracle WebLogic Security Framework: Main Steps](#).
 - If you want operation request policies to require digital encryption, configure a service key provider that contains an encryption credential. The proxy service will use this credential to decrypt the encrypted SOAP message. For information about service key providers, see [Working with Service Key Providers](#).
4. In the Oracle Service Bus Console, do one or more of the following depending whether the WS-Policy of any of the operations in the proxy service contains security policy assertions that secure responses from the proxy service to clients:
 - If any operation response policy requires digital signatures, configure a service key provider that contains a digital signature credential. You can create one service key provider that contains credentials for both encryption and digital

signatures. For information about service key providers, see [Working with Service Key Providers](#)

- If any operation response policy specifies encryption, the client must send its certificate to the proxy service on the request. The proxy service will use the client's public key to encrypt its response. The client certificate must *not* be the same as the proxy service's encryption certificate.
5. In the Oracle Service Bus Console, create a proxy service from the WSDL file that you imported. Activate your changes.
 6. To attach OWSM policies to the proxy service, see one of the following sections:
 - [Attaching and Configuring Policies in JDeveloper](#)
 - [Attaching and Configuring Policies in the Oracle Service Bus Console](#)
 7. Edit the proxy service you just created to do the following from the Security tab:
 - a. Specify the service key provider that you created.
 - b. Optionally, modify the proxy service's default message-level access control policy, which specifies conditions under which users, groups, or roles can invoke the secured operations. For more information, see [How to Configure Message-Level Access Policies](#).
 - c. Optionally, modify the proxy service's message-level custom authentication settings.

Creating a Pass-Through Proxy Service: Main Steps

To create a pass-through proxy service:

1. Create a business service to which the proxy service will pass the unprocessed SOAP message. There are two configuration methods:
 - The business service is a web service that contains WS-Policy statements.
 - The business service directly binds the WS-Policies. The WSDL file on which the service is based should not have any WS-Policy statements.

See [Configuring Business Service Message-Level Security: Main Steps](#).

2. To attach OWSM policies to the proxy service, see one of the following sections:
 - [Attaching and Configuring Policies in JDeveloper](#)
 - [Attaching and Configuring Policies in the Oracle Service Bus Console](#)
3. In the Oracle Service Bus Console, create a proxy service from a WSDL document. You can use the same WSDL document that you used for the business service that you created. Activate your changes.
4. If you do not want the proxy service to enforce the security policies associated with it, select **Passive Security Intermediary** under Security Settings on the Security tab.
5. Configure the proxy service to route to the business service that you created.

If you route to the business service based on the operation that the client's SOAP message is requesting to invoke, you must configure the routing so that it specifies

an operation selection algorithm other than the SOAP body algorithm. Make sure the actions in the proxy service pipeline do not modify the WS-Security header or any parts of the SOAP envelope that are signed or encrypted. Changes to clear-text message parts covered by digital signatures almost always break the digital signature because the signature cannot be verified later.

Configuring Business Service Message-Level Security: Main Steps

Outbound message-level security applies to messages between Service Bus proxy services and SOAP-HTTP or SOAP-JMS business services. It applies security to both the request and the response.

To configure outbound message-level security for a business service that represents a SOAP-HTTP or SOAP-JMS web service:

1. In a text editor or IDE, create a WSDL document to define the policy.
2. In the Oracle Service Bus Console, import the web service's WSDL document into the Service Bus WSDL repository and resolve any WSDL dependencies.

For more information about WSDL files, see [Working with WSDL Documents](#).

3. In the Oracle Service Bus Console, do one or more of the following depending on whether the WSDL document contains WS-Policy statements that secure requests from a proxy service to the business service:
 - If any operation request policy includes an identity assertion with WS-Security Username Token as one of the supported token types, configure a service account for the business service. In the service account, provide the user name and password that you want the proxy service to send to the business service. Proxy services that route to this business service will get the user name and password from this service account. See [Working with Service Accounts and Creating and Configuring Business Services](#).
 - If any operation request policy requires authentication with a WS-Security Username/Password token with password digest, make sure to enable password digests. See [Configuring the Oracle WebLogic Security Framework: Main Steps](#).
 - If any operation request policy requires digital signatures, configure a service key provider that contains a digital signature credential. You can create one service key provider that contains credentials for both encryption and digital signatures. For information about service key providers, see [Working with Service Key Providers](#).
4. If any operation response policy in the business service requires encryption (that is, the business service encrypts the response with the proxy service's encryption public key), configure a service key provider and assign an encryption credential to the service key provider.

Caution:

Encrypted back-end response messages: If the response policy of the business service specifies encryption, the proxy service will send its encryption certificate to the business service on the request. The business service will encrypt its response using the proxy service's public key. The proxy service encryption credential must not be the same as the business service encryption credential.

5. If any policy in the business service specifies using SAML assertions, configure a WebLogic SAML Credential Mapping Provider V2 asserting party. For more information, see [Mapping Identity to a SAML Token](#).
6. Create a business service from the WSDL file that you imported. Activate your changes.
See [Creating and Configuring Business Services](#).
7. To attach OWSM policies to the business service, see one of the following sections:
 - [Attaching and Configuring Policies in JDeveloper](#)
 - [Attaching and Configuring Policies in the Oracle Service Bus Console](#)
8. Create a proxy service that routes SOAP messages to the business service. You can use either an active-intermediary proxy service or a pass-through proxy service.
See [Creating an Active Intermediary Proxy Service: Main Steps](#).

Using the Service Identity Certificate Extensions

Service Bus supports publishing and consuming certificate ID extensions for WSDL SOAP proxy and business services attached with OWSM message protection policies.

Publishing Certificate Identity Extension in a Proxy Service Effective WSDL

WSDL SOAP proxy services that implement an OWSM message-protection policy publish the base64-encoded public certificate for the service in the WSDL file. The certificate is included for message protection policies regardless of whether the policy encrypts or decrypts data.

The certificate is based on the encryption key configured for the attached policy. Proxy service clients can use the certificate embedded in the WSDL file for encryption purposes.

For more information, see "Using the Service Identity Certificate Extensions" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Note:

In prior releases of OWSM, business services and clients of proxy services needed to store the proxy service's public certificate in their domain-level keystore.

The client then used the `keystore.recipient.alias` property to identify the certificate in the keystore.

Consuming Certificate Identity Extension in a Business Service

WSDL SOAP business services that implement an OWSM message-protection policy consume the certificate ID extension from the WSDL file. If the public key certificate is not found in the WSDL file, then the `keystore.recipient.alias` property is used and the certificate must be in the business service's domain-level keystore.

The hostname verification feature ensures that a certificate retrieved from a WSDL file was not the subject of a substitution attack or "man in the middle" attack and is indeed the expected certificate.

To verify the hostname, OWSM validates that the common name (CN) or the subject Group Base Distinguished Name (DN) in the certificate matches the hostname of the service. This feature depends upon the subject DN of the certificate.

OWSM provides domain configuration properties that enable you to specify whether to enforce web service policies by publishing the X509 certificate in the WSDL file and whether to use the hostname verification feature. For details about setting these properties, see "Configuring Identity Extension Properties Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Examples of Custom WS-Policy Statements

The following sections provide examples of custom WS-Policy statements written under the WS-Policy specification using the proprietary Oracle schema for security policy:

- [Example: Encrypting Part of the SOAP Body and Header](#)
- [Example: Encryption Policy for a Business Service](#)
- [Example: Encrypting a Custom SOAP Header](#)
- [Example: Signing the Message Body and Headers](#)
- [Example: Signing a SOAP Body with SAML Holder-of-Key](#)
- [Example: Authenticating, Signing, and Encrypting with SAML Sender Vouches](#)

Example: Encrypting Part of the SOAP Body and Header

If you need to specify that particular parts of the body of a SOAP message are encrypted or digitally signed, rather than the entire body, you must create a custom WS-Policy file.

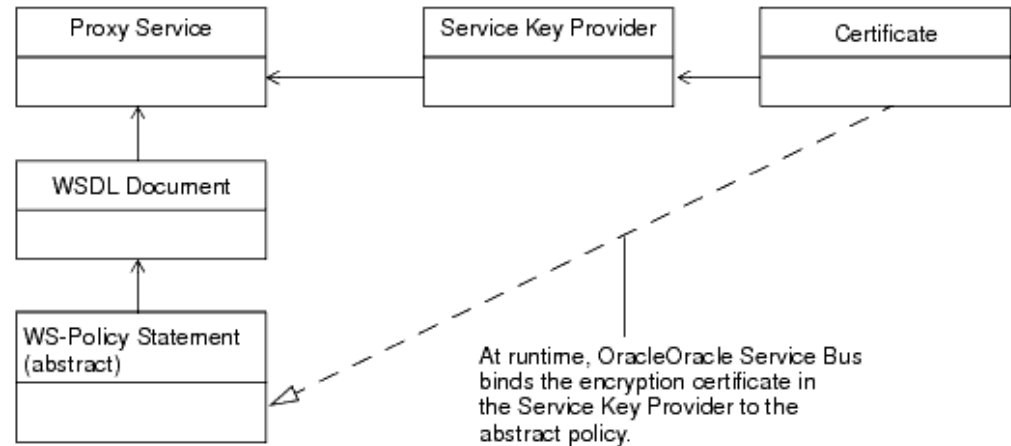
The example in this section is an abstract WS-Policy statement that does the following:

- Requires the message from the client to include a user name and password token for authentication
- Requires the client to encrypt the user name and password token (which is in the security header)
- Requires the client to encrypt the `/definitions/message/CreditCardNumber` element

This policy cannot be used with a business service because it is abstract: its `KeyInfo` element does not contain the certificate used for encryption. Instead, when you

activate a proxy service that uses this WS-Policy statement, Service Bus binds to the WS-Policy statement the encryption certificate from the service key provider that you associate with the proxy service. For more information, see [Working with Service Key Providers](#).

Figure 50-1 Binding a Certificate to an Abstract Policy



Also in the example:

- The `KeyWrappingAlgorithm` element specifies that the client must use the RSA 1.5 algorithm to wrap symmetric keys.
- The `EncryptionAlgorithm` specifies that the client must use the Triple DES (Data Encryption Standard) algorithm perform encrypt the security header and message body.

Example - Encrypting Part of the SOAP Body and Header

```

<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://www.bea.com/wls90/security/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd"
  xmlns:wssse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-secext-1.0.xsd"
  xmlns:m="http://example.org"
  wsu:Id="encrypt-custom-body-element-and-username-token">
  <!-- Require messages to provide a user name and password token
    for authentication -->
  <wssp:Identity>
    <wssp:SupportedTokens>
      <wssp:SecurityToken IncludeInMessage="true"
        TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
          wss-username-token-profile-1.0#UsernameToken">
        <wssp:UsePassword Type="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-username-token-profile-1.0#PasswordText"/>
      </wssp:SecurityToken>
    </wssp:SupportedTokens>
  </wssp:Identity>
  <wssp:Confidentiality>
    <wssp:KeyWrappingAlgorithm
      URI="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    <!-- Require the user name and password in the security header
      to be encrypted -->
  <wssp:Target>

```

```

<wssp:EncryptionAlgorithm
  URI="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
<wssp:MessageParts
  Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
  wls:SecurityHeader(wsse:UsernameToken)
</wssp:MessageParts>
</wssp:Target>
<!-- Require the /definitions/message/CreditCardNumber element to
  be encrypted -->
<wssp:Target>
  <wssp:EncryptionAlgorithm
    URI="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
  <wssp:MessageParts>
    wsp:GetBody(.) /m:CreditCardNumber
  </wssp:MessageParts>
</wssp:Target>
<!-- This is an abstract policy because the KeyInfo element is
  empty. The KeyInfo data is bound to the policy at runtime -->
<wssp:KeyInfo/>
</wssp:Confidentiality>
</wsp:Policy>

```

Example: Encryption Policy for a Business Service

Typically, you would require messages to a business service to be encrypted if the proxy service that sends messages to the business service is a pass-through proxy service. That is, the proxy service that receives messages from a client does not process the SOAP message. Instead, the proxy service routes the message to the business service, and the business service takes on the responsibility of Web Services Security. See [Configuring Message-Level Security for Web Services](#).

The example in this section is a WSDL document that contains a concrete policy. Note the following about this example:

- The policy requires clients to encrypt the message body.
- The `KeyInfo` element specifies the type of token that a client must provide to is the parent element that is used to describe and embed the encryption certificate. The `BinarySecurityToken` element contains the base-64 encoded encryption certificate (the value is truncated in the example). If your certificate is in PEM format, the content of the PEM file (without the PEM prefix and suffix) is the base-64 encoded representation of the certificate. If your encryption certificate is stored in a JDK keystore, you can easily export it to a PEM file.
- The policy provides a unique ID and the WSDL file uses a URI fragment to refer to the ID.

Example - Encrypting the Body with a Concrete Policy Embedded in the WSDL Document

```

<definitions name="WssServiceDefinitions"
  targetNamespace="http://com.bea.alsb/tests/wss"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  ...>
<wsp:UsingPolicy xmlns:n1="http://schemas.xmlsoap.org/wsdl/"
  n1:Required="true"/>
<!-- The policy provides a unique ID -->

```

```

<wsp:Policy wsu:Id="myEncrypt.xml">
  <wssp:Confidentiality
    xmlns:wssp="http://www.bea.com/wls90/security/policy">
    <wssp:KeyWrappingAlgorithm
      URI="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    <!-- Require the user name and password in the security header
      to be encrypted -->
    <wssp:Target>
      <wssp:EncryptionAlgorithm
        URI="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc"/>
      <wssp:MessageParts Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
        wsp:Body()
      </wssp:MessageParts>
    </wssp:Target>
    <!-- Embed the token type and encryption certificate -->
    <wssp:KeyInfo>
      <wssp:SecurityToken
        TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
          wss-x509-token-profile-1.0#X509v3"/>
      <wssp:SecurityTokenReference>
        <wssp:Embedded>
          <wsse:BinarySecurityToken
            EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-
              200401-wss-soap-message-security-1.0#Base64Binary"
            ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
              200401-wss-x509-token-profile-1.0#X509v3"
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
              200401-wss-wssecurity-secext-1.0.xsd">
              MIICfjCCAeegAwIBAgIQV/PDyJ3...
            </wsse:BinarySecurityToken>
          </wssp:Embedded>
        </wssp:SecurityTokenReference>
      </wssp:KeyInfo>
    </wssp:Confidentiality>
  </wsp:Policy>
<binding name="WssServiceSoapBinding" type="tns:WssService">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getPurchaseOrder">
    <soap:operation soapAction="" style="document"/>
    <input>
      <soap:body parts="parameters" use="literal"/>
      <!-- Use a URI fragment to refer to the unique policy ID -->
      <wsp:Policy>
        <wsp:PolicyReference URI="#myEncrypt.xml"/>
      </wsp:Policy>
    </input>
    <output>
      <soap:body parts="parameters" use="literal"/>
    </output>
  </operation>
</binding>
...
</definitions>

```

Example: Encrypting a Custom SOAP Header

The example in this section is an abstract WS-Policy statement that encrypts a custom header named `CreditCardNumber`.

If you need to specify that particular parts of the body of a SOAP message are encrypted or digitally signed, rather than the entire body, you must create a custom WS-Policy file.

This policy cannot be used with a business service because it is abstract: its `KeyInfo` element does not contain the certificate used for encryption. Instead, when you activate a proxy service that uses this WS-Policy statement, Service Bus binds to the WS-Policy statement the encryption certificate from the service key provider that you associate with the proxy service. For more information, see [Working with Service Key Providers](#).

Also of note in this example:

- The `KeyWrappingAlgorithm` element specifies that the client must use the RSA 1.5 algorithm to wrap symmetric keys.
- The `EncryptionAlgorithm` specifies that the client must use the Triple DES (Data Encryption Standard) algorithm perform encrypt the security header.

Example - Encrypting a Custom SOAP Header

```
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://www.bea.com/wls90/security/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd"
  wsu:Id="dig-sig-for-get-header">
  <wssp:Confidentiality>
    <wssp:KeyWrappingAlgorithm
      URI="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    <!-- Require the custom CreditCardNumber header to be encrypted -->
    <wssp:Target>
      <wssp:EncryptionAlgorithm
        URI="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
      <wssp:MessageParts
        Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">
        wsp:GetHeader(/n:CreditCardNumber
      </wssp:MessageParts>
    </wssp:Target>
    <wssp:KeyInfo/>
  </wssp:Confidentiality>
</wsp:Policy>
```

Example: Signing the Message Body and Headers

The example in this section is a WS-Policy statement that requires a digital signature to access the following in the SOAP message:

- A custom header named `header1`
- All system headers
- The timestamp security header
- The message body

Example - Requiring a Signature for SOAP Headers and Body

```
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://www.bea.com/wls90/security/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
```



```

wssecurity-utility-1.0.xsd"
wsu:Id="sign-custom-header-policy">
<wssp:Integrity>
  <wssp:SignatureAlgorithm
    URI="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <wssp:CanonicalizationAlgorithm
    URI="http://www.w3.org/2001/10/xml-exc-c14n#"/>
  <!-- Require the custom header header1 to be signed -->
  <wssp:Target>
    <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <wssp:MessageParts
      Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
        wssecurity-secext-1.0.xsd"
      xmlns:n="http://example.org">
      wsp:GetHeader(/n:header1
    </wssp:MessageParts>
  </wssp:Target>
  <!-- Require the system headers to be signed -->
  <wssp:Target>
    <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <wssp:MessageParts
      Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
      wls:SystemHeaders()
    </wssp:MessageParts>
  </wssp:Target>
  <!-- Require the Timestamp header to be signed -->
  <wssp:Target>
    <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <wssp:MessageParts
      Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
      wls:SecurityHeader(wsu:Timestamp)
    </wssp:MessageParts>
  </wssp:Target>
  <!-- Require the message body to be signed -->
  <wssp:Target>
    <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <wssp:MessageParts
      Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
      wsp:Body()
    </wssp:MessageParts>
  </wssp:Target>
</wssp:Integrity>
<wssp:MessageAge/>
</wsp:Policy>

```

Example: Signing a SOAP Body with SAML Holder-of-Key

The example in this section is a WS-Policy statement that requires the SAML asserter to use the holder-of-key method to sign the message body. The purpose of a SAML token with "holder-of-key" subject confirmation is to allow the subject to use an X.509 certificate that may not be trusted by the receiver to protect the integrity of the request messages.

For more information about the two SAML confirmation methods (sender-vouches or holder-of-key), see "SAML Token Profile Support in WebLogic Web Services" in *Understanding Security for Oracle WebLogic Server*.

The "Oracle WebLogic Server Security Policy Assertion Reference" in the *WebLogic Web Services Reference for Oracle WebLogic Server* describes the policy elements in detail.

Note the following about this example:

- `Integrity` specifies that part or all of the SOAP message must be digitally signed, as well as the algorithms and keys that are used to sign the SOAP message.
- `SignatureAlgorithm` specifies the cryptographic algorithm used to compute the digital signature.
- `CanonicalizationAlgorithm` specifies the algorithm used to canonicalize (use in simple or standard form) the SOAP message elements that are digitally signed. You can specify only `http://www.w3.org/2001/10/xml-exc-c14n#`.
- `DigestAlgorithm` specifies the digest algorithm that is used when digitally signing the specified parts of a SOAP message. You can specify only `http://www.w3.org/2000/09/xmlsig#sha1`.
- `MessageParts` specifies the parts of the SOAP message that should be signed, in this case the body.
- `Dialect` identifies the dialect used to identify the parts of the SOAP message that should be signed.
- `SupportedTokens` specifies the list of supported security tokens that can be used for digital signatures.
- `SecurityToken` specifies the security token that is supported for digital signatures.

`IncludeInMessage` specifies whether to include the token in the SOAP message. Valid values are true or false. The default value of this attribute is true when used in the `<Integrity>` assertion.

`TokenType` specifies the type of security token, in this case to specify a SAML token.

- `Claims` specifies additional metadata information that is associated with a particular type of security token. For SAML tokens, you must define a `<ConfirmationMethod>` child element to specify the type of SAML confirmation (sender-vouches or holder-of-key).
- `ConfirmationMethod` specifies the type of confirmation method, either sender-vouches or holder-of-key, that is used when using SAML tokens for identity.

Specify the `<ConfirmationMethod>` assertion within an `<Integrity>` assertion. The reason you put the SAML token in the `<Integrity>` assertion for this confirmation method is that the web service runtime must prove the integrity of the message, which is not required by sender-vouches.

Example - Signing a SOAP Body with SAML Holder-of-Key Method

```
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://www.bea.com/wls90/security/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd"
  xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part"
  wsu:Id="saml-holder-of-key-signed">
  <wssp:Integrity>
    <wssp:SignatureAlgorithm
      URI="http://www.w3.org/2000/09/xmlsig#rsa-sha1"/>
    <wssp:CanonicalizationAlgorithm
```

```

URI="http://www.w3.org/2001/10/xml-exc-c14n#" />
<wssp:Target>
  <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/xmldsig#sha1" />
  <wssp:MessageParts
    Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
    wsp:Body()
  </wssp:MessageParts>
</wssp:Target>
<wssp:SupportedTokens>
  <wssp:SecurityToken IncludeInMessage="true"
    TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-saml-
    token-profile-1.0#SAMLAssertionID">
    <wssp:Claims>
      <wssp:ConfirmationMethod>holder-of-key</wssp:ConfirmationMethod>
    </wssp:Claims>
  </wssp:SecurityToken>
</wssp:SupportedTokens>
</wssp:Integrity>
</wsp:Policy>

```

Example: Authenticating, Signing, and Encrypting with SAML Sender Vouches

The example in this section is a WS-Policy statement that requires the SAML asserter to use the sender-vouches method to sign the message body and headers.

In sender-vouches the asserting party (different from the subject) vouches for the verification of the subject. The receiver must have a trust relationship with the asserting party.

For more information about the two SAML confirmation methods (sender-vouches or holder-of-key), see "SAML Token Profile Support in WebLogic Web Services" in *Understanding Security for Oracle WebLogic Server*.

The "Oracle Web Services Security Policy Assertion Reference" in the *WebLogic Web Services Reference for Oracle WebLogic Server* describes the policy elements in detail.

Note the following about this example:

- `Identity` specifies the type of security tokens.
- `SupportedTokens` specifies the list of supported security tokens that can be used for digital signatures.
- `SecurityToken` specifies the security token that is supported for digital signatures.

`IncludeInMessage` is not specified because the value of this attribute is always true when used in the `<Identity>` assertion, even if you explicitly set it to false.

`TokenType` specifies the type of security token to specify a SAML token.
- `Claims` specifies additional metadata information that is associated with a particular type of security token. For SAML tokens, you must define a `<ConfirmationMethod>` child element to specify the type of SAML confirmation (sender-vouches or holder-of-key).
- `ConfirmationMethod` specifies the type of confirmation method, either sender-vouches or holder-of-key, that is used when using SAML tokens for identity.

- Integrity specifies that part or all of the SOAP message must be digitally signed (in this example both the body and security headers), as well as the algorithms and keys that are used to sign the SOAP message.
- SignatureAlgorithm specifies the cryptographic algorithm used to compute the digital signature.
- CanonicalizationAlgorithm specifies the algorithm used to canonicalize (use in simple or standard form) the SOAP message elements that are digitally signed. You can specify only `http://www.w3.org/2001/10/xml-exc-c14n#`.
- Target encapsulates information about which targets of a SOAP message are to be encrypted or signed, depending on the parent element. The child elements also depend on the parent element:
 - When used in `<Integrity>`, you can specify the `<DigestAlgorithm>`, `<Transform>`, and `<MessageParts>` child elements.
 - When used in `<Confidentiality>`, you can specify the `<EncryptionAlgorithm>`, `<Transform>`, and `<MessageParts>` child elements.
- DigestAlgorithm specifies the digest algorithm that is used when digitally signing the specified parts of a SOAP message. You can specify only `http://www.w3.org/2000/09/xmlsig#sha1`.
- MessageParts specifies the parts of the SOAP message that should be signed, in this case the body and security header.
- Dialect identifies the dialect used to identify the parts of the SOAP message that should be signed.
- Confidentiality specifies that part or all of the SOAP message must be encrypted, as well as the algorithms and keys that are used to encrypt the SOAP message. The example requires that the body and security headers must be encrypted using triple-DES.

Example - Signing a SOAP Body and Headers with SAML Sender-Vouches Method

```
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://www.bea.com/wls90/security/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd"
  xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part"
  wsu:Id="samlPolicy-sender-vouches-signed-encrypted">
  <wssp:Identity>
    <wssp:SupportedTokens>
      <wssp:SecurityToken
        TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-
          saml-token-profile-1.0#SAMLAssertionID">
        <wssp:Claims>
          <wssp:ConfirmationMethod>
            sender-vouches
          </wssp:ConfirmationMethod>
        </wssp:Claims>
      </wssp:SecurityToken>
    </wssp:SupportedTokens>
  </wssp:Identity>
```

```

<wssp:Integrity>
  <wssp:SignatureAlgorithm
    URI="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <wssp:CanonicalizationAlgorithm
    URI="http://www.w3.org/2001/10/xml-exc-c14n#" />
  <wssp:Target>
    <wssp:DigestAlgorithm
      URI="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <wssp:MessageParts
      Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
        wsp:Body()
      </wssp:MessageParts>
    </wssp:Target>
  </wssp:Target>
  <wssp:Target>
    <wssp:DigestAlgorithm
      URI="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <wssp:MessageParts
      Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
        wls:SecurityHeader(Assertion)
      </wssp:MessageParts>
    </wssp:Target>
  </wssp:Integrity>
<wssp:Confidentiality>
  <wssp:KeyWrappingAlgorithm
    URI="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
  <wssp:Target>
    <wssp:EncryptionAlgorithm
      URI="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc" />
    <wssp:MessageParts
      Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
        wls:SecurityHeader(Assertion)
      </wssp:MessageParts>
    </wssp:Target>
  </wssp:Target>
  <wssp:Target>
    <wssp:EncryptionAlgorithm
      URI="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc" />
    <wssp:MessageParts
      Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
        wss:Body()
      </wssp:MessageParts>
    </wssp:Target>
  <wssp:KeyInfo/>
</wssp:Confidentiality>
</wsp:Policy>

```

Disabling Outbound WS-Security

Note:

On proxy services that forward to other proxies (such as local proxies) containing Oracle Web Services Manager service policies, outbound WS-Security processing is disabled. Service Bus handles that behavior automatically and does not use the `doOutboundWss` property. For more information, see [Using OWSM Security with Local Proxy Services](#).

The remainder of this section describes how to disable outbound WS-Security processing for other design patterns.

Some infrequently used design patterns preempt a proxy service from automatically generating the outbound WS-Security SOAP envelope and instead use an XQuery expression to create the envelope. If you use this design pattern, to prevent a proxy service from automatically generating the outbound WS-Security SOAP envelope, you must create an action in the proxy service's message flow that sets the value of the `./ctx:security/ctx:doOutboundWss` element in the `$outbound` message context variable to `xs:boolean("false")`. You can create the action in either of the following places:

- In a request stage of a pipeline pair. See [How to Add Pipeline Pairs to Pipelines](#).
- In a request action of a route node. See [How to Add Route Nodes to Pipelines in the Console](#).

For information about the `$outbound` message context variable, see [Message Context](#).

Under some circumstances, when you attempt to activate a session in which you have created or modified a proxy service with outbound message-level security disabled, the Oracle Service Bus Console reports validation errors (you cannot commit a session that contains errors). If your session validation reports errors because you have disabled outbound message-level security, modify the Service Bus startup command so that it sets the following system property to `true`:

```
com.bea.wli.sb.security.wss.LaxOutboundWssValidation
```

Then restart Service Bus. With this property set to `true`, the Oracle Service Bus Console reports warnings instead of errors (you can commit a session that reports warning messages).

Future releases of Service Bus will provide an easier way to disable outbound message-level security.

Configuring Transport-Level Security

This chapter describes how to configure transport-level security for different transports in Service Bus.

Transport-level security applies security checks as part of establishing a connection between service consumers, proxy services, and business services. The type of security checks that Service Bus can apply depends on the protocol that the proxy service or business service uses to communicate. Some protocols can also encrypt the communication between client and endpoint to prevent snooping from third parties.

Inbound transport-level security secures the communication between clients and Service Bus proxy services. **Outbound** transport security secures all three techniques of sending outbound requests from Service Bus proxy services: route actions, publish actions, and callout actions.

This chapter includes the following sections:

- [Configuring Transport-Level Security for HTTPS](#)
- [Configuring Transport-Level Security for HTTP](#)
- [Configuring Transport-Level Security for JMS](#)
- [Configuring Transport-Level Security for SFTP Transport](#)
- [Email, FTP, and File Transport-Level Security](#)
- [Configuring Transport-Level Security for SB Transport](#)
- [Configuring Transport-Level Security for WS Transport](#)
- [Configuring Transport-Level Security for WebSphere Message Queue Transport](#)
- [Transport-Level Security Elements in the Message Context](#)

For instructions on configuring security in business and proxy services, see [Securing Business and Proxy Services](#).

Configuring Transport-Level Security for HTTPS

The HTTPS protocol uses SSL to secure communication. SSL can be used to encrypt communication, ensure message integrity, and to require strong server and client authentication. Before you can use HTTPS, you must configure SSL in WebLogic Server, see [Configuring the Oracle WebLogic Security Framework: Main Steps](#).

The following sections describe configuring transport-level security for the HTTPS protocol:

- [HTTPS Authentication Levels](#)

- [Configuring Inbound HTTPS Security: Main Steps](#)
- [Configuring Outbound HTTPS Security: Main Steps](#)

HTTPS Authentication Levels

For each proxy service or business service that communicates over the HTTPS protocol, you can configure the service to require one of the following levels of authentication:

- **One-way SSL, no authentication**

This level enables encrypted communication but does not require clients to provide credentials. To establish a one-way SSL connection, the client initiates the connection and Service Bus sends its certificate to the client. In other words, the client authenticates Service Bus.
- **One-way SSL, basic authentication**

This level enables encrypted communication and requires clients to supply a user name and password after the one-way SSL connection is established. The client supplies a user name and password by encoding it in the HTTP request header (which is encrypted by SSL). When the proxy service receives the encrypted request, it passes the credentials to the domain's authentication provider, which determines whether the client's credentials match a user account that you have created.
- **Two-way SSL, client certificate authentication**

This level enables encrypted communication and strong client authentication (two-way SSL).

To establish a two-way SSL connection, the client initiates the connection and Service Bus sends its X.509 certificate to the client. Then, the client sends its certificate to Service Bus and Service Bus authenticates the client.

To get the user name from the client's certificate, you configure an Identity Assertion provider, which extracts a field in the certificate to use as the client identity (X.509 token), typically the CN (common name) or E (email) of the `SubjectDistinguishedName` in the certificate. After extracting the X.509 token, the token is compared to the user accounts created in Fusion Middleware Control.

For more information about SSL and Identity Assertion providers, see "Security Fundamentals" in *Understanding Security for Oracle WebLogic Server*.
- **Transport-Level Custom Credentials.**

You can authenticate client requests at the transport-level using custom authentication tokens. You specify a custom token in an HTTP header. The HTTP-specific configuration pages of the service definition wizard allows you to configure client authentication. Custom authentication concepts are described in [Configuring Custom Authentication](#).

Configuring Inbound HTTPS Security: Main Steps

To configure inbound transport-level security for a proxy service:

1. Make sure that you have configured the WebLogic security framework to support SSL, an authentication provider, and an Identity Assertion provider, depending on the HTTPS authentication level that you want to use:

- For no client authentication (anonymous requests), set Client Authentication to None.
- For basic authentication, set client authentication to basic. See "Creating Oracle Service Bus Users" in *Administering Oracle Service Bus*.
- For SSL client authentication, set client authentication to client certificate, configure the WebLogic Identity Assertion provider and the WebLogic CertPath Provider.
- For custom authentication token, set client authentication to custom authentication. The custom authentication token can be any active token type previously configured for an Identity Assertion provider that is carried in an HTTPS header. Custom authentication concepts are described in [Configuring Custom Authentication](#).

Note:

You must first configure, or create and configure, a WebLogic Server Identity Assertion provider as described in [Configuring Identity Assertion Providers for Custom Tokens](#), and add the user names and passwords of the clients that you want to allow access to Fusion Middleware Control.

See [Configuring the Oracle WebLogic Security Framework: Main Steps](#).

2. When you create a proxy service, on the Transport Configuration page select HTTP.
3. On the HTTP Transport Configuration page, select the "HTTPS required" option.
4. Choose an authentication level, as described in [HTTPS Authentication Levels](#).
5. Make your Dispatch Policy, Request Encoding, and Response Encoding choices, as described in the online help provided with Service Bus.
6. If the service you are creating is WSDL-based and has operations, make your selections on the Message Handling Configuration page. Determine whether to enforce WS-I compliance (for SOAP 1.1 services only) and select the selection algorithm to use to determine the operation called by this proxy service. This option is available only for SOAP or XML services defined from a WSDL file.

Configuring Outbound HTTPS Security: Main Steps

In outbound transport-level security, a proxy service is the client that opens a connection with a business service.

To configure outbound transport-level security:

1. If you are configuring transport-level security for a production environment (as opposed to a development or testing environment), make sure that Host Name Verification is enabled. See "Using Host Name Verification" in "Configuring SSL" in *Administering Security for Oracle WebLogic Server*.
2. When you create a business service, on the Transport Configuration page, select HTTP.
3. Choose an authentication level, as described in [HTTPS Authentication Levels](#).

If you configured the proxy service such that Service Bus does not authenticate clients, configure the enterprise system to authenticate clients by selecting an authentication level of one-way SSL, which is the basic authentication option.

4. The URI determines whether HTTP or HTTPS is used. HTTP business services can combine HTTP and HTTPS URLs unless the authentication method is client certificate, in which case all URLs must be HTTPS.
5. If the business service uses HTTPS with basic authentication, create a service account to provide the user name and password that the business service requires.
You can add a user name and password directly to the service account, or configure the service account to pass through the credentials that it received from its client's request, or you can map a client user name to a Service Bus user. If you configured the service so that Service Bus does not authenticate clients, create a service account that passes through the credentials.
6. If the business service uses client certificate authentication, do the following:
 - a. Create a service key provider to provide the key-pair that proxy services use for SSL client authentication with the business service. For information about service key providers, see [Working with Service Key Providers](#).
 - b. Create a proxy service or edit an existing proxy service so that it specifies the service key provider.
7. If the business service uses custom authentication, specify a custom Java class for authentication.

Configuring Transport-Level Security for HTTP

The HTTP protocol does not encrypt communication between clients and proxy services or business services, but it does support basic authentication in which clients send user names and passwords in requests. HTTP also supports custom token authentication.

Caution:

Unless you have configured strong network security, Oracle recommends that you do not use basic authentication with HTTP in production environments because the password is sent in clear text. Instead, use basic authentication with HTTPS.

The following sections describe configuring transport-level security for the HTTP protocol:

- [Configuring Inbound HTTP Security: Main Steps](#)
- [Configuring Outbound HTTP Security: Main Steps](#)
- [Using Custom Authentication for Outbound HTTP Security](#)

Configuring Inbound HTTP Security: Main Steps

To configure inbound transport-level security for a proxy service:

1. When you create a proxy service, on the Transport Configuration page select HTTP. Choose the Client Authentication option None, Basic, or Custom Authentication. If you choose Custom Authentication, you must also specify the HTTP header that is to carry the token and the token type.

The steps for configuring transport-level custom credentials are described in [Modeling Message Flow in Oracle Service Bus](#). Custom authentication concepts are described in [Configuring Custom Authentication](#).

The custom authentication token can be any active token type, previously configured for an Identity Assertion provider, that is carried in an HTTP header.

Note:

To use custom authentication you must first configure, or create and configure, a WebLogic Server Identity Assertion provider as described in [Configuring Identity Assertion Providers for Custom Tokens](#).

If you want Service Bus to authenticate clients (Basic or Custom Authentication) you must create user accounts for the clients. See "Configuring Oracle Service Bus Administrative Security" in *Administering Oracle Service Bus*.

2. Modify the proxy service's default transport-level access control policy, which specifies conditions under which users, groups, or roles can access a proxy service. See [How To Configure Transport-Level Access Policies](#).

Configuring Outbound HTTP Security: Main Steps

In outbound transport-level security, a proxy service is the client that opens a connection with a business service.

To configure outbound transport-level security:

1. When you create a business service, select HTTP on the Transport Configuration page. Choose the Client Authentication option None, Basic, or Custom Authentication. If you choose Custom Authentication, you must also specify the HTTP header that is to carry the token and the token type.

Custom authentication concepts are described in [Configuring Custom Authentication](#).

2. If you selected Basic security, create a service account to provide the user name and password that the business service requires. This is optional for Custom Authentication. See [Working with Service Accounts](#).

The custom authentication token can be any active token type, previously configured for an Identity Assertion provider, that is carried in an HTTP header.

Note:

To use custom authentication for outbound security, you need to implement the `doOutboundAuthentication` method of the `OutboundAuthentication` interface. See [Using Custom Authentication for Outbound HTTP Security](#) for more information.

If you want Service Bus to authenticate clients (Basic or Custom Authentication) you must create user accounts for the clients. See "Configuring Oracle Service Bus Administrative Security" in *Administering Oracle Service Bus*.

You can add a user name and password directly to the service account, or configure the service account to pass through the credentials that it received from its client's request, or you can map a client user name to a Service Bus user. If you configured the service so that Service Bus does not authenticate clients, create a service account that passes through the credentials.

3. Create a proxy service or edit an existing proxy service so it specifies the service account.

Using Custom Authentication for Outbound HTTP Security

To use custom authentication on the outbound, you need to implement the `doOutboundAuthentication` method of the `com.bea.wli.sb.transports.http.OutboundAuthentication` interface. Oracle Service Bus initializes the `HttpURLConnectionFactory` with the connection parameters before invoking the `OutboundAuthentication` implementation.

The custom authentication code can use the factory method `HttpURLConnectionFactory.newConnection()` for retrieving a new instance of `HttpURLConnection`. The `HttpURLConnection` instance returned from the factory method will be configured with the parameters set by Service Bus runtime.

In the final step of the authentication process, the business service payload and user headers need to be sent to the target service, along with the authentication parameters. Service Bus handles this automatically. So, the custom authenticator has to ensure that only the authentication parameters are set for the last `Http` connection instance created in the custom authentication code. The following `HttpURLConnection` methods which establish the connection to the target service must not be invoked for the last instance of `HttpURLConnection` created in the custom authentication code:

- `connect`
- `getResponseCode`
- `getResponseMessage`
- `getHeader` methods
- `getContent`
- `getInputStream`
- `getOutputStream`

Configuring Transport-Level Security for JMS

While transport-level security for JMS does not provide end-to-end security for JMS messaging, it does provide the following:

- The option to use a secure SSL channel for communication between Service Bus and a JMS server for sending or receiving JMS messages.

Note:

The JMS transport does not support two-way SSL.

Service Bus can communicate with local JMS servers or foreign JMS servers. The connection to JMS servers can be secured using the T3S protocol (T3 over SSL). T3 and T3S are proprietary Oracle protocols.

- The ability to specify the user name and password that Service Bus proxy services use to authenticate while establishing a connection to a JMS server and/or while looking up JMS destinations in the JNDI tree.

Note:

JMS administrators use the Oracle WebLogic Server Administration Console to create access control policies that restrict access to WebLogic JMS servers and destinations in the JNDI tree. For more information, see *Resource Types You Can Secure with Policies in Securing Resources Using Roles and Policies for Oracle WebLogic Server* and *Methods for Configuring JMS System Resources in Administering JMS Resources for Oracle WebLogic Server*.

If a JMS administrator configures or changes an access control policy for a JMS destination, WebLogic Server can take up to 60 seconds to recognize the changes.

By default, WebLogic Server JMS checks the policy for each JMS destination every 60 seconds. To change this behavior, modify the WebLogic Server startup command so that it sets the following system property to the frequency (in seconds) that you want WebLogic Server JMS to check access control policies: `weblogic.jms.securityCheckInterval`

A value of 0 (zero) for this property ensures that an authorization check is performed for every `send`, `receive`, and `getEnumeration` action on a JMS resource.

The following sections describe configuring JMS transport-level security:

- [Configuring Inbound JMS Transport-Level Security: Main Steps](#)
- [Configuring Outbound JMS Transport-Level Security: Main Steps](#)

Configuring Inbound JMS Transport-Level Security: Main Steps

To configure inbound JMS transport-level security:

1. When you create or edit a JMS proxy service, on the Transport Configuration page, under Advanced Settings, select the **Use SSL** check box.

Service Bus configures the JMS proxy service to use the T3S protocol.

2. If the JMS administrator created access control policies that restrict access to a JMS connection pool, configure the proxy service to authenticate when it connects to the JMS server:

- a. Create a service account to provide the user name and password that the JMS server requires. See [Working with Service Accounts](#).

The JMS service account for the proxy service is used not only for the JMS object access, but also for the JNDI lookup.

You must add a user name and password directly in the service account. JMS cannot use a service account that passes through the credentials that it received from its client's request or that maps a client user name to a Service Bus user.

- b. When you create or edit the proxy service, on the JMS Transport page, click the **Browse** button next to JMS Service Account. Select the service account that you created in the previous step.

Configuring Outbound JMS Transport-Level Security: Main Steps

To configure outbound JMS transport-level security:

1. When you create or edit a JMS business service, on the Transport Configuration page, under Advanced Settings, select the **Use SSL** check box.

Service Bus configures the JMS business service to use the T3S protocol.

2. If the JMS administrator created access control policies that restrict access to a JMS connection pool, configure the business service to authenticate when it connects to the JMS server:

- a. Create a service account to provide the user name and password that the JMS server requires. See [Working with Service Accounts](#).

The JMS service account for the proxy service is used not only for the JMS object access, but also for the JNDI lookup.

You must add a user name and password directly in the service account. JMS cannot use a service account that passes through the credentials that it received from its client's request or that maps a client user name to a Service Bus user.

- b. When you create or edit the business service, on the JMS Transport page, click the **Browse** button next to JMS Service Account. Select the business account that you created in the previous step.

3. Select the **Pass Caller's Subject** check box to have Service Bus pass the authenticated subject when sending a message.

Configuring Transport-Level Security for SFTP Transport

As described in [Using the SFTP Transport](#), Service Bus supports the SFTP transport for inbound and outbound transport-level security. The SFTP transport uses Secure Shell (SSH) version 2 to transfer files.

- [How Two-Way Authentication is Performed](#)

- [Use of the known_hosts File](#)
- [SFTP Transport Authentication Process](#)
- [Configuring Inbound SFTP Transport-Level Security: Main Steps](#)
- [Configuring Outbound SFTP Transport-Level Security: Main Steps](#)
- [SFTP Security Attributes Preserved During Import](#)
- [SFTP Credential Life Cycle](#)

How Two-Way Authentication is Performed

The SFTP authentication is two-way: both the SFTP server and SFTP client (Service Bus service) authenticate each other through different mechanisms:

- The SFTP server uses the authentication method you specified in the Transport Configuration page to authenticate the SFTP client (the Service Bus service): Username Password, Host Based, or Public Key.
- The SFTP client (the Service Bus service) uses a `known_hosts` file to authenticate the SFTP server. The `known_hosts` file on the Service Bus proxy service (inbound requests) or business service (outbound requests) system must have the hostname, IP address, and public key of the remote SFTP servers to which the proxy service or business service can connect. SSH version 2 uses this public key to authenticate the connection.

The SFTP client (the Service Bus service) always uses the `known_hosts` file to determine whether to connect to an SFTP server, no matter which of the three authentication methods is chosen in the Transport Configuration page. That is, in all cases the SFTP server is authenticated by the Service Bus service using the information present in this file. This ensures that the Service Bus service is connecting to a known server.

For example, in case of Username Password authentication, the SFTP Client (Service Bus service) authenticates the SFTP server against the SFTP server's public key in the `known_hosts` file. The SFTP server authenticates the client (Service Bus service) with the user name and password from the service account.

Use of the known_hosts File

No matter which authentication method you choose in the Transport Configuration page, a `known_hosts` file on the Service Bus proxy service (inbound requests) or business service (outbound requests) system must have the hostname, IP address, and public key of the remote SFTP servers to which the proxy service or business service can connect.

The Service Bus service authenticates the SFTP server with the public-key/host/IP combination present in the `known_hosts` file.

Note:

This SSH authentication mechanism is outside of the typical Service Bus service key provider/PKI credential mapper process.

The `known_hosts` file requirement must be satisfied during authentication. SFTP servers not listed in the `known_hosts` file are not authenticated.

Creating the `known_hosts` File

1. Use the editor of your choice to create a `known_hosts` text file.

The format for `known_hosts` is as follows:

```
Hostname,IP algorithm public-key
```

where `Hostname`, `IP`, and `public_key` identify the SFTP server.

The algorithms supported are RSA (entered only as `ssh-rsa`) and DSA (entered only as `ssh-dsa` or `ssh-dss`).

The public key format for this file is "OpenSSH public key format."

For example:

```
getafix,172.22.52.130 ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAtR
+M3Z9HFxnKZTx66fZdnQqAHQcFlvQe1+EjJ/HWYtgAnqsn0hMJzqWMatb/
u9yFwUpZBir jm3g2I9Qd8VocmeHwoGPhDGFQ5LQ/PPo3esE+CGwdnCOyRcktNHeuKxo4kiCCJ/
bph5dRpghCQIvsQvRE3sks+XwQ7Wuswz8pv58=
```

Multiple entries are supported, one entry per line.

2. Move the `known_hosts` file to the following directory:

```
MW_HOME/user_projects/domains/osb_domain/osb/transports/sftp
```

The directories `/transports/sftp` are not created automatically. You must create them.

SFTP Transport Authentication Process

The following general principles apply to the SFTP authentication process for both a proxy service and business service.

- **Connection:** The Service Bus service (proxy and business) always acts as the SFTP client and connects to the SFTP server.
- **Authentication by the SFTP Server:** For Public Key and Host Based authentication, the SFTP server authenticates the connection with the public key of the Service Bus service. For Username Password, the SFTP server authenticates the connection with the user name and password.
- **Authentication by the SFTP Client:** The Service Bus service always authenticates the SFTP server with the public-key/host/IP combination present in the `known_hosts` file.
- **Connection established:** If both the server and client authentications are successful, only then is the connection established and ready for transfer.
- **Transfer:** The file (message) is downloaded in case of the proxy service and uploaded in the case of the business service.

The SFTP authentication process for proxy services is described in [Inbound One-Way Download to the Proxy Service](#). The SFTP authentication process for business services is described in [Outbound One-Way Upload from the Business Service](#).

Inbound One-Way Download to the Proxy Service

Inbound one-way download to the proxy service is described as follows:

1. The proxy service, which is the SFTP client, attempts to connect to the SFTP server.
2. The proxy service is authenticated by the SFTP server using the authentication mechanism selected on the Transport Configuration page.

For Host Based and Public Key authentication, the remote SFTP server uses the host name and public key of the proxy service to authenticate the Service Bus system. For Username Password authentication, the SFTP server uses the user name and password supplied by the proxy service (using the service account) to authenticate the Service Bus system.

3. A `known_hosts` file (on the Service Bus proxy service system) keeps the information of the remote SFTP servers to which the Service Bus proxy service can connect.

Specifically, this file contains the host name, IP address, and public key of the accepted remote servers.

SSH version 2 uses this public key to authenticate the connection. SFTP servers not listed in the `known_hosts` file are not authenticated.

4. If authentication is successful, the proxy service is the SFTP client connected to the remote SFTP server.
5. If allowed by the SFTP server, the proxy service (the SFTP client) polls a remote directory on the SFTP server and downloads any files (messages) present in the remote directory.

The proxy service configuration determines which remote directory to poll, how often to poll it, and what to do with any files (messages) that it downloads.

Outbound One-Way Upload from the Business Service

Outbound one-way upload from the business service is described as follows:

1. The business service (which is the SFTP client) attempts to connect to the SFTP server.
2. The business service is authenticated by the SFTP server using the authentication mechanism selected on the Transport Configuration page.

For Host Based and Public Key authentication, the SFTP server uses the host name and public key of the business service to authenticate the Service Bus system. For Username Password authentication, the SFTP server uses the user name and password (from the service account) to authenticate the Service Bus system.

3. A `known_hosts` file (on the Service Bus business service system) keeps the information of the SFTP servers to which the Service Bus business service can connect.

Specifically, this file contains the host name, IP address, and public key of the accepted servers.

SSH version 2 uses this public key to authenticate the connection. SFTP servers not listed in the `known_hosts` file are not authenticated.

4. If authentication is successful, the business service is the SFTP client connected to the remote SFTP server.
5. If allowed by the SFTP server, the business service (the SFTP client) uploads files to the remote directory on the SFTP server.

The business service configuration determines in which remote directory to upload the file, how often to retry the upload, and any file prefix or suffix to add to the uploaded file name.

Configuring Inbound SFTP Transport-Level Security: Main Steps

To configure inbound transport-level security for a proxy service:

1. Create a `known_hosts` file, as described in [Use of the known_hosts File](#), on the Service Bus proxy service system.

`known_hosts` keeps the information of the remote SFTP servers to which the Service Bus proxy service can connect. Specifically, this file contains the host name, IP address, and public key of the accepted remote servers.

SSH version 2 uses this public key to authenticate the connection. SFTP servers not listed in the `known_hosts` file are not authenticated.

2. When you create a proxy service, on the Transport Configuration page select SFTP.
3. Specify the endpoint URI in `sftp://hostname:port/directory` format, where:
 - `hostname` is the host name or IP address of the SFTP server.
 - `port` is the port on which SFTP server is listening. Default port for SFTP is 22.
 - `directory` is the location that is periodically polled for files. This directory is relative to the home directory of the user.
4. On the SFTP Transport Configuration page, select either Username Password, Host Based, or Public Key authentication.

The authentication choices are summarized here. See [Using the SFTP Transport](#) for complete information.

- Username/Password authentication specifies that a static service account (using user credentials on the SFTP server) is associated with this authentication method. The service account provides a user name and password that the proxy service uses for authentication to the SFTP server. The SFTP client is authenticated using the provided credentials. Only the static service account type is supported.
- Host Based Authentication specifies that only connections from identified, known hosts are allowed. This authentication method requires a user name and a service key provider.

The SFTP Server authenticates the proxy service with the public key of the proxy service.

Note:

The Service Bus proxy service does not itself use the service key provider to authenticate any credentials from the SFTP server. It uses only the `known_hosts` file to authenticate the SFTP server.

The public key of the proxy service is present in the key-pair referred by the service key provider. You need to extract this key when you set up the service key provider, and then configure the SFTP server to use the public key.

For example, with SFTP server on Linux, you need to:

- Edit the `/etc/ssh/shosts.equiv` file and add the host name or IP address of the machine on which Service Bus domain is running.
- Edit the `/etc/ssh/ssh_known_hosts` file and add the host name or IP address of the machine on which Service Bus domain is running, followed by space and the public key.

The user name is used to determine which directory on the SFTP server to poll.

- Public Key specifies a user name and service key provider are required to use this authentication method. Every user has their own private key.

The SFTP Server authenticates the proxy service with the public key.

Note:

The Service Bus proxy service does not itself use the service key provider to authenticate any credentials from the SFTP server. It uses only the `known_hosts` file to authenticate the SFTP server.

The public key of the proxy service is present in the key-pair referred by the service key provider. You need to extract this key when you set up the service key provider, and then configure the SFTP server to use the public key.

For example, to allow access to a system for a given identity with an SFTP server on Linux, place the public key in a `$HOME/.ssh/authorized_keys` file on that system. All keys listed in that file are allowed access.

The user name is used to determine which directory on the SFTP server to poll. It is also use to identify the location of the public key on the SFTP server.

5. If allowed by the remote SFTP server, the proxy service (SFTP client) polls a remote directory on the SFTP server and downloads any files present in the remote directory.

The proxy service configuration determines which remote directory to poll, how often to poll it, and what to do with any files (messages) that it downloads.

The directory to be polled is an absolute path.

Configuring Outbound SFTP Transport-Level Security: Main Steps

To configure outbound transport-level security for a business service:

1. Create a `known_hosts` file, as described in [Use of the known_hosts File](#), on the Service Bus business service system.

`known_hosts` keeps the information of the remote SFTP servers to which the Service Bus business service can connect. Specifically, this file contains the host name, IP address, and public key of the accepted remote servers.

SSH version 2 uses this public key to authenticate the connection. SFTP servers not listed in the `known_hosts` file are not authenticated.

2. When you create a business service, on the Transport Configuration page select SFTP.
3. Specify the endpoint URI in `sftp://hostname:port/directory` format, where:
 - `hostname` is the host name or IP address of the SFTP server.
 - `port` is the port on which SFTP server is listening. Default port for SFTP is 22.
 - `directory` is the location to which files are uploaded. This directory is relative to the home directory of the user.
4. On the SFTP Transport Configuration page, select either Username Password, Host Based, or Public Key authentication.

The authentication choices are summarized here. See [Using the SFTP Transport](#) for complete information.

- Username/Password authentication specifies that a static service account (using user credentials on the SFTP server) is associated with this authentication method. The service account provides a user name and password that the business service uses for authentication to the SFTP server. The SFTP client is authenticated using the provided credentials. Only the static service account type is supported.
- Host Based Authentication specifies that only connections from identified, known hosts are allowed. This authentication method requires a user name and a service key provider.

The SFTP Server authenticates the business service with the public key of the business service.

Note:

The Service Bus business service does not itself use the service key provider to authenticate any credentials from the SFTP server. It uses only the `known_hosts` file to authenticate the SFTP server.

The public key of the business service is present in the key-pair referred by the service key provider. You need to extract this key when you set up the service key provider, and then configure the SFTP server to use the public key.

For example, with SFTP server on Linux, you need to:

- Edit the `/etc/ssh/shosts.equiv` file and add the host name or IP address of the machine on which Service Bus domain is running.

- Edit the `/etc/ssh/ssh_known_hosts` file and add the host name or IP address of the machine on which Service Bus domain is running, followed by space and the public key.

The user name is used to determine the upload directory on the SFTP server.

- Public Key specifies a user name and service key provider are required to use this authentication method. Every user has their own private key.

The SFTP Server authenticates the business service with the public key.

Note:

The Service Bus business service does not itself use the service key provider to authenticate any credentials from the SFTP server. It uses only the `known_hosts` file to authenticate the SFTP server.

The public key of the business service is present in the key-pair referred by the service key provider. You need to extract this key when you set up the service key provider, and then configure the SFTP server to use the public key.

For example, to allow access to a system for a given identity with an SFTP server on Linux, place the public key in a `$HOME/.ssh/authorized_keys` file on that system. All keys listed in that file are allowed access.

The user name is used to determine the upload directory on the SFTP server and for identifying the location of the public key on the SFTP server.

5. If allowed by the remote SFTP server, the business service (SFTP client) uploads files to the remote directory on the SFTP server.

The business service configuration determines in which remote directory to upload the file, how often to retry the upload, and any file prefix or suffix to add to the uploaded file name.

The upload directory is an absolute path and is automatically created.

SFTP Security Attributes Preserved During Import

The following security attributes are preserved when the **Preserve Security and Policy Configuration** option is selected during import:

- Client authentication method
- Reference to the service account (in case of Username Password authentication)
- Reference to the service key provider (in case of Host Based and Public Key authentication)
- Username (in case of Host Based and Public Key authentication)

SFTP Credential Life Cycle

Whenever the user name/password or public key credential changes, the SFTP transport drops all idle connections made with the previous credential and attempts to reconnect. For active connections, the SFTP transport drops the connection after the current operation is finished.

Email, FTP, and File Transport-Level Security

The following sections describe the security measures that are available for communication over the email, FTP, and file protocols:

- [Email and FTP Transport-Level Security](#)
- [File Transport Security](#)

Email and FTP Transport-Level Security

Email and FTP are not secure protocols. They support weak authentication, typically over insecure channels. The supported security method for email or FTP transport is the user name and password needed to connect to the email or FTP server.

To secure email, you must designate a service account as an alias for the user name and password. The service will use the user name and password to authenticate to the SMTP server.

To secure the FTP transport, select **external_user** and designate a service account as an alias for the user name and password. The service will use the user name and password to authenticate to the FTP server.

For information about how to add security to email and FTP transport, see [Creating and Configuring Business Services](#)

File Transport Security

The supported security method for file transport is the user login to the computer on which the files are located.

The SFTP transport, described in [Configuring Transport-Level Security for SFTP Transport](#), is the preferred mechanism to secure FTP.

Configuring Transport-Level Security for SB Transport

The Service Bus (SB) transport allows client Service Bus servers to invoke a Service Bus proxy service synchronously using RMI. RMI is the only mechanism by which client Service Bus servers can access the SB transport. In this release of Service Bus the associated API is for internal use only and is not documented.

The SB proxy service is accessed in one of two ways:

- By a client Service Bus server that uses an SB business service to connect to the Service Bus server of the proxy service by using the JNDI context and the proxy service URI.
- By products such as Oracle WebLogic Integration and Oracle Data Service Integrator that use proprietary artifacts to access SB proxy services. These artifacts are unique to those products and are not described here.

The SB business service can send messages only to SB proxy services. A JNDI provider, which is specified in the endpoint URI of the business service, is used to do a JNDI lookup on the remote Service Bus server. Specifically, the JNDI provider points to the Service Bus server where the service is deployed to retrieve the RMI stubs corresponding to the SB proxy service.

For example, the endpoint URI you specify in the business service could be `sb://some_secured_jndi_provider/some_remote_sb_proxy`.

A secure JNDI provider should have a provider URL with a secure protocol. In the SB business service case, you can use the HTTPS or t3s protocols.

The service account (of the business service) specifies the user credentials that should be used for invoking the remote SB proxy service. If no service account is specified, the user credentials of the inbound proxy service (the inbound client) of this business service are used for security context propagation.

The SB transport can use SSL to require strong server and client authentication. Before you can use the SB transport with SSL, you must configure SSL in WebLogic Server. See [Configuring the Oracle WebLogic Security Framework: Main Steps](#).

Caution:

When set, the Use SSL flag means that request must be sent over an SSL connection. However, the SB transport does not forbid unsecured connections. The proxy service will be advertised (through the effective WSDL file or UDDI) with a secured URI (indicated by `sbs`), but secured access is not enforced.

The Service Bus server administrator must close all unsecured protocols on the server (t3, http, and so forth) to strictly enforce secured-client connections.

Configuring SAML Authentication With Service Bus (SB) Transport

If you are using SAML-based authentication with the SB transport, be sure to follow these configuration requirements:

- On the SB client side, configure a SAML Credential mapper provider and create a SAML relying party for each SB proxy service you plan to invoke from this client. In the target URL field enter `http://openuri.org/<OSBProxyServiceURI>`, where `OSBProxyServiceURI` is the service URI of the SB proxy service.
- On the Service Bus side (where the SB proxy service resides), configure a SAML Identity Assertion provider and create a SAML asserting party. In the target URL field enter the service URI of the SB proxy service. Do not include the SB protocol or host/port information. For example, `/<OSBProxyServiceURI>`.

Configuring Transport-Level Security for WS Transport

Web Services Reliable Messaging (WS-RM) functionality is available in Service Bus as the WS transport. Service Bus supports the specification submitted in February 2005. For more information about the specification, see Web Services Reliable Messaging Protocol (WS-ReliableMessaging) at <http://schemas.xmlsoap.org/ws/2005/02/rm/>.

The WS transport has both proxy service (inbound) and business service (outbound) components that are based on SOAP1.1- and SOAP1.2-based WSDL files, along with WS-RM policy. It supports both one-way and request-response patterns, but response is unreliable.

Reliable Web Services Messaging Defined

As described in "Overview of Web Service Reliable Messaging" in *Programming Advanced Features of JAX-RPC Web Services for Oracle WebLogic Server*, WS-RM is a framework in which an application running in one application server can reliably invoke a web service running on another application server, assuming that both servers implement the WS-ReliableMessaging specification. "Reliable" is defined as the ability to guarantee message delivery between the two web services. In particular, the specification describes an interoperable protocol in which a message sent from a source endpoint (or client web service) to a destination endpoint (or web service whose operations can be invoked reliably) is guaranteed either to be delivered, according to one or more delivery assurances, or to raise an error.

WS Transport Resources Visible in WLS Console

WS proxy services are visible from the Oracle WebLogic Server Administration Console, but attempts to assign policies from WLS are ignored.

Specifically, administrators can navigate to the **Home > Summary of Security Realms > myrealm > Realm Roles** pages in the Oracle WebLogic Server Administration Console and seemingly edit the security policy for the WS proxy service.

However, this policy will have no effect and it will not be evaluated at runtime.

The EAR application is auto-generated and deployed by Service Bus when you activate the session. This is one EAR file for each WS proxy service.

Use of WS-Policy Files for Web Service Reliable Messaging Configuration

You configure WS transport security through WS-Policy files, either from a WSDL file or bound directly to the service.

Service Bus use WS-Policy files to enable a destination endpoint to describe and advertise its WS-RM capabilities and requirements. The WS-Policy specification provides a general purpose model and syntax to describe and communicate the policies of a web service.

These WS-Policy files are XML files that describe features such as the version of the supported WS-ReliableMessaging specification, the source endpoint's retransmission interval, the destination endpoint's acknowledgment interval, and so on.

WS-Policy with RM assertions and WSSP transport-level security assertions are supported for the WS transport only.

Preconfigured WS-RM Policy Files

Service Bus includes two simple WS-RM WS-Policy files that you can specify if you do not want to create your own WS-Policy files:

- `DefaultReliability.xml` – Specifies typical values for the reliable messaging policy assertions, such as inactivity timeout of 10 minutes, acknowledgment interval of 200 milliseconds, and base re-transmission interval of 3 seconds.
- `LongRunningReliability.xml` – Similar to the default reliable messaging WS-Policy file, except that it specifies a much longer activity timeout interval (24 hours.)

You cannot change these pre-packaged files. If their values do not suit your needs you must create your own WS-Policy file.

For example, the complete `LongRunningReliability.xml` file (as extracted from `weblogic.jar`) is shown in the following example:

Example - LongRunningReliability.xml File

```
<?xml version="1.0"?>
<wsp:Policy
  xmlns:wsm="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:beapolicy="http://www.bea.com/wsm/policy"
  >
  <wsm:RMAssertion >
    <wsm:InactivityTimeout
      Milliseconds="86400000" />
    <wsm:BaseRetransmissionInterval
      Milliseconds="3000" />
    <wsm:ExponentialBackoff />
    <wsm:AcknowledgementInterval
      Milliseconds="200" />
    <beapolicy:Expires Expires="PlM" optional="true"/>
  </wsm:RMAssertion>
</wsp:Policy>
```

RM WS-Policy Required Prior to Activation

A proxy or business service that uses the WS transport must have a WS-Policy with RM assertions, either from a WSDL file or bound directly to the service. Services that use any other transport must not have a WS-Policy with RM assertions.

You can bind RM assertions only at the service level and not at the operation or request/response levels.

Async Responses

WS-RM supports two messaging patterns: one way, and request/response. The WS transport supports both patterns, but does not support reliable response. That is, the response is not sent reliably but the request is always reliable.

Async responses from a proxy service using the WS transport to an RM client connect to the `AcksTo` or `ReplyTo` endpoint references specified by the RM client. The RM client is free to use an HTTP or HTTPS URL. When using HTTPS, the RM client is free to request a client certificate during the SSL handshake. The WS transport will use the SSL key-pair of the service key provider upon request.

Proxy Service Authentication

The WS transport supports the following HTTPS security modes using WS-Policy files:

- HTTPS – no client authentication
- HTTPS with basic authentication
- HTTPS with client certificate authentication (2-way SSL)

[Table 51-1](#) shows the preconfigured security policies that implement these modes and indicates when you might use them.

Table 51-1 WS Transport Authentication Matrix

HTTPS Required	Authentication Required	Preconfigured Transport Security Policy
Yes	None	Wssp1.2-Https.xml
Yes	Basic	Wssp1.2-HttpsBasic.xml
Yes	Client certificate	Wssp1.2-HttpsClientCert.xml

WS proxy services support both basic and client certificate (2-way SSL) authentication, as determined by the WSSP 1.2 transport-level security assertions in the WS-Policy.

Consider the example of the HTTPS token and the Basic256 algorithm as extracted from the packaged Wssp1.2-Https.xml policy, as shown in the following example.

When basic authentication is specified in the WS-policy, all HTTPS requests (including RM protocol messages to the WS proxy service) must have a valid user name and password.

Example - Wssp1.2-Https.xml File (Partial)

```

:
<sp:TransportBinding>
  <wsp:Policy >
    <sp:TransportToken>
      <wsp:Policy>
        <sp:HttpsToken />
      </wsp:Policy>
    </sp:TransportToken>
    <sp:AlgorithmSuite>
      <wsp:Policy>
        <sp:Basic256/>
      </wsp:Policy>
    </sp:AlgorithmSuite>
    <sp:Layout>
      <wsp:Policy>
        <sp:Lax/>
      </wsp:Policy>
    </sp:Layout>
    <sp:IncludeTimestamp/>
  </wsp:Policy>
</sp:TransportBinding>
</wsp:Policy>

```

Proxy service authentication is supported as follows:

- Outbound client certificate authentication using the SSL key-pair assigned to the service key provider configured for the proxy service.

If you plan to create a service key provider (which passes key-certificate pairs in outbound requests), use the Oracle WebLogic Server Administration Console to configure a PKI credential mapping provider. In any WebLogic Server domain that hosts Service Bus, you can configure at most one PKI credential mapping provider.

- Username/password identity propagation through a WS proxy service (with basic authentication) to any other outbound transport, or outbound WSS user name token.

If a business service requires user name and password tokens, you can configure the business service's service account to pass through the user credentials from the original client request. See [Working with Service Accounts](#).

- Credential mapping between WS proxy service (with basic or 2-way SSL authentication) and any other transport.
- Sending (non-reliable) asynchronous responses from a WS proxy service to an RM client using HTTP or HTTPS. The default protocol used by proxy and business services is HTTP.

Asynchronous responses from a WS proxy service to an RM client connect to the `AcksTo` or `ReplyTo` endpoint references specified by the RM client. The RM client can use either HTTP or HTTPS URL. If the RM client uses HTTPS, the RM client can request a client certificate during the SSL handshake. The WS transport uses the SSL key-pair of the service key provider upon request.

Preserving Security Configuration on Import

If the `Preserve Security and Policy Configuration` flag is set, the WS transport provider preserves the following security configuration: The reference to the service account (WS business services only)

Configuring Inbound and Outbound WS Transport-Level Security

You configure WS transport security through WS-Policy, either from a WSDL file or bound directly to the service.

Configuring Transport-Level Security for WebSphere Message Queue Transport

Service Bus provides support for a native Message Queue (MQ) transport that can send messages to and from WebSphere MQ. In this context, the MQ transport is a client that connects to an MQ Server using MQ libraries.

You configure the security-related properties for the transport when you create an MQ Connection resource. These properties are then used by the MQ proxy or business service.

Note:

Make sure that you add the MQ client libraries to your environment, as described in [How to Add MQ Client Libraries to Your Environment](#).

The MQ Connection resource has two modes:

binding mode – You use the binding mode to connect to the MQ Queue Manager located on the same machine as Service Bus. In this mode, the service calls directly into the existing queue manager API rather than communicating over the network. This mode provides a fast path to connect to local queue managers.

TCP mode – You use the `tcp` mode when the MQ Queue Manager is not available on the same machine as Service Bus.

Configuring Inbound MQ Transport-Level Security: Main Steps

To configure inbound transport-level security for a proxy service:

1. Before you create a proxy service that uses the MQ transport, create an MQ Connection resource for the transport to use. Choose from the following security configuration settings:
 - **SSL Required.** Select the check box to use HTTPS for sending messages. Only server-side SSL (server authenticates to client) is supported when the 2-way SSL Required option is not selected.
 - **Cipher Suite.** This option is available only when the SSL Required check box is selected. Select the Cipher Suite algorithm to be used by SSL.

A cipher suite is an SSL encryption method that includes the key exchange algorithm, the symmetric encryption algorithm, and the secure hash algorithm. A cipher suite is used to protect the integrity of a communication.

The Cipher Suite algorithm is used to encrypt and decrypt message communications between the WebSphere MQ server and the MQ Transport.
 - **2-way SSL Required.** This option is available only when the SSL Required check box is selected. Select the check box to force the use of both client-side and server-side SSL authentication.
 - **Reference to the Service Key Provider.** If you select 2-way SSL Required, you must provide a reference to the service key provider for obtaining the appropriate key manager for client-side SSL.

Enter the path (project/folder) and name of a service key provider, or click Browse to select one from the Select Service Key Provider page.
 - **Reference to the Static Service Account.** Required for user name and password authentication. Enter the path (project/folder) and name of a static service account, or click Browse to select a service account.
2. When you create a proxy service, on the Transport Configuration page select `mq`.

Configuring Outbound MQ Transport-Level Security: Main Steps

To configure outbound transport-level security for a business service:

1. Before you create a proxy service that uses the MQ transport, create a MQ Connection resource for the transport to use. Choose from the following security configuration settings:
 - **SSL Required.** Select the check box to use HTTPS for sending messages. Only server-side SSL (server authenticates to client) is supported when the 2-way SSL Required option is not selected.
 - **Cipher Suite.** This option is available only when the SSL Required check box is selected. Select the Cipher Suite algorithm to be used by SSL.

A cipher suite is an SSL encryption method that includes the key exchange algorithm, the symmetric encryption algorithm, and the secure hash algorithm. A cipher suite is used to protect the integrity of a communication.

The Cipher Suite algorithm is used to encrypt and decrypt message communications between the WebSphere MQ server and the MQ Transport.

- **2-way SSL Required.** This option is available only when the SSL Required check box is selected. Select the check box to force the use of both client-side and server-side SSL authentication.
- **Reference to the Service Key Provider.** If you select 2-way SSL Required, you must provide a reference to the service key provider for obtaining the appropriate key manager for client-side SSL.

Enter the path (project/folder) and name of a service key provider, or click Browse to select one from the Select Service Key Provider page.

- **Reference to the Static Service Account.** Required for user name and password authentication. Enter the path (project/folder) and name of a static service account, or click Browse to select a service account.

2. When you create a business service, on the Transport Configuration page select **mq**.

Transport-Level Security Elements in the Message Context

If you configure a proxy service to authenticate clients, then you can access the client's identity and the security groups to which the client belongs from the proxy service's pipeline. The identity and group information is located in the message context at

```
$inbound/ctx:security/ctx:transportClient/ctx:username
```

and

```
$inbound/ctx:security/ctx:transportClient/ctx:principals/ctx:group
```

(the message context contains one `ctx:group` element for each group the user belongs to)

If a proxy service does not authenticate clients, then the value of `$inbound/ctx:security/ctx:transportClient/ctx:username` is `<anonymous>` and there will not be any `ctx:group` elements.

For more information, see [Inbound and Outbound Variables](#).

Securing Oracle Service Bus with Oracle Web Services Manager

This chapter describes how to use Service Bus in conjunction with Oracle Web Services Manager (OWSM) to provide a scalable, standards-based, centrally managed approach to securing your service integration environment with WS-Security policies while leveraging your existing security providers.

OWSM is a runtime framework for policy creation for security, management, and governance. You create policies, attach them to services in Service Bus, and enforce those policies at various points in the messaging life cycle with OWSM agents.

OWSM policies enable WS-AT, WS-RM and WS-Security/WS-SC. WSDL-embedded policies cannot be used.

Note:

Oracle Web Services Manager (OWSM) is the Web Services security mechanism used by Service Bus. All newly created resources, such as business services and proxy services, use OWSM policies for security. WLS 9 policies are deprecated, and cannot be used to configure security for a new Service Bus resource.

However, you can import resources already configured with WLS 9 policies into your Service Bus project. You cannot edit or modify these WLS 9 policies, but you can replace them with OWSM policies.

This chapter includes the following sections:

- [About Oracle Web Services Manager Integration with Oracle Service Bus](#)
- [Using Oracle Web Services Manager with Oracle Service Bus](#)

For more information about Oracle Web Services Manager, see *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

About Oracle Web Services Manager Integration with Oracle Service Bus

OWSM is a component of the Oracle Enterprise Manager Fusion Middleware Control, a runtime framework that provides centralized management and governance of Oracle SOA Suite environments and applications. You create and configure OWSM policies in Oracle Enterprise Manager, and those policies are persisted in a policy store (a database is recommended). OWSM lets you define policies against an LDAP directory and generate standard security tokens (such as SAML tokens) to propagate identities across multiple web services used in a single transaction.

In Service Bus, when defining a business or proxy service that lets you attach security policies, you can attach available OWSM policies.

Security Providers

Service Bus and Oracle Web Services Manager (OWSM) use certain services for authentication and authorization. OWSM uses Java Platform Security (JPS), so Service Bus uses JPS providers for OWSM policies. Service Bus also uses Common Security Services (CSS), which is a part of Oracle Platform Security Services (OPSS), for other aspects of message security.

For more information about Oracle security services, see "Introduction to Oracle Platform Security Services" in the *Securing Applications with Oracle Platform Security Services*.

The following topics describe which security providers Service Bus and OWSM use for different security areas.

JPS Providers

When using Oracle Web Services Manager policies, the following apply:

- OWSM policies use SAML providers from JPS and not from WebLogic Server. For information on configuring SAML with OWSM, see "Configuring SAML" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- For authentication, OWSM uses the JPS Login Module, which in turn calls authentication providers configured on WebLogic Server.
- OWSM and Service Bus support the Java Keystore (JKS) and the Keystore Service (KSS) provided by Oracle Platform Security Services. For OWSM policies, a best practice is to configure the keystore on JPS, with both the WebLogic Server and the JPS keystore referencing the same kind of keystore. For example, if you use a JKS file keystore, JPS and WebLogic Server should point to the same file. If you use an KSS keystore, JPS and WebLogic Server should point to the same KSS configuration.

For information on creating the keystore, see "Configuring Keystores for Message Protection" in the *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

- A JPS keystore serves as both a keystore and a truststore for OWSM policies.

CSS Providers

Service Bus uses the following providers:

- CSS providers to enforce WLS 9 policies
- CSS providers to enforce transport security
- WebLogic Server authorization providers for authorization policies
- Custom WebLogic Server authentication providers and identity asserters for custom authentication policies
- WebLogic Server credential providers and mappers
- WebLogic Server keystore and truststore for WLS 9 policies

- Authentication and identity assertion through Oracle Web Services Manager agents

Using Oracle Web Services Manager with Oracle Service Bus

This section includes the following topics:

- [Attaching Oracle Web Services Manager Policies to Oracle Service Bus Services](#)
- [Configuring SAML](#)
- [Advertising WSDL Files to Support WS Standards](#)
- [Deployment Considerations](#)
- [Auditing](#)
- [Monitoring Statistics](#)
- [Predefined Policies and Unsupported Assertions](#)

Attaching Oracle Web Services Manager Policies to Oracle Service Bus Services

You can attach OWSM policies to the following types of proxy and business services in Service Bus on the Policies page:

- WSDL
- Any SOAP
- Any XML
- Messaging
- REST

You can attach OWSM policies only at the service level, and you cannot embed them in service WSDL files. Note that policy support for non-SOAP services is limited.

When attaching policies in the development environment, keep in mind that services in the development environment can be out of sync with services in the Oracle Service Bus Console, so take care when updating services from JDeveloper to the Console. If you copy a service to create a same type of service (for example, copy a business service to create a new business service), be sure to review your OWSM policies in the new service and make any necessary adjustments.

Policy Overrides

After adding OWSM policies to a service, you can provide policy overrides on the Security page. For the policies used, the user interface displays the override keys (properties) and their default values. The key names come from the policy binding. If allowed, a text box appears next to a key's default value where you can provide an override value. Service Bus does not provide well-known keys for override, such as sign key alias or CSF key, which points to user credentials in a CSF store. (Service Bus provides user credentials in the service account.) Override keys you provide are passed to the Oracle Web Services Manager agent during invocation.

Configuring SAML

For information on configuring SAML for use with Service Bus, see [Using SAML with Oracle Service Bus](#). For information on configuring SAML with OWSM, see

"Configuring SAML" in the *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Advertising WSDL Files to Support WS Standards

When WSDL files contain embedded Oracle Web Services Manager policies, you can advertise the policies to be compatible with the following policy standards, supported by Oracle Service Bus and Oracle SOA Suite:

- WS-Policy 1.2 (default) and 1.5
- WS-Security Policy 1.1 (default), 1.2, and 1.3

Using special query parameters in URLs to access WSDL files embedded with OWSM policies, Service Bus generates WSDL files that comply with the required standards. For more information on accessing WSDL files with a URL, see [Viewing Service Bus Resources in a Web Browser](#).

Note:

This feature is not available in the Service Bus "Export WSDL" or "Generate WSDL" functionality.

The special query parameters are **&wsp** (WS-Policy) and **&wssp** (WS-Security Policy), and you can use them in conjunction with the WSDL, PROXY, and BIZ URL patterns for retrieving WSDL files. For example:

- `http://localhost:7001/proxy/myProxy?WSDL&wsp=1.5&wssp=1.2`
Returns the WSDL file for myProxy, a WSDL-based proxy service, so that the OWSM policy reference conforms to WS-Policy 1.5 and WS-Security Policy 1.2.

Note:

In the previous URL, `/proxy/myProxy` is the endpoint URI for the proxy service.

- `http://localhost:7001/sbresource?PROXY/myProject/myProxy&wsp=1.5&wssp=1.2`
Returns the WSDL file for myProxy, a WSDL-based proxy service, so that the OWSM policy reference conforms to WS-Policy 1.5 and WS-Security Policy 1.2.
- `http://localhost:7001/sbresource?BIZ/myProject/myBiz&wsp=1.5&wssp=1.3`
Returns the WSDL file for myBiz, a WSDL-based business service, so that the OWSM policy reference conforms to WS-Policy 1.5 and WS-Security Policy 1.3.
- `http://localhost:7001/sbresource?WSDL/proxy/myProxy`
Returns the WSDL file for myProxy, a WSDL-based proxy service, so that the OWSM policy reference conforms to WS-Policy 1.2 and WS-Security Policy 1.1. Because no query parameters are used, Service Bus uses the defaults.
- `http://localhost:7001/proxy/myProxy?WSDL&wssp=1.3`

Because WS-Security Policy 1.3 is compatible only with WS-Policy 1.5, this returns the WSDL file for myProxy so that the OWSM policy reference conforms to WS-Security Policy 1.3 and WS-Policy 1.5.

- Invalid Values/Combinations

WS-Security Policy 1.2 and 1.3 are compatible only with WS-Policy 1.5. For invalid value examples, see [Table 52-1](#).

Tip:

In a web browser, try different query parameter versions see how the returned WSDL changes.

For a quick reference of query parameter combinations, see the following section, [WSDL Query Parameter Reference for WS Policies](#).

WSDL Query Parameter Reference for WS Policies

This section provides a quick reference showing valid and invalid combinations of the `&wsp` and `&wssp` query parameters described in the previous section, [Advertising WSDL Files to Support WS Standards](#). The examples use `?WSDL` to retrieve the WSDL file. You can also use the `?PROXY` and `?BIZ` methods of WSDL file retrieval, as described in [Viewing Service Bus Resources in a Web Browser](#).

As shown in [Table 52-1](#), when one or more parameters is omitted, Service Bus provides the valid default. For the invalid value exceptions, WS-Security Policy 1.2 and 1.3 are compatible with only WS-Policy 1.5, and vice versa.

Table 52-1 Valid and Invalid Combinations of the `&wsp` and `&wssp` Query Parameters

Query Parameter Combinations	WS-Policy Version	WS-Security Policy Version
...?WSDL	1.2	1.1
...?WSDL&wsp=1.2	1.2	1.1
...?WSDL&wsp=1.5	1.5	1.3
...?WSDL&wssp=1.1	1.2	1.1
...?WSDL&wssp=1.2	1.5	1.2
...?WSDL&wssp=1.3	1.5	1.3
...?WSDL&wsp=1.2&wssp=1.1	1.2	1.1
...?WSDL&wsp=1.5&wssp=1.2	1.5	1.2
...?WSDL&wsp=1.5&wssp=1.3	1.5	1.3
...?WSDL&wsp=1.2&wssp=1.2	Invalid value exception	Invalid value exception
...?WSDL&wsp=1.2&wssp=1.3	Invalid value exception	Invalid value exception
...?WSDL&wsp=1.5&wssp=1.1	Invalid value exception	Invalid value exception
...?WSDL&wsp=3.0&wssp=1.2	Invalid value exception	Invalid value exception
...?WSDL&wsp=1.2&wssp=2.0	Invalid value exception	Invalid value exception

Deployment Considerations

When you export Service Bus configurations that contain services with OWSM policy references, the references are maintained. You must ensure that the referenced policies also exist in the target environment. If the target environment is the IDE, warnings are displayed saying that policies will be validated on publish.

Auditing

To audit policy events in Oracle Enterprise Manager, you must set up an audit data repository and set up event collection. For more information, see the following topics in *Administering Web Services*:

- "Managing Audit Data Collection and Storage"
- "Viewing Audit Reports" – Pre-defined audit reports for OWSM in Oracle Business Intelligence Publisher include statistics for Service Bus.

You can audit the following policy-level events:

- Policy creation, deletion, or modification
- Assertion template creation, deletion, or modification

Monitoring Statistics

Fusion Middleware Control lets you monitor and manage policies attached to your Service Bus services, including their usage and violation metrics. You can also attach policy sets globally, define policy overrides, and attach and detach policies from your services. See "Monitoring and Managing Security Policies" in *Administering Oracle Service Bus* for details on monitoring and managing these policies.

Service Bus collects WS-Security error statistics for OWSM policy enforcement errors as it does for WLS 9 policies, and those statistics are available in the Service Bus service monitoring dashboard.

Predefined Policies and Unsupported Assertions

This section lists, and provides links to, the OWSM predefined policies and assertions that Service Bus supports and does not support. Custom assertions are supported.

Note:

The assertion or policy "enabled/disabled" option in the Oracle Enterprise Manager Fusion Middleware Control user interface does not determine whether or not an assertion or policy is supported in Service Bus. Supported policies and assertions are listed in this section.

- [Predefined Policies](#)
- [wss_http_token_*_policy Guidelines](#)
- [OWSM Authentication Policy Guidelines](#)
- [OWSM Policies and SOAP with Attachments \(SwA\)](#)

- [OWSM Policies and MTOM-Formatted Messages](#)
- [Unsupported Assertions](#)

Predefined Policies

See the following topics for information about which OWSM policies are supported with Service Bus:

- **SOAP Services:** See "Determining Which Predefined Policies to Use" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*, which provides information about the predefined policies available in the current release.
- **Non-SOAP Services:** See [Table 52-2](#), which lists the supported OWSM predefined policies for business and proxy services that are configured with the WSDL (non-SOAP), XML, or Messaging service type using the HTTP Transport. User-defined policies are also supported.
- **REST Services:** See "Which OWSM Policies Are Supported for RESTful Web Services and Clients?" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*, which lists supported OWSM predefined policies for services configured with the REST service type.
- **JCA Services:** See "Which OWSM Policies Are Supported for JCA Adapters?" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Note:

In the development environment, if you use unsupported seed policies:

- An effective WSDL file generated in the development environment will skip unsupported policies.
 - Validation is performed on service publish.
-
-

For more information on the following policies, see "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Table 52-2 Supported OWSM Predefined Policies for WSDL (non-SOAP), XML, and Messaging Service Service Types with HTTP Transport

Type	Client Policy	Service Policy
Authentication only	oracle/wss_http_token_client_policy Basic authentication only. For more information on this policy, see wss_http_token_*_policy Guidelines and OWSM Authentication Policy Guidelines .	oracle/wss_http_token_service_policy Basic authentication only. For more information on this policy, see wss_http_token_*_policy Guidelines and OWSM Authentication Policy Guidelines .
Authentication and Message Protection	oracle/wss_http_token_over_ssl_client_policy For more information on this policy, see wss_http_token_*_policy Guidelines and OWSM Authentication Policy Guidelines .	oracle/wss_http_token_over_ssl_service_policy For more information on this policy, see wss_http_token_*_policy Guidelines and OWSM Authentication Policy Guidelines .

Table 52-2 (Cont.) Supported OWSM Predefined Policies for WSDL (non-SOAP), XML, and Messaging Service Service Types with HTTP Transport

Type	Client Policy	Service Policy
Authorization only	N/A	oracle/whitelist_authorization_policy
Authorization only	N/A	oracle/binding_authorization_denyall_policy
Authorization only	N/A	oracle/ binding_authorization_permitall_policy

wss_http_token_*_policy Guidelines

This section provides guidance on using the wss_http_token policies with Service Bus.

Note:

When using the HTTP transport with an OWSM policy, set the **Authentication** property for the transport to **None**. Setting it to any other value conflicts with the OWSM policy.

When you enable specific options on the policies in OWSM, certain guidelines apply. The options are:

- **Authentication Mode** – OWSM and Service Bus support only "Basic" authentication mode in the policy. Any other mode causes an exception.
- **Transport Security** – This option indicates that the invocation has to be done on the SSL channel. At runtime:
 - Proxy Services: If you enable this option on the policy, you must enable the **HTTPS Required** option on the proxy service containing the policy.
 - Business Services: No validation occurs on the business service configuration when you enable this option on the policy, so be sure that the business service endpoint addresses use HTTPS. A runtime error is thrown if an endpoint does not use HTTPS.
- **Mutual Authentication Required** – This option indicates two-way SSL.
 - Proxy Services: This option is not supported for use on proxy services. Clear this option when using the wss*_over_ssl*_policy policies provided by OWSM.
 - Business Services: Because OWSM ignores this option on outbound messages, this option has no effect when used with business services.
- **Include Timestamp** – This option enforces the inclusion of timestamp in the SOAP header.
 - Proxy Services: When you enable this option with proxy services, OWSM ensures the timestamp is available and valid in the SOAP header.

- Business Services: When you enable this option with business services, OWSM adds a timestamp to the SOAP header if a timestamp does not already exist.

Note:

When applying `wss_http_token` policies to proxy and business services that use non-SOAP service types with the HTTP transport, the **Include Timestamp** option in the OWSM policy must be disabled.

OWSM Authentication Policy Guidelines

When you use token transport policies on a Service Bus service, such as `wss_http_token_over_ssl_client_policy` or `wss_username_token_over_ssl_client_policy`, the **Authentication** property on the service's transport configuration page to **None**. You can use either an OWSM token policy or handle authentication through the transport, but not both.

OWSM Policies and SOAP with Attachments (SwA)

You can attach any of the supported OWSM policies to proxy and business services that include support for SOAP with Attachments (SwA). In addition to securing the message body, you can configure message protection policies to include SwA attachments and MIME headers for message signing or message encryption.

OWSM Policies and MTOM-Formatted Messages

You can attach OWSM policies to proxy and business services that include support for MTOM-formatted SOAP messages and use the HTTP, Local, or SB Transport. Message processing for all supported policies is the same for SOAP messages in MTOM format as for any other SOAP message, with the exception of message protection policies.

For a message protection policy that encrypts any part of the message body, the `<xop:Include>` elements of incoming messages are inlined prior to decryption. On decryption, the `<xop:Include>` elements are replaced by the `base64Binary` representation of the data. The message contents do not change.

Note:

When applying OWSM policies to services that support messages in MTOM format:

- For proxy services, the incoming message must first be encrypted using an equivalent client policy.
 - Set the **Authentication** property for the transport to **None** when configuring the service. Setting it to any other value conflicts with the OWSM policy.
-
-

Unsupported Assertions

[Table 52-3](#) lists unsupported OWSM assertions for both SOAP and non-SOAP services, shows which policies contain the assertions, and describes the affected capabilities and alternatives to achieve the capabilities.

Table 52-3 *Unsupported assertions*

Unsupported Assertion	OWSM Policies Containing the Assertion	Capability Affected and Alternative
binding-permission-authorization	oracle/ binding_permission_authorization_policy	Permission-based access control to service. Alternative: Use XACML authorization policies.
OptimizedMimeSerialization (MTOM)	oracle/wsmtom_policy	MTOM Alternative: Use MTOM configuration directly on proxy/business service.
RMAssertion	oracle/wsrn10_policy oracle/wsrn11_policy	WS-RM 1.0/1.1 Alternative: Use the WS transport directly in Service Bus for WS-RM 1.0.
sca-component-authorization	oracle/ component_authorization_denyall_policy oracle/ component_authorization_permitall_policy	Role-based access control to deny/permit all to access the component. Alternative: Not applicable
sca-component-permission-authorization	oracle/ component_permission_authorization_policy	Permission based Access Control to component Alternative: Not applicable
UsingAddressing	oracle/wsaddr_policy	To require WS-Addressing Alternative: Configure WS-Addressing on business services that use the SOA-DIRECT transport; or add WS-Addressing to messages in a Service Bus pipeline.

Custom Assertions

Oracle Service Bus provides support for the execution of a set of predefined policies and assertion templates that are delivered by OWSM. When a specific functionality is not provided with the standard policies available with the product, users can develop custom assertions.

See "About Custom Assertions" in *Developing Extensible Applications for Oracle Web Services Manager* for more information on custom assertions.

See "Creating Custom Assertions" in *Developing Extensible Applications for Oracle Web Services Manager* for more information on creating custom assertions.

Securing Oracle Service Bus Proxy and Business Services with WS-Policy

This chapter describes how to use WebLogic Server security policies in Service Bus.

Note:

This chapter applies primarily to WLS 9 policies and not Oracle Web Services Manager (OWSM) policies. WLS policies are deprecated in this release, and are replaced by OWSM policies. You can import services with WLS9 policies attached, but you can use only OWSM policies for new services.

To express the message-level security requirements for a proxy service or business service that is a web service, you use the Web Services Policy (WS-Policy) framework.

The following sections describe configuring WS-Policy for proxy services and business services:

- [About Web Services Policy](#)
- [Oracle-Proprietary Security Policy Best Practices](#)
- [Policy Subjects and Effective Policy](#)

About Web Services Policy

Web Services Policy (WS-Policy) is a standards-based framework for defining a web service's constraints and requirements. It expresses constraints and requirements in a collection of XML statements called policies, each of which contains one or more assertions.

In Service Bus, WS-Policy assertions are used to specify a web service's requirements for digital signatures and encryption, along with the security algorithms and authentication mechanisms that it requires.

The WS-Policy framework allows other specifications to declare "policy assertions." These are domain-specific XML elements that appear inside a *<policy>* element. Policy assertions specifications describe the syntax and semantics of these domain-specific assertions.

WS-SecurityPolicy is one example of a domain-specific assertion language. The WS-SecurityPolicy specification defines a set of security policy assertions for use with the WS-Policy framework.

WS-ReliableMessaging is another example of a domain-specific assertion language; it defines assertions for declaring reliable-messaging policy.

Relationship Between WS-Security and WS-Policy

Web Services Security (WS-Security) works in conjunction with the Web Services Policy Framework (WS-Policy), and it is important that you understand what these terms mean and how they relate:

- Web Services Security (WS-Security) is an OASIS standard that defines interoperable mechanisms to incorporate message-level security into SOAP messages. WS-Security determines "how" message-level security is incorporated into SOAP messages.

WS-Security supports message integrity and message confidentiality. It also defines an extensible model for including security tokens in a SOAP envelope and a model for referencing security tokens from within a SOAP envelope. WS-Security allows you to specify which parts of a SOAP message are digitally signed or encrypted.

- The Web Services Policy Framework (WS-Policy) provides a general-purpose model and corresponding syntax to describe and communicate the policies of a web service. WS-Policy is an abstract XML framework. The interesting aspects of a WS-Policy are defined in child elements called policy "assertions."
- WS-SecurityPolicy defines assertions for specifying the security aspects of a WS-Policy. WS-SecurityPolicy determines "what" message-level security is required of SOAP messages.

The policies can determine which operations are secured and which security measures a web services client must apply.

When you configure the WS-Policy of a proxy or business service, if the WS-Policy contains one or more security policy assertions, then the proxy service or business service is considered to be WS-Security enabled.

Abstract and Concrete WS-Policy Statements

For security policy assertions written under the WS-Policy specification (using the proprietary Oracle schema for security policy), the WebLogic Web Services runtime environment recognizes two types of WS-Policy statements:

- **Concrete** WS-Policy statements specify the security tokens that are used for authentication, encryption, and digital signatures. A concrete encryption policy always has the server's encryption certificate embedded in the form of a base-64 encoded certificate in an X.509 binary security token.

You can create concrete WS-Policy statements if you know at design time the type of authentication (such as using X.509 or SAML tokens) that you want to require.

- **Abstract** WS-Policy statements do not specify security tokens. Specifically, this means the <Identity> and <Integrity> elements (or assertions) of the WS-Policy files do not contain a <SupportedTokens><SecurityToken> child element, and the <Confidentiality> element WS-Policy file does not contain a <KeyInfo><SecurityToken> child element.

The Service Bus runtime environment determines which security token types an abstract policy will accept.

Oracle-Proprietary Security Policy Best Practices

This section describes best practices you should follow when using security policy assertions written under the WS-Policy specification, using the proprietary Oracle schema for security policy.

Note:

Carefully analyze your security requirements before you design your WS-SecurityPolicy. These best practices may or may not apply to your specific business security needs.

- Make sure you do not use Identity assertions on an operation's response policy. As a corollary, do not use the predefined `Auth.xml` policy in a response policy.
When using WS-Security user name tokens on inbound to an active intermediary proxy service, if you want to pass the user name/password to a back-end service (user name/password pass-through), the user name token must include the password in clear-text.
- Whenever using WS-Security user name tokens with clear-text passwords, it is strongly recommended that you protect the confidentiality of the user name token, either by encrypting the entire token (with WS-Security) or by sending the message over SSL.
- Whenever using an Identity assertion, you may also want to use an Integrity assertion to digitally sign the authentication token (user name, X.509 or SAML token) together with sensitive message content (SOAP body and/or SOAP header parts). The digital signature protects the integrity of the signed content and binds together the authentication token and message content. This is important to prevent someone from copying the authentication token into an arbitrary SOAP envelope, thus forging a message. (You can also send the message over SSL instead of using an integrity assertion.)
- When using an Integrity assertion, you should also use a MessageAge assertion. You should also include the signing token (that is, the verification certificate) in the `wsse:Security` header and that the digital signature covers the signing token and the timestamp, in addition to whatever SOAP body and/or SOAP header parts you wish to sign. The message age assertion guarantees a timestamp will be included in the security header. The timestamp is used to prevent some replay attacks. The predefined `Sign.xml` policy follows this best practice.
- When using timestamps over JMS (MessageAge assertions), make sure you set the age of the MessageAge assertion appropriately. If the value is too low, the message may expire while on the queue.
- Whenever an Identity assertion includes X.509 tokens in the supported token list, your policy must also have an Integrity assertion. The server will not accept X.509 tokens as proof of authentication unless the token is also used in a digital signature.

If the Identity assertion accepts other token types, you may use the `X509AuthConditional` attribute of the Integrity assertion to specify that the digital signature is required only when the actual authentication token is an X.509 token. Remember that abstract Identity assertions are pre-processed at deploy time and

converted into concrete assertions by inserting a list of all token types supported by your runtime environment.

- Oracle recommends that you do not use abstract Identity assertions in your policy. It is preferable instead to directly specify exactly which token types are supported for authentication. Furthermore, Oracle recommends that your Identity assertion supports only one token type.

Note:

This makes the X509AuthConditional attribute of Integrity assertions unnecessary, as there is no ambiguity as to which token types are supported.

As a corollary, Oracle recommends that you do not use the Auth.xml policy file: use the Sign.xml and Encrypt.xml policies whenever possible.

- Whenever an Service Bus proxy processes digital signatures (on inbound request messages or back-end response messages), it is strongly recommended that you configure a certificate registry in your security realm and import your trading partner certificates in the registry.

Policy Subjects and Effective Policy

A policy subject is an entity, such as service, endpoint, operation, or message, with which a policy can be associated. You can associate a single WS-Policy statement with multiple policy subjects; conversely, multiple WS-Policy statements can be associated with a single policy subject. A policy scope is the collection of policy subjects to which a policy applies. For example, the policy scope implied by a policy attached to `wsd:binding/wsd:operation/wsd:input` is the input message, the operation, the endpoint, and the service.

The effective policy for a given policy subject is the merge of all policies whose scopes contain that policy subject. For example, the effective policy of the input message of a binding operation is the merge of all policies attached to the following:

- The input message of the binding operation
- The binding operation
- The binding
- The input message of the port-type operation
- The port-type operation
- The port-type
- The service

The Oracle Service Bus Console displays the effective policy (read only) when configuring a business or proxy service with WS-Policy statements.

Using SAML with Oracle Service Bus

This chapter describes how to use Security Assertion Markup Language (SAML) policies for exchanging authentication and authorization information between clients and services in Service Bus.

You can use SAML with either the WLS 9 policy framework or with Oracle Web Services Manager. Oracle recommends that you use Oracle Web Services Manager for service security, as described in [Securing Oracle Service Bus with Oracle Web Services Manager](#).

This chapter includes the following topics:

- [Mapping Identity to a SAML Token](#)
- [Configuring SAML Pass-Through Identity Propagation](#)
- [Authenticating SAML Tokens in Proxy Service Requests](#)
- [Configuring SAML Authentication with Service Bus \(SB\) Transport](#)
- [Using SAML Identity Switching](#)
- [Troubleshooting SAML with Oracle Service Bus](#)

Mapping Identity to a SAML Token

If your clients do not provide SAML tokens but your business services require them, you can configure a proxy service to map the client's identity to a SAML token.

To configure SAML credential mapping:

1. Configure a proxy service to authenticate clients using any of the following techniques:
 - HTTP or HTTPS basic (client provides user name and password in the request)
 - HTTPS client certificate
 - Message-level authentication (using any of the supported token profiles)

If a client request includes a WS-Security security header, you must configure the proxy service to process this header on the proxy service side of the message. In Service Bus, you cannot add a SAML header (or any other WS-Security header) to a SOAP envelope that already contains a WS-Security header, neither can you add SAML (or other) security tokens to an existing security header.

- Third-party authentication

When the proxy service authenticates the user, the proxy automatically generates a Subject before forwarding the message to a business service.

2. Configure the business service to include a SAML client policy. The policy generates a SAML token for the authenticated user in the Subject.

For a list of Oracle Web Services Manager SAML policies supported with Service Bus, see [Predefined Policies and Unsupported Assertions](#).

Note:

The procedure in this section assumes a proxy-to-business service invocation. When your use case involves proxy-to-proxy invocations prior to the business service invocation, it is helpful to understand how Service Bus handles security headers. For that information, see [Using OWSM Security with Local Proxy Services](#).

Configuring SAML Pass-Through Identity Propagation

If your clients provide SAML tokens to a pass-through proxy service, you can propagate the client's SAML token to the business service.

This technique requires the business service to be a web service with policy statements that require authentication using SAML tokens.

To configure SAML pass-through identity propagation:

- **Proxy Service** – Configure a pass-through proxy service as described in [Creating a Pass-Through Proxy Service: Main Steps](#).
- **Business Service** – Configure a SOAP-HTTP or SOAP-JMS business service with policy statements that require authentication using SAML tokens, as described in [Configuring Business Service Message-Level Security: Main Steps](#).

Authenticating SAML Tokens in Proxy Service Requests

If your clients provide SAML tokens to an active intermediary proxy service, you can configure the proxy service to assert the client's identity.

To configure a proxy service to use SAML tokens to authenticate clients:

1. When configuring the Identity Assertion provider, note the following requirements:
 - The confirmation method from the policy must match the SAML profile in the SAML asserting party.
 - Specify the asserting party target URL to be the relative URL of the proxy service (omitting the protocol and host information).
 - For signed assertions, add the certificate to the Identity Asserter registry.
2. Create an active intermediary proxy service that communicates over the HTTP, HTTPS, or JMS protocol. The proxy service must be a web service with a policy statement that requires authentication and accepts SAML tokens.

A proxy service that communicates over the "local" transport type cannot use a SAML token profile to authenticate.

Configuring SAML Authentication with Service Bus (SB) Transport

If you are using SAML-based authentication with the SB transport, follow these configuration requirements:

- On the asserting party, configure the SAML Credential mapper with URI `http://openuri.org/sb_proxy_uri`, where `sb_proxy_uri` is the SB transport service URI.
- When configuring the Identity Assertion provider on the Service Bus side (the relying party), use the asserting party target URL as the proxy endpoint URI. Do not include the protocol and host information. For example, `/sb_proxy_uri`.

Using SAML Identity Switching

Oracle Web Services Manager provides a `wss11_saml_token_identity_switch_with_message_protection_client_policy` that lets you perform identity switching. The policy, which you attach to a business service, propagates a different identity than the one based on the authenticated Subject from the proxy service. For more information about the policy, see *Configuring SAML Web Service Clients for Identity Switching in Securing Web Services and Managing Policies with Oracle Web Services Manager*.

If you set the policy property `subject.precedence = false` and provide a credential store format (CSF) key for the identity you want to switch to, the business service ignores the current subject and creates a SAML token with the credentials in the `csf-key`.

If you set `subject.precedence = true`, the current subject is used to create the SAML token. However, if the subject is anonymous, Oracle Web Services Manager attempts to use the `csf-key` to perform the identity switching.

For information on working with CSF, see *Developing with the Credential Store Framework in Securing Applications with Oracle Platform Security Services*.

Protecting the Identity-Switching Resource

To prevent malicious access to the identity-switching functionality, you must grant special permissions to the resources that perform identity switching. For example, in Service Bus, you give permissions to the project containing the identity-switching business service.

Use Fusion Middleware Control to give the Service Bus project the proper permissions. Using that topic for guidance, enter the following information in the permissions fields:

- **Permission Class:** `oracle.wsm.security.WSIdentityPermission`
- **Resource Name:** Name (not the path) of the Service Bus project containing the business service
- **Permission Actions:** `assert`

Troubleshooting SAML with Oracle Service Bus

Question: I am trying to propagate my proxy service transport identity to a destination business service and keep receiving error, Unable to add security token for identity. What does this mean?

Answer: There are various causes for this error. Generally this means one of the following problems:

- The SAML Credential Mapper is not configured correctly. Double check that the configuration is in accordance with the instructions in "Configuring SAML" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Another common source of this error is that there is no subject information to propagate. To generate a SAML token, you must have a transport-level or message-level subject. Make sure that the client has a subject. This can be done by inspecting the `$security` message context variable.

Question: I am trying to propagate my proxy service transport identity to a destination business service using SAML holder-of-key and keep receiving error, *Failure to add signature*. What does this mean?

Answer: There are various causes for this error, but most likely is that the credentials are not configured for the business service's service key provider. When Service Bus generates an outbound holder-of-key assertion, it also generates a digital signature over the message contents, so that the recipient can verify not only that a message is received from a particular user but that the message has not been tampered with. To generate the signature, the business service must have a service key provider with a digital signature credential associated with it.

Question: I am trying to configure an active intermediary proxy service that receives SAML identity tokens and keep receiving errors that look like: *The SAML token is not valid*. How do I fix this?

Answer: This is generally caused by a missing SAML Identity Asserter or SAML Identity Asserter asserting party configuration for the proxy service. For a proxy service to receive SAML assertions in active intermediary mode, it must have a SAML Identity Asserter configured. For more details, see "Configuring a SAML Identity Assertion Provider" in *Administering Security for Oracle WebLogic Server*.

Configuring Custom Authentication

This chapter describes how to configure custom transport- and message-level authentication in Service Bus.

This chapter includes the following sections:

- [Introduction to Custom Authentication in Oracle Service Bus](#)
- [Format of XPath Expressions](#)
- [Configuring Identity Assertion Providers for Custom Tokens](#)
- [Configuring Custom Authentication Transport-Level Security](#)
- [Configuring Message-Level Custom Authentication](#)

For instructions on configuring proxy or business service security, see [Securing Business and Proxy Services](#).

Introduction to Custom Authentication in Oracle Service Bus

Service Bus supports client-specified custom authentication credentials for transport-level proxy and business service requests, and for message-level proxy service requests. The custom authentication credentials can be in the form of tokens, or a user name and password token combination. At the transport level, Service Bus accepts and attempts to authenticate a custom token passed to a proxy or business service in an HTTP header. At the message level, Service Bus can get the information from a SOAP header (for SOAP-based services) or the payload (for non-SOAP services).

You use the Proxy Service Definition Editor to configure a proxy service with the token type and the mechanism by which the token is passed. For business services, you define a custom authentication Java class, which you can then reference from the Business Service Definition Editor.

Note:

The custom authentication mechanisms work alone or in concert with the message-level security for web services described in [Configuring Message-Level Security for Web Services](#). See [Combining WS-Security with Custom User Name/Password and Tokens](#) for information about using both types of security.

Understanding Custom Authentication Tokens

An authentication token is some data, represented as a string or XML, that identifies an entity (user or process), such as an X509 client certificate. Typically, authentication tokens are designed to be used within specific security protocols. Some authentication

tokens are cryptographically protected and some are not. Some authentication tokens carry key material.

In the context of Service Bus, a custom authentication token can be a user name and password or an opaque Identity Assertion token in a user-defined location in the request. A user name and password token is allowed in a SOAP header (for SOAP-based services) or in the payload of some non-SOAP proxy services. An Identity Assertion token is allowed in an HTTP header, in a SOAP header (for SOAP-based services), or in the payload of some non-SOAP proxy services. The Service Bus domain must include an Identity Assertion provider that supports the token type.

Service Bus uses the authenticated user to establish a security context for the client. The security context established by authenticating a custom token or user name and password can be used as the basis for outbound credential mapping and access control. To authenticate and authorize clients who supply tokens for authentication, you must configure an Identity Assertion provider that maps the client's credential to a Service Bus user. Service Bus uses this resulting user name to establish a security context for the client.

Custom Authentication Token Use and Deployment

Custom authentication token support in Service Bus addresses two requirements. For the first requirement, a proxy service request has a user name and password somewhere in the message payload, for example in a SOAP header. Service Bus must get this user name and password and authenticate the user. For the second requirement, the message contains some kind of authentication token other than a user name and password, such as a secure-token-xyz token. The token may be in an HTTP header or in the message payload. Service Bus must get the token and authenticate it. In either case, a security context is established if authentication succeeds.

Most security-related configuration is typically done at deployment time, and custom authentication fits that model. It can be configured directly on the production environment at deployment time. Alternatively, you can configure authentication during staging and import it into the production environment.

Custom authentication, which includes both custom tokens and user name and password tokens, is an integral part of the proxy service definition. When a proxy service is exported, any configuration of custom tokens is included in the JAR file. When a new version of the proxy service is imported, the previous configuration is overwritten with whatever configuration is contained in the JAR file.

Only users in the `MiddlewareAdministrator` or `Developer` application role can configure custom token authentication (or the `IntegrationDeployer` or `IntegrationAdministrator` enterprise role). For information about roles, see "Role-Based Access in Oracle Service Bus" in *Administering Oracle Service Bus*.

Understanding Transport-Level Custom Authentication

You can authenticate client requests at the transport-level using custom authentication tokens, which can be specified in an HTTP header. You configure transport-level custom authentication on the Transport Details page of the service definition editor. The options for HTTP and HTTPS services are:

- None
- Basic
- Custom Authentication

- Client Certificate (HTTPS Only)

These are mutually exclusive options.

If you choose custom authentication for a proxy service, you must also specify the token type and the name of the HTTP header that is to carry the token. If you choose custom authentication for a business service, you must also specify the custom authenticator Java class. The custom authentication token can be any active token type, previously configured for an Identity Assertion provider, that is carried in an HTTP header.

Import/Export and Transport-Level Custom Token Authentication

Transport-level custom authentication tokens are published to the UDDI. The `client-auth` property is present in the `instanceParms` of the HTTP or HTTPS transport attributes whenever authentication is configured. As described in [Transport Attributes](#), the possible values of `client-auth` are `BASIC`, `CLIENT-CERT` and `CUSTOM-TOKEN`. Whenever the value is `CUSTOM-TOKEN`, two additional properties are present: `token-header` and `token-type`.

Understanding Message-Level Custom Authentication

Service Bus supports client-specified custom authentication credentials for proxy service message-level requests. The custom authentication credentials can be in the form of a custom token, or a user name and password. Service Bus accepts and attempts to authenticate a custom token passed to a proxy service in a SOAP header (for SOAP-based proxy services), or in the payload (for non-SOAP proxy services). You configure message-level custom authentication on the Security page (Oracle Service Bus Console) or the Security Settings page (JDeveloper) of the Proxy Service Definition Editor, which includes specifying the token type and the mechanism by which the token is passed.

The following proxy service message-level authentication mechanisms are supported:

- For SOAP-based proxy services
 - Custom token in a SOAP header
 - Username/password in a SOAP header
- For non-SOAP-based proxy services
 - Custom token in the payload of any XML-based proxy services
 - Username/password in the payload of any XML-based proxy services

Message-level custom tokens and message-level user name and password are supported on proxy services of the following binding types:

- WSDL-SOAP
- WSDL-XML
- Abstract SOAP
- Abstract XML
- Mixed – XML (in the request)
- Mixed – MFL (in the request)

Propagating the Identity Obtained From Custom Authentication Tokens

The security context established using a custom token or custom user name/password is in no way unique, and you can use it for credential mapping. If you implement both transport-level authentication and message-level authentication, the message-level security context is always used for credential mapping and identity propagation.

For example, if the proxy service authenticates the client using a secure-token-xyz token in a SOAP header, the authenticated subject is used during any mapped service account lookup. The subject is also used when generating SAML tokens on outbound messages. Java callouts can also run under the authentication context associated with a custom token or custom user name/password.

If a custom user name/password is used, the user name/password in the custom token can be used for outbound HTTP basic or outbound WS-Security Username Token authentication if a pass-through service account is used.

Combining WS-Security with Custom User Name/Password and Tokens

You can secure Service Bus proxy services with transport-level security (for example, HTTPS) or message-level security (for example, WS-Security (WSS) and custom tokens), or a combination of both. For example, client requests can be authenticated at the transport level with custom tokens in HTTP headers, and at the message level with WS-Security tokens, custom tokens, or user name/passwords, except in the WS-Security header.

There is one restriction. Although it is possible to combine WS-Security and message-level custom tokens, the WS-Security policy must not require proxy service authentication based on WS-Security tokens. Message-level custom tokens and WS-Security proxy service authentication are mutually exclusive.

Consider the following examples:

- You cannot configure a proxy service that expects a custom token of type `MyToken` in SOAP header `<soap:MyToken>` and that has a WS-Security policy that requires signing or encryption of some message parts (for example, the `<soap:MyToken>` header and SOAP body).
- It is not allowable to configure a proxy service that requires a custom token in header `<soap:MyToken>` and that also has a WS-Security policy that requires a SAML token or any other form of authentication.

Format of XPath Expressions

The configuration for both custom user name/password and custom token authentication is similar. In both cases, you specify XPath expressions so Service Bus can locate the necessary information. The root of these XPath expressions is as follows:

- Use `soap-env:Envelope/soap-env:Header` if the service binding is anySOAP or WSDL-SOAP.
- Use `soap-env:Body` (specifically, the contents of the `$body` variable) if the service binding is not SOAP based.

Note:

All XPath expressions must be in a valid XPath 2.0 format. The XPath expressions must use the XPath "declare namespace" syntax to declare any namespaces used, as follows:

```
declare namespace ns='http://webservices.mycompany.com/MyExampleService';
```

For example,

```
declare namespace y="http://foo";./y:my-custom-token/text()
```

Configuring Identity Assertion Providers for Custom Tokens

An Identity Assertion provider is a specific form of authentication provider that allows users or system processes to assert their identity using tokens. A client's identity is established through the use of client-supplied tokens. The Identity Assertion provider validates the token. If the token is successfully validated, the Identity Assertion provider maps the token to an Service Bus user name, and returns the user name. Identity is said to be "asserted" when the token is mapped to the user name. Service Bus then uses this user name to establish a security context for the client.

If you want the proxy or business service to consume a custom token, check the provided WebLogic Server Identity Assertion providers to see if one meets your needs. WebLogic Server includes a broad array of Identity Assertion providers, including the following:

- The *WebLogic Identity Assertion provider* validates X.509 and IIOP-CSIV2 tokens and optionally can use a user name mapper to map that token to a user.
- The *SAML Identity Assertion provider*, which acts as a consumer of SAML security assertions.

If you want the service to consume a custom token that is not handled by one of the bundled Identity Assertion providers, for example a `secure-token-xyz` token, you (or a third-party) must first write a WebLogic Server Identity Assertion provider that supports the token type and use the Oracle WebLogic Server Administration Console to add that provider to the security realm.

You develop Identity Assertion providers to support the specific types of custom tokens that you will be using to assert the identities of users. You can develop an Identity Assertion provider to support multiple token types. While you can have multiple Identity Assertion providers in a security realm with the ability to validate the same token type, only one Identity Assertion provider can actually perform this validation.

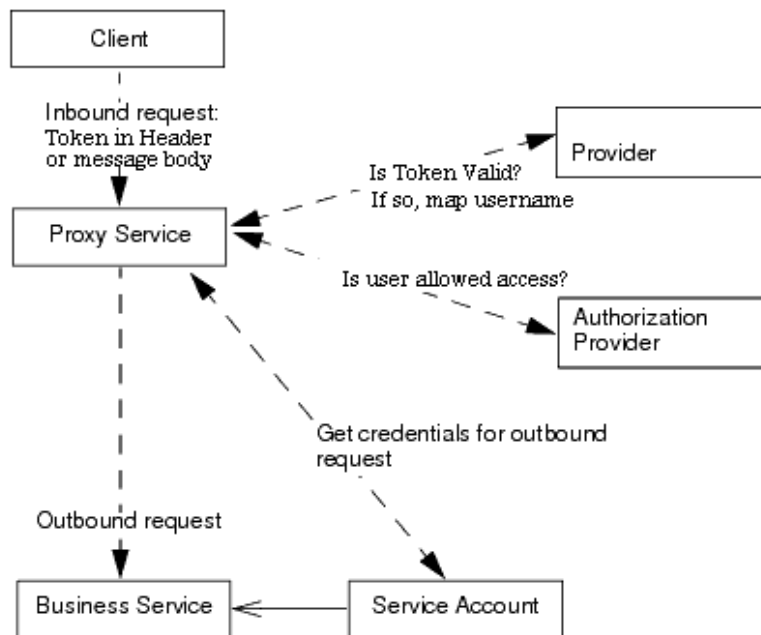
The Identity Assertion process for a proxy service configured for custom authentication is shown in [Figure 55-1](#), and works as follows:

1. The proxy service gets the authentication token from the inbound request.
2. The token is passed to an Identity Assertion provider that is responsible for validating tokens of that type and that is configured as "active."
3. The Identity Assertion provider validates the token.

4. If the token is successfully validated, the Identity Assertion provider maps the token to a user name, and returns the user name.
5. Service Bus then continues the authentication process with this user name and, if successful, obtains the authenticated subject.
6. Service Bus creates the security context. The security context established by authenticating a custom token or user name and password can be used as the basis for outbound credential mapping and access control.

See "Identity Assertion and Tokens" in *Understanding Security for Oracle WebLogic Server* for additional information.

Figure 55-1 Identity Assertion and Custom Tokens in a Proxy Service



Object Type of Custom Tokens

For transport-level identity assertion, the header value is passed as a `java.lang.String` to the Identity Assertion providers. For message-level identity assertion, the XPath expression is evaluated as follows:

- If the XPath expression returns multiple nodes, an error is raised and identity assertion is not called.
- If the XPath expression returns an empty result, identity assertion is called with a null argument.
- If the XPath expression returns a single token of type TEXT or ATTR (See `XmlCursor.TokenType` at <http://xmlbeans.apache.org/docs/2.0.0/reference/org/apache/xmlbeans/XmlCursor.TokenType.html>), the string value of the text node or attribute is passed (as returned by `XmlCursor.getStringValue()`). Otherwise, a single `XmlObject` is passed.

Configuring a Custom Token Type in an Identity Assertion Provider

The steps required to complete these tasks are provided in the following WebLogic Server documents:

- How to Create New Token Types in *Developing Security Providers for Oracle WebLogic Server* describes how to create custom token types for an Identity Assertion provider.
- Configure the Custom Identity Assertion Provider using the Administration Console in *Developing Security Providers for Oracle WebLogic Server* describes how to configure Identity Assertion providers in the Oracle WebLogic Server Administration Console.

How to Configure a Custom Token Type in an Identity Assertion Provider

For your convenience, the steps for creating custom token types for an Identity Assertion provider and configuring that provider in the Oracle WebLogic Server Administration Console are briefly listed here. However, you will need to consult the WebLogic Server documentation to actually complete the tasks.

To configure a custom token type in an Identity Assertion provider:

1. Create the new token types. See "How to Create New Token Types" in *Developing Security Providers for Oracle WebLogic Server*.
2. Create the runtime classes. See "Create Runtime Classes Using the Appropriate SSPIs" in *Developing Security Providers for Oracle WebLogic Server*. That section shows the `SampleIdentityAsserterProviderImpl.java` class, which is the runtime class for the sample Identity Assertion provider.
3. Generate the MBean type. See "Generate an MBean Type Using the WebLogic MBeanMaker" in *Developing Security Providers for Oracle WebLogic Server*.
4. Configure the custom Identity Assertion provider. See "Configure the Custom Identity Assertion Provider Using the Administration Console" in *Developing Security Providers for Oracle WebLogic Server*.
5. Define the active token type. See "Configuring Identity Assertion Providers" in *Administering Security for Oracle WebLogic Server* and "How to Make New Token Types Available for Identity Assertion Provider Configurations" in *Developing Security Providers for Oracle WebLogic Server*.

Setting the Supported and Active Types in the MBean

When you configure a custom Identity Assertion provider, the Supported Types field displays a list of the token types that the Identity Assertion provider supports. You enter zero or more of the supported types in the Active Types field.

The content for the Supported Types field is obtained from the SupportedTypes attribute of the MBean Definition File (MDF), which you use to generate your custom Identity Assertion provider's MBean type. An example from the sample Identity Assertion provider is shown in the following example. For more information about MDFs and MBean types, see "Generate an MBean Type Using the WebLogic MBeanMaker" in *Developing Security Providers for Oracle WebLogic Server*.

Example - SampleIdentityAsserter MDF: SupportedTypes Attribute

```
<MBeanType>
...
<MBeanAttribute
Name = "SupportedTypes"
Type = "java.lang.String[]"
Writeable = "false"
Default = "new String[] {&quot;SamplePerimeterAtnToken&quot;}"
/>
...
</MBeanType>
```

Similarly, the content for the Active Types field is obtained from the ActiveTypes attribute of the MBean Definition File (MDF). You can default the ActiveTypes attribute in the MDF so that it does not have to be set manually with the Oracle WebLogic Server Administration Console. An example from the sample Identity Assertion provider is shown in the following example.

Example - SampleIdentityAsserter MDF: ActiveTypes Attribute with Default

```
<MBeanAttribute
Name= "ActiveTypes"
Type= "java.lang.String[]"
Default = "new String[] { &quot;SamplePerimeterAtnToken&quot; }"
/>
```

While defaulting the ActiveTypes attribute is convenient, you should only do this if no other Identity Assertion provider will ever validate that token type. Otherwise, it would be easy to configure an invalid security realm (where more than one Identity Assertion provider attempts to validate the same token type). Best practice dictates that all MDFs for Identity Assertion providers turn off the token type by default; then an administrator can manually make the token type active by configuring the Identity Assertion provider that validates it.

Configuring Custom Authentication Transport-Level Security

Before you can configure your HTTP proxy or business service for transport-level security, you must configure, or create and configure, an Identity Assertion provider that understands the token type you plan to use.

- [How to Create a Custom Authentication Class for Outbound](#)
- [How to Configure Transport-Level Custom Authentication](#)

How to Create a Custom Authentication Class for Outbound

For outbound custom authentication, you need to create a custom Java class that defines the custom logic. The following interfaces provide the methods you need to implement custom authentication for business services:

- `com.bea.wli.sb.transports.http.OutboundAuthentication`
- `com.bea.wli.sb.transports.http.HttpURLConnectionFactory`

The custom authenticator must implement the `doOutboundAuthentication` method of the `OutboundAuthentication` interface. The custom authenticator can set the authentication headers and read the response from the target service for the initial and the intermediate transactions. For the final step it is expected that the custom authenticator sets only the authentication headers and does not establish the connection with the target service. Service Bus will send the payload along with the

authentication headers in the final step. The custom authenticator should not manipulate the payload.

How to Configure Transport-Level Custom Authentication

The following procedure provides the high-level steps you need to complete to define custom authentication at the transport level. It includes links more detailed instructions.

To configure transport-level custom authentication:

1. Determine which custom token format to use.
2. Determine if an existing provider meets your needs. For guidance, see "Choosing an Authentication Provider" in *Administering Security for Oracle WebLogic Server*.
3. Configure, or create and configure, an Identity Assertion provider that supports the token format. See the following for instructions:
 - To configure an existing Identity Assertion provider, see "Configuring Authentication Providers" in *Administering Security for Oracle WebLogic Server* (specifically, "Configuring Identity Assertion Providers."
 - To create custom token types for a provider, see [How to Configure a Custom Token Type in an Identity Assertion Provider](#).
 - To develop a custom provider, see "Identity Assertion Providers" in *Developing Security Providers for Oracle WebLogic Server*.
4. The Identity Assertion provider maps the token to a user name. Add the client's user name in Fusion Middleware Control.
5. For proxy services, do the following on the Transport Details page of the Proxy Service Definition Editor:
 - a. By Authentication, select **Custom Authentication**.
 - b. In the **Authentication Header** field under Advanced Options (or Advanced Settings), enter the message header where Service Bus can find the token.
 - c. In the **Authentication Token Type**, enter the token type expected by this endpoint.
6. For business services, create and compile the custom authenticator Java class, and add the class to the system classpath.
7. For business services, do the following on the Transport Details page of the Business Service Definition Editor:
 - a. By Authentication, select **Custom Authentication**.
 - b. In the **HTTP Custom Authentication Class Name** field, enter the name of the Java class that defines the custom authentication.
8. Save and activate your changes.

Configuring Message-Level Custom Authentication

Before you can configure custom authentication message-level security for a proxy service, you must first configure, or create and configure, an authentication provider or Identity Assertion provider that understands the token type you plan to use.

How to Configure Message-Level Custom Authentication for Proxy Services

The following procedure provides the high-level steps you need to complete to define custom authentication at the transport level. It includes links more detailed instructions.

Note:

The provider you use must also understand any context properties that you want to provide. If you specify any Context Properties you will probably need to create your own provider because the provider must know which property names to expect.

To configure message-level custom authentication:

1. Determine which custom user name/password or token format to use.
2. Determine if an existing provider meets your needs. For guidance, see "Choosing an Authentication Provider" in *Administering Security for Oracle WebLogic Server*.
3. Configure, or create and configure, an authentication provider or Identity Assertion provider that supports the user name/password or token format, respectively. See the following for instructions:
 - To configure an existing authentication or Identity Assertion provider, see "Configuring Authentication Providers" in *Administering Security for Oracle WebLogic Server*.
 - To create custom token types for a provider, see [How to Configure a Custom Token Type in an Identity Assertion Provider](#).
 - To develop a custom Identity Assertion provider, see "Identity Assertion Providers" in *Developing Security Providers for Oracle WebLogic Server*.
4. Add the client's user name in Fusion Middleware Control.
5. In Service Bus, define the custom token or user name and password in the security settings for the proxy service. Optionally, specify context properties for the provider.

For information and instructions, see [How to Configure Custom Authentication for a Proxy Service in the Console](#) or [How to Configure Custom Authentication for Proxy Services in JDeveloper](#).

Defining Message-Level Security with .Net 2.0

This chapter describes how to configure message-level security between .NET 2.0 and Service Bus.

The chapter includes the following sections:

- [Message-Level Security Between .NET 2.0 and Oracle Service Bus](#)
- [What is .NET?](#)
- [Message-Level Security Configuration in .NET](#)
- [Oracle Service Bus Configuration for Message-Level Security with .NET](#)

Message-Level Security Between .NET 2.0 and Oracle Service Bus

You can set up Message-level security between the Microsoft .NET 2.0 framework and Service Bus. Message-level security applies security checks to a SOAP message after a web services client establishes a connection with an Service Bus proxy service or business service and before the proxy service or business service processes the message.

What is .NET?

The .NET framework is a software component that you can add to the Microsoft Windows operating system. It provides pre-coded solutions to common program requirements, and manages the execution of programs written specifically for the framework.

Message-Level Security Configuration in .NET

This section provides the steps that you need to perform for .NET 2.0 and for Service Bus to configure message-level security.

Caution:

Before you perform these steps, you must follow the steps in [Configuring Message-Level Security for Web Services](#) to configure inbound and outbound messaging for Service Bus.

To configure message-level security between .NET and Service Bus:

1. Verify that you completed the steps to configure inbound and outbound messaging for Service Bus. See the Warning above for instructions.

2. Download Web Service Enhancements (WSE) 3.0 from <http://msdn2.microsoft.com/en-us/webservices> and install it. WSE 3.0 is a SOAP extension managed API (Microsoft.Web.Services3.dll) that is compatible with the .NET 2.0 framework.
3. After you install WSE 3.0, you must enable the WSE features for your web application and enable WSE Soap Protocol Factory support. You can enable both these features using wizards in Visual Studio.

After you enable WSE 3.0, you will notice the following restrictions:

- WSE 3.0 no longer supports WS-Policy and therefore, WS-SecurityPolicy for configuration purposes, as it did in .NET 1.1 and WSE 2.0. WSE 3.0 supports only a proprietary policy configuration using the `wse3policyCache.config` file (or equivalent .NET code) that provides similar features to those in WSE 2.0. One consequence of this is that the WSDL files for the .NET web service no longer contain WS-Policy statements. On the other hand, Service Bus supports a WebLogic Server-proprietary format that is based on the assertions described in the December 18, 2002 version of the Web Services Security Policy Language (WS-SecurityPolicy) specification. In order to consume .NET WSDL files in Service Bus, you must incorporate the equivalent Service Bus proprietary version of WS-Policy in the WSDL file.

The WSDL code sample in [Sample WSDL File](#) shows how to configure WS-Policy for message-level identity propagation, confidentiality, and integrity in Service Bus.

- WSE 3.0 provides policy configuration for a few Turnkey Security Assertions in the `wse3policyCache.config` file, which can be selected with a wizard in Visual Studio. The certificate that maps to providing message-level security (encryption and signing, for example) is `MutualCertificate10`. For details on configuring the `MutualCertificate10` Security Assertion, see <http://msdn2.microsoft.com/en-us/library/aa480581.aspx>.
- The WSE Soap Protocol Factory does not support security with SOAP 1.2. The generated client stubs using the Web Reference option in Visual Studio contain the security-enabled operations only if you select SOAP 1.1. Message-level security interoperability works only with SOAP 1.1.
- As with .NET 1.1 and WSE 2.0, you must disable automatic signing of WS-Addressing headers and timestamps that are configured by default. You must change some of the properties in the `wse3policyCache.config` file, as shown in the following example:

Default Config

```
<protection>
  <request signatureOptions="IncludeAddressing, IncludeTimestamp,
    IncludeSoapBody" encryptBody="true" />
  <response signatureOptions="IncludeAddressing, IncludeTimestamp,
    IncludeSoapBody" encryptBody="true" />
  <fault signatureOptions="IncludeAddressing, IncludeTimestamp,
    IncludeSoapBody" encryptBody="false" />
</protection>
```

Required Config

```
<protection>
  <request signatureOptions="IncludeSoapBody" encryptBody="true" />
```

```

    <response signatureOptions="IncludeAddressing, IncludeTimestamp,
IncludeSoapBody" encryptBody="true"
    />
    <fault signatureOptions="IncludeSoapBody" encryptBody="false" />
</protection>

```

- By default, WSE 3.0 expects the key wrapping algorithm to be OAEP. However, Service Bus uses the RSA15 algorithm. If the configuration remains as OAEP, the following exception appears:

```

Microsoft.Web.Services3.Security.SecurityFault: An
unsupported signature or encryption algorithm was used
System.Exception: WSE3002: The receiver is expecting the
key wrapping algorithm to be http://www.w3.org/2001/04/
xmlenc#rsa-oaep-mgflp, but the incoming message usedhttp://
www.w3.org/2001/04/xmlenc#rsa-1_5. You can change the key
wrapping algorithm by configuring the security token manager.

```

To avoid this error, add the following configuration to the `web.config` file (on the .NET web service) and the `app.config` file (on the .NET client side) under the `<microsoft.web.services3>` `<security>` elements:

```

<binarySecurityTokenManager>
  <add valueType="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-x509-token-profile-1.0#X509v3";
  <keyAlgorithm name="RSA15" />
  </add>
</binarySecurityTokenManager>

```

This configuration forces WSE to use RSA15 instead of OAEP.

- For Username Token Authentication, .NET provides a `usernameForCertificateSecurity` turnkey assertion that secures the communication channel between the client and the service at the message layer using the service's X.509 certificate. However, this certificate depends on the ability to reference `<EncryptedKey>` elements as security tokens, and enables the option for signature confirmation to correlate a response message with the request that prompted it.

An alternative for Username Token Authentication is the `.NET usernameOverTransportSecurity` turnkey assertion, which assumes that communication between the client and service will be secured at the transport layer. This approach is WS-Security compatible and supports message-level authentication over SSL. If you want to combine the `usernameOverTransportSecurity` turnkey assertion with other message-level security mechanisms, such as encryption and signing, you must write custom code in .NET.

Oracle Service Bus Configuration for Message-Level Security with .NET

Before you configure Service Bus, the following conditions must exist:

- A .NET client invokes an Service Bus proxy with a plain text message (for example, message-level security does not exist between the .NET client and the Service Bus proxy).
- Service Bus enforces outbound message-level security on the SOAP request.

Note:

For cases where the .NET client has message-level security enabled, you can use Service Bus as a pass-through proxy.

To configure Service Bus for message-level security with .NET:

1. Change the encryption algorithm from `tripledes-cbc` to `aes256-cbc`:

```
<wssp:EncryptionAlgorithm URI="http://www.w3.org/2001/04/xmlenc#aes256-cbc"/>
```

2. Change the `sign.xml` policy on the WSDL file. This attribute is on the integrity assertion element.

```
<wssp:Integrity SignToken='false' .... >
...
</wssp:Integrity>
```

By default this value is true.

3. The .NET web service expects the WS-Addressing `<wsa:To>` element to contain its own URL. As the .NET client first invokes the Service Bus proxy, the `<wsa:To>` element is originally set to the Service Bus proxy URL. Change this URL to the URL of the .NET web service in the Service Bus proxy message flow, using a Replace action as shown in the following example:

Original URL

```
<wsa:To wsu:Id="To_1mbmRK4w0bo2Dz1z" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd";>http://localhost:7001/SecurityALSBProxy</wsa:To>
```

URL after Replace Action

```
<wsa:To wsu:Id="To_1mbmRK4w0bo2Dz1z" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd";>http://localhost/SimpleSecurity/SecurityService.asmx</wsa:To>
```

If you do not change this URL, the following error appears:

```
Microsoft.Web.Services3.Addressing.AddressingFault: Destination Unreachable
System.Exception: WSE846: The header must match the actor URI value of the web
service. The actor URI value can be explicitly specified using
SoapActorAttribute on the ASMX class. In the absence of the attribute, the
actor URI is assumed to be equal to the HTTP Request Url of the incoming
message. The header received contained
"http://localhost:7001/SecurityALSBProxy"; while the receiver is expecting
"http://localhost/SimpleSecurity/SecurityService.asmx";
```

4. The .NET client includes its own Timestamp elements to the SOAP header. Service Bus adds an additional Timestamp header that results in the following error:

```
Microsoft.Web.Services3.Security.SecurityFault: An error was discovered
processing the header Microsoft.Web.Services3.Security.Security
FormatException: WSE001: The input was not a valid Security element
because it contains more than one Timestamp child element.
```

To solve this issue, use a Delete action to remove the original Timestamp elements that the .NET client adds in the message flow.

5. Add the CertificateRegistry certification path provider. You add this from the WLS Administration Console from **realm > Providers > Certification Path > New** and then select CertificateRegistry from the list of options.

Activate the change and restart the server.

After you restart the server, edit the CertificateRegistry provider you just created. From the Management tab add the following three certificates:

- The public certificate of Service Bus
- The public certificate of .NET
- The root agency (issuer of these certificates)

Note:

One way to add the certificates is to import them from a jks store using the Migration tab. Provide the actual path of the identity store.

6. On the Configuration (Common) tab for the CertificateRegistry provider, select Current Builder to make it the current builder.

Save these changes. Then, activate and restart the server.

7. The WLS keystore requires these same certificates:

- The public certificate of Service Bus
- The public certificate of .NET
- The root agency (issuer of these certificates)

You configure the identity and trust keystores for a WebLogic Server instance on the server Configuration: Keystores page. To do this, see *Configure Identity and Trust* in the WebLogic Server online help.

Sample WSDL File

The sample WSDL file in this section shows how to configure WS-Policy for message-level identity propagation, confidentiality, and integrity in Service Bus.

Example - Configuring WS-Policy for Message-Level Security

```
<?xml version='1.0' encoding='UTF-8'?>
<definitions name="SecureHello WorldServiceDefinitions" targetNamespace=
  "http://www.bea.com"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:s0="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd"
  xmlns:s1="http://www.bea.com"
  xmlns:s2="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:UsingPolicy xmlns:nl="http://schemas.xmlsoap.org/wsdl/"
    nl:Required="true"/>
  <wsp:Policy s0:Id="Encrypt.xml">
    <wssp:Confidentiality xmlns:wssp="http://www.bea.com/wls90/
      security/policy">
      <wssp:KeyWrappingAlgorithm URI="http://www.w3.org/2001/04/
        xmlenc#rsa-1_5"/>
    </wssp:Confidentiality>
  </wsp:Policy>
</definitions>
```

```

<wssp:Target>
  <wssp:EncryptionAlgorithm URI="http://www.w3.org/2001/
    04/xmlenc#aes256-cbc"/>
  <wssp:MessageParts Dialect="http://schemas.xmlsoap.org
    /2002/12/wsse#part">wssp:Body()
  </wssp:MessageParts>
</wssp:Target>
<wssp:KeyInfo>
  <wssp:SecurityToken TokenType="http://docs.oasis-open.
    org/wss/2004/01/oasis-200401-wss-x509-token-
    profile-1.0#X509v3"/>
  <wssp:SecurityTokenReference>
    <wssp:Embedded>
      <wsse:BinarySecurityToken EncodingType="http:
        //docs.oasis-open.org/wss/2004/
        01/oasis-200401-wss-soap-message
        -security-1.0#Base64Binary"
        ValueType="http://docs.oasis-open.org/
        wss/2004/01/oasis-200401-wss-x509
        -token-profile-1.0#X509v3"
        xmlns:wsse="http://docs.oasis-open.org/
        wss/2004/01/oasis-200401-wss-wssecurity-
        secext-1.0.xsd">MIIB7DCCAZYCEN+FHOMYRZU
        YPLiIutc01IiwDQYJKoZIhvcNAQEEBQAwTELMak
        GA1UEBhMCVVMxEDA0BgNVBAGTB015U3RhdGUxZDZa
        NBgNVBACtBk15VG93bjEXMBUGA1UEChMOTXlPcmd
        hbml6YXRpb24xGTAXBgNVBAstEEZPUiBURVNUSU5
        HIE9OTFkxEzARBgNVBAMTCkN1cnRHZW5DUiWWhc
        NMDYwNjA3MDQ0MDM2WhcNMjEwNjA4MDQ0MDM2WjB
        6MQswCQYDVQQGEwJVUzEQMA4GA1UECBYHTXlTdGF
        0ZTEPMA0GA1UEBxYGTXlUb3duMRcwFQYDVQQKFg5
        NeU9yZ2FuaXphdGlvbWJlZmBcGA1UECXYQRk9SIFR
        FU1RJTkcgT05MWTEUMBIGA1UEAxYLYmFuZ3BsdHc
        zazIwXDANBgkqhkiG9w0BAQEFAANLADBIakEAXv2
        nWByAF2Xr9wrb06ydrccqPt2VQa0xcwfdZz6oG1j
        1TXq+G5/Q82v7CdxjyWUQBuaZduQx9wFCrAe/aWV
        pgQIDAQABMA0GCSqGSIb3DQEBAUAA0EARBwfl8w
        X915jL5reY+isriNF0EFUs5ck53WRNowiapJx2ea
        ZE03quksJgeJ0z0Hekkr/aTQnkMV1xIt1HxMKRw=
        =</wsse:BinarySecurityToken>
    </wssp:Embedded>
  </wssp:SecurityTokenReference>
</wssp:KeyInfo>
<wssp:Confidentiality>
</wssp:Policy>
<wssp:Policy s0:Id="Auth.xml">
  <wssp:Identity xmlns:wssp="http://www.bea.com/wls90/security/
    policy">
    <wssp:SupportedTokens>
      <wssp:SecurityToken TokenType="http://docs.oasis-open.
        org/wss/2004/01/oasis-200401-wss-username-token
        -profile-1.0#UsernameToken">
      <wssp:UsePassword Type="http://docs.oasis-open.
        org/wss/2004/01/oasis-200401-wss-username
        -token-profile-1.0#PasswordText"/>
      </wssp:SecurityToken>
    </wssp:SupportedTokens>
  </wssp:Identity>
</wssp:Policy>
<wssp:Policy s0:Id="Sign.xml">
  <wssp:Integrity SignToken='false' xmlns:wls="http://www.bea.com/wls90/

```


security/

```

policy/wsee#part" xmlns:wssp="http://www.bea.com/wls90/
security/policy" xmlns:wsu="http://docs.oasis-open.org/wss
/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
<wssp:SignatureAlgorithm URI="http://www.w3.org/2000/09/
xmldsig#rsa-shal"/>
<wssp:CanonicalizationAlgorithm URI="http://www.w3.org/
2001/10/xml-exc-c14n#"/>
<wssp:Target>
  <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09
/xmldsig#shal"/>
  <wssp:MessageParts Dialect="http://www.bea.com/wls90/
security/policy/wsee#part">
    wls:SystemHeaders()
  </wssp:MessageParts>
</wssp:Target>
<wssp:Target>
  <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09
/xmldsig#shal"/>
  <wssp:MessageParts Dialect="http://www.bea.com/wls90/
security/policy/wsee#part">
    wls:SecurityHeader(wsu:Timestamp)
  </wssp:MessageParts>
</wssp:Target>
<wssp:Target>
  <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/
xmldsig#shal"/>
  <wssp:MessageParts Dialect="http://schemas.xmlsoap.
org/2002/12/wsse#part">
wsp:Body()
</wssp:MessageParts>
</wssp:Target>
<wssp:SupportedTokens>
  <wssp:SecurityToken IncludeInMessage="true" TokenType=
"http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-x509-token-profile-1.0#X509v3">
  <wssp:TokenIssuer>CN=CACERT,OU=FOR TESTING ONLY,
O=MyOrganization,L=MyTown,ST=MyState,C=US,1.2.
840.113549.1.9.1=#160f737570706f7274406265612e636
f6d,CN=Demo Certificate Authority Constraints,OU=
Security,O=BEA WebLogic,L=San Francisco,ST=
California,C=US,1.2.840.113549.1.9.1=#16107365637
572697479406265612e636f6d,CN=Demo Certificate
Authority Constraints,OU=Security,O=BEA WebLogic,
L=San Francisco,ST=California,C=US,CN=CertGenCAB,
OU=FOR TESTING ONLY,O=MyOrganization,L=MyTown,ST=
MyState,C=US,CN=Equifax Secure eBusiness CA-1,O=
Equifax Secure Inc.,C=US,CN=VeriSign Class 1
Public Primary Certification Authority - G3,OU=
(c)1999 VeriSign\, Inc. - For authorized use only,
OU=VeriSign Trust Network,O=VeriSign\, Inc.,C=US,
OU=VeriSign Trust Network,OU=(c) 1998 VeriSign\,
Inc. - For authorized use only,OU=Class 2 Public
Primary Certification Authority - G2,O=VeriSign\,
Inc.,C=US,CN=VeriSign Class 3 Public Primary
Certification Authority - G3,OU=(c) 1999
VeriSign\,Inc. - For authorized use only,OU=
VeriSign Trust Network,O=VeriSign\,Inc.,C=US,CN=
Entrust.net Client Certification Authority,OU=(c)
2000 Entrust.net Limited,OU=www.entrust.net/
GCCA_CPS incorp. by ref. (limits liab.),O=Entrust

```

```
.net,OU=Go Daddy Class 2 Certification Authority,
O=The Go Daddy Group\, Inc.,C=US,CN=GTE Cyber
Trust Global Root,OU=GTE CyberTrust Solutions\,
Inc., O=GTE Corporation,C=US,CN=Entrust.net
Secure Server Certification Authority,OU=(c) 2000
Entrust.net Limited,OU=www.entrust.net/SSL_CPS
incorp. by ref. (limits liab.),O=Entrust.net,OU=
Class 1 Public Primary Certification Authority,
O=VeriSign\, Inc.,C=US,1.2.840.113549.1.9.1=#161
9706572736f6e616c2d6261736963407468617774652e636
f6d,CN=Thawte Personal Basic CA,OU=Certification
Services Division,O=Thawte Consulting,L=Cape
Town, ST=Western Cape,C=ZA,OU=VeriSign Trust
Network, OU=(c) 1998 VeriSign\, Inc. - For
authorized use only,OU=Class 1 Public Primary
Certification Authority - G2,O=VeriSign\, Inc.,
C=US,CN=Entrust.net Secure Server Certification
Authority,OU=(c) 1999 Entrust.net Limited,OU=
www.entrust.net/CPS incorp. by ref.(limits iab.),
O=Entrust.net,C=US, 1.2.840.113549.1.9.1=#161c706
572736f6e616c2d667265656d61696c407468617774652e63
6f6d,CN=Thawte Personal Freemail CA,OU=
Certification Services Div,O=Thawte Consulting, L
=Cape Town,ST=Western Cape,C=ZA,OU=Class 3 Public
Primary Certification Authority,O=VeriSign\, Inc.
C=US,CN=GTE CyberTrust Root,O=GTE Corporation,C=
US,CN=VeriSign Class 2 Public Primary Certificate
Authority - G3,OU=(c) 1999 VeriSign\, Inc. - For
authorized use only,OU=VeriSign Trust Network,O=
VeriSign\,Inc.,C=US,1.2.840.113549.1.9.1=#1617736
5727665722d6365727473407468617774652e636f6d,CN=
Thawte Server CA,OU=Certification Services
Division,O=Thawte Consulting cc,L=Cape Town,ST=
Western Cape,C=ZA,OU=Equifax Secure Certificate
Authority,O=Equifax,C=US,1.2.840.113549.1.9.1=#16
1b706572736f6e616c2d7072656d69756d407468617774652
e636f6d,CN=Thawte Personal Premium CA,OU=
Certification Services Division,O=Thawte
Consulting,L=Cape Town,ST=Western Cape,C=ZA,1.2.
840.113549.1.9.1=#16197072656d69756d2d73657276657
2407468617774652e636f6d,CN=Thawte Premium Server
CA,OU=Certification Services Division,O=Thawte
Consulting cc,L=Cape Town,ST=Western Cape,C=ZA,
OU=VeriSign Trust Network,OU=(c) 1998 VeriSign\,
Inc. - For authorized use only,OU=Class 3 Public
Primary Certification Authority - G2,O=VeriSign\,
Inc.,C=US,CN=Entrust.net Certification Authority
(2048),OU=(c) 1999 Entrust.net Limited,OU=www
.entrust.net/CPS_2048 incorp. by ref. (limits
liab.),O=Entrust.net,1.2.840.113549.1.9.1=#1611
696e666f4076616c69636572742e636f6d,CN=http://www.
valicert.com/,OU=ValiCert Class 2 Policy
Validation Authority,O=ValiCert\, Inc.,L=Vali
cert Validation Network,CN=Baltimore CyberTrust
Root, OU=CyberTrust,O=Baltimore,C=IE,OU=Secure
Server Certification Authority,O=RSA Data
Security\, Inc.,C=US,CN=Entrust.net Client
Cert Authority,OU=(c) 1999 Entrust.net Limited,
OU=www.entrust.net/Client_CA_Info/CPS incorp. by
ref. limits liab.,O=Entrust.net,C=US,CN=GeoTrust
Global CA,O=GeoTrust Inc.,C=US,CN=GTE CyberTrust
```

```

        Root 5,OU=GTE CyberTrust Solutions\, Inc.,O=GTE
        Corporation,C=US,OU=Starfield Class 2
        Certification Authority,O=Starfield
        Technologies\, Inc.,C=US,CN=Equifax Secure
        Global eBusiness CA-1,O=Equifax Secure Inc.,C=US,
        CN=Baltimore CyberTrust Code Signing Root,OU=
        CyberTrust,O=Baltimore,C=IE,OU=Class 2 Public
        Primary Certification Authority,O=VeriSign\,
        Inc.,C=US,OU=Equifax Secure eBusiness CA-2,O=
        Equifax Secure,C=US,</wssp:TokenIssuer>
    </wssp:SecurityToken>
</wssp:SupportedTokens>
</wssp:Integrity>
<wssp:MessageAge Age="60" xmlns:wssp="http://www.bea.com/wls90/
security/policy"/>
</wsp:Policy>
<types>
    <xs:schema attributeFormDefault="unqualified" elementFormDefault=
    "qualified" targetNamespace="http://www.bea.com" xmlns:s0="
    http://www.bea.com" xmlns:s1="http://schemas.xmlsoap.org
    /wsdl/soap/" xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/
    09/policy" xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="sayHello">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="s" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="sayHelloResponse">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="return" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:schema>
</types>
<message name="sayHello">
    <part element="s1:sayHello" name="parameters"/>
</message>
<message name="sayHelloResponse">
    <part element="s1:sayHelloResponse" name="parameters"/>
</message>
<portType name="SecureHelloWorldPortType" wsp:PolicyURIs="#Sign.xml
#Auth.xml #Encrypt.xml">
    <operation name="sayHello" parameterOrder="parameters">
        <input message="s1:sayHello"/>
        <output message="s1:sayHelloResponse"/>
    </operation>
</portType>
<binding name="s1:SecureHelloWorldServiceSoapBinding" type="s1:
SecureHelloWorldPortType">
    <s2:binding style="document" transport="http://schemas.
xmlsoap.org/soap/http"/>
    <operation name="sayHello">
        <s2:operation soapAction="" style="document"/>
        <input>
            <s2:body parts="parameters" use="literal"/>
        </input>
        <output>

```

```
        <s2:body parts="parameters" use="literal"/>
    </output>
</operation>
</binding>
<service name="SecureHelloWorldService">
    <port binding="s1:SecureHelloWorldServiceSoapBinding"
        name="SecureHelloWorldServicePort">
        <s2:address location="http://localhost:9111/
            SecureHelloWorldService/SecureHelloWorld
            Service"/>
    </port>
</service>
</definitions>
```

Part IX

Completing Oracle Service Bus Services

This part describes how to use debugging and testing tools to test the Service Bus services you create, and to then deploy those services.

This part contains the following chapters:

- [Debugging Oracle Service Bus Applications](#)
- [Using the Test Console](#)
- [Deploying Oracle Service Bus Services](#)
- [Using the Oracle Service Bus Development Maven Plug-In](#)

Debugging Oracle Service Bus Applications

This chapter describes how to debug Service Bus pipelines and split-joins by setting breakpoints in the message flow and using the JDeveloper debugger. It includes the following sections:

- [Introduction to the Debugger](#)
- [Configuring the Project and Debugger](#)
- [Accessing the Debugger](#)
- [Debugging a Service Bus Application](#)
- [Working with the Debugger Windows](#)

Introduction to the Debugger

JDeveloper provides a comprehensive integration debugger to help you assess and solidify your Service Bus project components. The debugger reduces the development cycle by providing troubleshooting capability directly in the development environment. This means you do not need to build a Service Bus application in JDeveloper, run it, and then return to JDeveloper to fix any issues and repeat these steps. Instead, you can set breakpoints directly in JDeveloper for troubleshooting on pipelines and split-joins.

The debugger can handle Java callouts and supports multi-threaded debugging on split-joins that use parallel processing. Note the following guidelines when using the debugger:

- Debugging is limited to design view in JDeveloper.
- You cannot debug cross-language features, such as a Java callout action, XSLT and XQuery transformations, and so on.
- Only one client at a time can connect to the debugger.
- You can only debug if the server is in development mode.
- The debugger cannot be enabled in production mode or when the server is part of a cluster or an Administration Server plus one or more Managed Servers in a non-clustered domain.

For general information about the debugger windows, see "Running and Debugging Java Projects" in *Developing Applications with Oracle JDeveloper*.

Debug Servers

You can debug the Service Bus components that are deployed on an integrated or standalone WebLogic Server. JDeveloper can manage the lifecycle for a local

integrated WebLogic server instance. When you configure a local integrated server, you can select **Let JDeveloper Manage the lifecycle for this server instance**. With this configuration, JDeveloper starts the server when you select the Run or Debug command. If the integrated server is not configured to be managed by JDeveloper, you need to manually start and stop the server; however you can still use the Run command to deploy the service to be tested to the server. If the integrated server is remote, do *not* select **Let JDeveloper Manage the lifecycle for this server instance**.

When you use a standalone server for debugging, the configuration is the same whether it is local or remote. You do not need to configure the server in the application properties. With a standalone server, you cannot use the Run command from the Application Navigator. Instead, you need to manually deploy the Service Bus application to the server using the Deploy command, and you need to start the test console manually.

Local and Remote Debugging

Debugging can be performed on either a local or remote WebLogic server. A local debugging session is started by setting breakpoints in source files, and then starting the debugger. When debugging a Service Bus component, you have complete control over the execution flow and can view and modify values of variables.

Remote debugging requires two JDeveloper processes: a debugger and a debugee. The debugee is a running server that may or may not be defined in JDeveloper and may reside on a different platform. A Service Bus application must be deployed on the debugee server. In order to perform remote debugging, you must configure a run configuration in the project properties, as described in [How to Create Run Configuration for Remote Debugging](#).

Debugging With Breakpoints

A breakpoint marks a point in a pipeline or split-join where message processing pauses. This lets you examine the values of some or all of the message variables. By setting breakpoints in potential problem areas of your message flow, you can run data through a message flow until it reaches a location you want to debug. When a breakpoint is encountered, message processing pauses and the debugger focuses on the action containing the breakpoint in the source editor. You can then use the debugger to view the state of your program. Breakpoints are flexible in that they can be set before you begin debugging or at any time while you are debugging.

Breakpoints can be added to the following nodes on a pipeline:

- Route node
- Route action
- Branch node
- Pipeline pair path (request or response)
- Stage (both in pipeline pairs and error handlers)
- Stage action (both in pipeline pairs and error handlers)

You can add breakpoints to all split-join actions, but toggling a breakpoint on an If node toggles the breakpoint for the If condition within the If node and not the If node itself.

JDeveloper Debugging Windows

JDeveloper provides several different debugging windows with the debugger, where you can view and analyze data as it moves through the debugging process. Service Bus utilizes the following debugging windows:

- Breakpoints
- Data
- Watches
- Stack
- Threads
- Log

For more information about these windows, see "Using the Debugger Windows" in *Developing Applications with Oracle JDeveloper*.

Configuring the Project and Debugger

You can configure project and debugger settings to control the debugging process. Settings that control the way programs are debugged or run are defined in run configurations. These settings include such things as the target, launch options, and the behavior of the debugger, logger, and profiler. Service Bus provides a default run configuration for local debugging and testing, but you can create new configurations. A project may have several run configurations, each set up for a specific facet of the project or phase of the development process. A run configuration can be bound to the project and be available to all who work on the project, or it can be custom configuration, for your use only.

These steps are optional, and you can perform local debugging using the default configuration with the integrated server. For more information and instructions, see the following topics in *Developing Applications with Oracle JDeveloper*:

- How to Configure a Project for Debugging
- How to Set the Debugger Start Options
- Configuring a Project for Running

How to Create Run Configuration for Remote Debugging

You create a new run configuration by copying an existing configuration, such as the Default configuration provided with Service Bus. Then you modify the settings for the new configuration

To create a run configuration for remote debugging:

Tip:

For more information about any of the windows and fields you work with on the run configuration windows, click **Help**.

1. From the main menu click **Application** and select **Project Properties**.

2. In the navigation pane, select **Run/Debug**, and then click **New** on the Run/Debug page.
3. On the Create Run Configuration dialog, enter a name for the new configuration and select an existing run configuration to copy the initial properties from.
4. Click **OK**.
5. In the Run Configurations section of the Run/Debug page, select the configuration you just created and then click **Edit**.

The Edit Run Configuration window appears with the Launch Settings page displayed.

6. Select **Remote Debugging**, and modify any other fields as needed.
7. In the navigation pane, select **Remote** under **Tool Settings > Debugger**.
8. In the **Protocol** field, select **Attach to Service Bus**.
9. Do one or both of the following:
 - To connect a specific server automatically when you select this run configuration, enter the host name, port number, and a timeout value.
 - To display a dialog that lets you enter host information when you select this run configuration, select **Show Dialog Box Before Connecting Debugger**.

If you perform both of the above steps, the dialog appears when you select the run configuration, and is already populated with the connection information you specified above.

10. Click **OK**, and then click **OK** again.
11. Click **Save**.

How to Choose a Run Configuration for Debugging

A default run configuration is created for each new project, and it uses the integrated WebLogic server for local debugging. You can select this default or any other configuration you have created to run a selected project. If you run the debugger from the JDeveloper toolbar, you can select the run configuration from the list of configuration options available next to the Debug icon.

To choose a run configuration for debugging:

1. From the main menu click **Application** and select **Project Properties**.
2. Select **Run/Debug**.
3. In the Run Configurations list, select the run configuration you want to use, and then click **OK**.

Accessing the Debugger

There are several ways to start the JDeveloper debugger. The instructions in this chapter use the Application Navigator to start the debugger, but you can use any of the following methods to start the debugger when you are ready.

- Right-click a project or component in the application and select **Debug**.
- In the JDeveloper toolbar, click the **Debug** icon. The debug process uses the selected run configuration. See [How to Choose a Run Configuration for Debugging](#).
- Right-click a pipeline or split-join in the Service Bus Composite Overview Editor and select **Debug**.
- Right-click in the editor of an open pipeline or split-join and select **Debug**.

Debugging a Service Bus Application

This section describes how to start the debugger, create breakpoints, and debug Service Bus applications in JDeveloper. There are ways to debug other than using the Test Console and debugger. You can also run your service in other ways, such as posting a JMS message or placing an input file in the directory of a file proxy service.

- [How to Set Breakpoints on Service Bus Components](#)
- [How to Debug Using Breakpoints](#)
- [How to Step Through a Debugging Session](#)
- [How to End or Detach from Debugging](#)

How to Set Breakpoints on Service Bus Components

Breakpoints are the intentional pausing locations in a Service Bus application that you set for debugging purposes. When you run test data using the Test Console, the process pauses at each breakpoint and does not resume until you tell it to. You can set breakpoints on pipelines and split-joins.

To set breakpoints on Service Bus components:

1. Open the pipeline or split-join you want to debug in its editor.
2. Expand the actions until you see the node to which you want to add a breakpoint.
3. Right-click the node, and select **Toggle Breakpoint**.

A red icon appears next to the node to indicate that a breakpoint is set.

4. Repeat the above step for each node to which you want to add a breakpoint.
5. To disable a breakpoint, right-click the node and select **Disable Breakpoint**.

You can also enable, disable, and remove breakpoints from the Breakpoints view, as described in [How to Remove or Disable Breakpoints](#) and [How to Enable a Disabled Breakpoint](#).

6. To remove a breakpoint, right-click the node and select **Toggle Breakpoint** again.
7. Begin debugging, as described in [How to Debug Using Breakpoints](#).

How to Debug Using Breakpoints

When you debug a component using a local server, the Test Console is launched so you can enter the input for the debugging process. You can only test locally when using the integrated WebLogic server. You specify whether to test locally or remotely

in the project's properties. You can also configure how breakpoints are handled during debugging.

For more information about setting these options, see "How to Configure a Project for Debugging" and "How to Set the Debugger Start Options" in *Developing Applications with Oracle JDeveloper*.

To initiate debugging on components with breakpoints:

1. Define the breakpoints for the component to debug, as described in [How to Set Breakpoints on Service Bus Components](#).
2. Right-click the pipeline or split-join in the Application Navigator, and select **Debug**.

The Test Console appears.

Note:

If you do not have a domain running, the Create Default Domain dialog appears. Enter the connection information for the integrated server and click **OK**. This may take several minutes.

3. Enter the test data in the Request Document section, and configure any additional input.

For more information, see [Using the Test Console](#).

4. Click **Execute** on the Test Console.

The Test Console executes the command, but pauses when it reaches the first breakpoint and returns to the editor for the pipeline or split-join. The breakpoint icon is now blue instead of red to indicate where the debugger is stopped.

5. In the Log window at the bottom of JDeveloper, click **Data**.

In this window, you can view variable values at the current state in processing. For information about what you can do here, see [Working with the Debugger Windows](#).

6. To continue working through the message processing, perform the steps under [How to Step Through a Debugging Session](#).

How to Step Through a Debugging Session

The debugging framework lets you debug incrementally by performing step actions to debug code. You can step over parts of the message flow and begin debugging at a different location, such as a different breakpoint in the same or a different component. As you proceed with debugging, additional frames are created for breakpoints and are displayed in the Stack view.

All of the icons mentioned in the following steps are located in the toolbar to the right of the **Debug** icon. Hover over an icon to see its name.

To step through a debugging session:

1. Begin debugging the pipeline or split-join as described in [How to Debug Using Breakpoints](#).

When the debugger reaches a breakpoint, you can begin to step through.

2. To find the action on which the current breakpoint is stopped, click the **Find Execution Point** icon.

The current breakpoint icon is blue instead of red.

3. To step over a breakpoint and move to the next action, click the **Step Over** icon.
4. To step into the next valid location in the message flow, click the **Step Into** icon.
5. To step out of a breakpoint frame, click the **Step Out** icon.
6. If the debugger is paused on a breakpoint, click the **Resume** icon to resume processing. The process runs until the next breakpoint is reached.

How to End or Detach from Debugging

To stop debugging, you can either detach or terminate the debugger.

To end or detach from a debugging session:

1. Click the **Terminate** icon in the tools menu.

The Terminate Debugger Process dialog appears.

2. Select **Detach** or **Terminate**.

Both options terminate the debugger connection to the server and the process continues to run on the server until it completes.

3. If you selected **Detach**, click the debugger icon in the tools menu to resume debugging.

Working with the Debugger Windows

When you start a debugging session, several tabs appear in the Log window that provide you with additional information and features to help you with the debugging process. You can also access these windows from the **Window > Debugger** menu.

How to Edit Breakpoint Options

The Breakpoints window lets you add breakpoints to a group and enable or disable all the breakpoints in a group. You can also configure logging, sounds, and whether reaching a breakpoint halts processing. The Breakpoints window is available whether you are inside or outside a debugging session.

To view and modify the options of a breakpoint:

1. If the Breakpoints window is not open, select **Window > Breakpoints** from the main menu.
2. Right-click a breakpoint and select **Edit**, or select the breakpoint and click the **Edit** icon on the Breakpoint toolbar.

The Edit Breakpoint dialog appears with a Definition tab, a Conditions tab, and an Actions tab. For Service Bus, there is nothing to modify on the Conditions tab; conditional breakpoints are not supported.

3. Make any necessary changes to the breakpoint options.

For more information about the Definition and Actions tabs, click **Help**.

4. To save the changes, click **OK**.

How to Create a Breakpoint Group

Creating breakpoint groups lets you perform bulk edits to breakpoints. When you create a group, a new node is added to the Breakpoint window and any breakpoints added to the group appear under that group node.

Creating a Breakpoint Group

To create a breakpoint group:

1. If the Breakpoints window is not open, select **Window > Breakpoints** from the main menu.
2. Right-click a breakpoint and select **Edit**, or select the breakpoint and click the **Edit** icon on the Breakpoint toolbar.

The Edit Breakpoint dialog appears with the Definition tab displayed.

3. In the **Breakpoint Group Name** field, enter a new name for the group.
4. To save the changes, click **OK**.

The new group is added to the Breakpoint window, and the breakpoint is added to the group.

Adding a Breakpoint to an Existing Group

To add a breakpoint to an existing group:

1. If the Breakpoints window is not open, select **Window > Breakpoints** from the main menu.
2. Right-click a breakpoint and select **Edit**, or select the breakpoint and click the **Edit** icon on the Breakpoint toolbar.

The Edit Breakpoint dialog appears with the Definition tab displayed.

3. In the **Breakpoint Group Name** field, select the name of the group to which you want to add the breakpoint.
4. To save the changes, click **OK**.

How to Remove or Disable Breakpoints

You can disable or remove individual breakpoints or all breakpoints.

To remove or disable breakpoints:

1. To remove an individual breakpoint, right-click the action associated with the breakpoint in the pipeline or split-join editor and select **Toggle Breakpoint**. You can also select the breakpoint in the Breakpoint window, and click **Delete**.
2. To remove all breakpoints, right-click in the Breakpoints window, and select **Delete All**.

3. To disable an individual breakpoint, right-click the action associated with the breakpoint in the pipeline or split-join editor and select **Disable Breakpoint**. You can also select the breakpoint in the Breakpoint window, and click **Disable**.
4. To disable all breakpoints, right-click in the Breakpoints window, and select **Disable All**.

How to Enable a Disabled Breakpoint

If you disable a breakpoint, you can enable it again to include it back in your debug process.

To enable a breakpoint:

1. To enable an individual breakpoint, right-click the action associated with the breakpoint in the pipeline or split-join editor and select **Enable Breakpoint**. You can also select the breakpoint in the Breakpoint window, and click **Enable**.
2. To enable all breakpoints that have been set, right-click in the Breakpoints window, and select **Enable All**.

How to View and Modify Variable Values at the Current Breakpoint

The Data window displays the context variables and their values for the current breakpoint. You can view request or response data throughout the debugging process, and modify their values for further debugging. Only simple-type variables values can be modified.

To view and modify variable values:

1. Begin debugging and when the debugger reaches a breakpoint, click the **Data** tab in the Log window.
2. Expand the variables in the Name column to view the values of each variable and node.
3. To modify a variable value, double-click the value and enter the new value in the dialog that appears.
4. To view the entire string that defines the variable value, right-click the variable and select **View Whole Value**.

How to Add a Watch

A watch lets you monitor the changing values of variables or expressions as your program runs. After you enter a watch expression, the Watches window displays the current value of the expression. As your program runs, the value of the watch changes as your program updates the values of the variables in the watch expression.

A watch evaluates an expression according to the current context which is controlled by the selection in the Stack window. If you move to a new context, the expression is re-evaluated for the new context. If the execution point moves to a location where any of the variables in the watch expression are undefined, the entire watch expression becomes undefined. If the execution point returns to a location where the watch expression can be evaluated, the Watches window again displays the value of the watch expression.

To add a watch:

1. Begin debugging and when the debugger reaches a breakpoint, click the **Data** tab in the Log window.
2. In the **Name** column, expand the nodes until you see the variable you want to watch.
3. Right-click the variable and select **Watch**.

The Watch window opens with a new row defining the watch for the selected variable.

Using the Test Console

This chapter provides guidelines and information on testing services using the Service Bus Test Console, including how to undeploy the Test Console in production environments.

This chapter contains the following topics:

- [Introduction to the Test Console](#)
- [Accessing the Test Console](#)
- [Testing Proxy Services, Business Services, Pipelines, and Split-Joins](#)
- [Testing MFL Transformations](#)
- [Testing XSLT Transformations \(Resources\)](#)
- [Testing XQuery Transformations \(Resources\)](#)
- [Testing Inline Expressions](#)
- [Testing Services With OWSM Security](#)
- [About Security and Transports](#)
- [Undeploying the Test Console](#)
- [Test Console Page Reference for Services](#)

Introduction to the Test Console

The Service Bus Test Console is a browser-based test environment you use to validate and test the design of your system. It is an extension of the Oracle Service Bus Console, and can be accessed from the console, JDeveloper, and Fusion Middleware Control. The Test Console appears as a tabbed window in JDeveloper or in a web browser, depending on the operating system and the JDeveloper configuration.

When you test a Service Bus resource, you configure the object of your test, execute the test, and view the results in the Test Console windows. In some cases, you can trace through the code and examine the state of the message at specific trace points. Design-time testing helps isolate design problems before you deploy a configuration to a production environment. The components you can test using the Test Console include proxy services, business services, pipelines, split-joins, XQuery resources, XSLT resources, MFL resources, and XQuery/XPath expressions.

The Test Console can test specific parts of your system in isolation and it can test your system as a unit. You can use the Test Console in clustered environments. However, Oracle does not recommend deploying the Test Console in production environments. Only users in the `IntegrationAdmin` and `IntegrationDeployer` roles can use

the Test Console. For more information about roles, see "Roles" in *Administering Oracle Service Bus*.

Most services can be tested using the Test Console, however the Test Console cannot invoke a service that expects a Java object as the message input. For example, messaging services that have a message request or response type of Java cannot be tested, and JEJB operations that expect Java objects cannot be tested.

Proxy Service Testing

When you test a proxy service, the message is sent to the proxy service through the transport layer. The transport layer performs manipulation of message headers or metadata as part of the test. The configuration data you enter for the test simulates the data that is sent to the proxy service from the client. The response from a proxy service test is the message that is sent to the next component in the system. This loosely correlates to an *indirect* call in previous versions of Service Bus.

Use this test approach in conjunction with setting custom (outbound) transport headers in the Test Console Transport section to accurately simulate the service call. For more information on transport settings, see [Test Console Transport Settings](#).

Note:

Testing a request/response MQ or JMS proxy service does not work. The Test Console does not display the response from a call to an MQ or JMS request/response proxy service using a correlation based on a messageID. When you test an MQ or JMS request/response proxy service, the response is retained in the response queue and is not displayed in the Test Console.

Pipeline Testing

When testing pipelines, the input messages are sent directly to the pipeline. Tracing is turned on by default, allowing you to diagnose and troubleshoot a message flow in the Test Console. The input data you enter in the Test Console must be that which is expected by the pipeline from the proxy service that invokes it. In other words, the Test Console plays the role of the proxy service invoking the pipeline. This loosely correlates to a *direct* call in previous versions of Service Bus.

Testing a pipeline tests the internal message flow logic. Use this test approach in conjunction with setting custom (inbound) transport headers in the Test Console Transport panel to accurately simulate the service call.

Execution Tracing in Pipelines Using the Test Console

Tracing a message through a pipeline involves examining the message context and outbound communications at various points in the message flow. The points at which the messages are examined are predefined by Service Bus, which defines tracing for stages, error handlers, and route nodes. For each stage, the trace includes the changes that occur to the message context and all the services invoked during the stage execution.

The following stage information is provided by the trace:

- **Initial Message Context:** Shows the variables initialized by the proxy service when it is invoked. To see the value of any variable, click the + sign associated with the variable name.

- **New variables:** The names of all new variables and their values. Expand a variable to view its value.
- **Deleted variables:** The names of all deleted variables.
- **Changed variables:** The names of all variables for which the value changed, including `$header`, `$body`, and `$inbound` changes as a result of the processing of the message through the stages. Expand a variable to view the new value.
- **Faults:** If an error occurs, the contents of the `fault` context variable (`$fault`) is shown as a result of the stage error handler handling the validation error.
- **Publish:** Every publish call is listed. For each publish call, the trace includes the name of the service invoked, and the value of the `outbound`, `header`, `body`, and `attachment` variables.
- **Service callout:** Every service callout is listed. For each service callout, the trace includes the name of the service that is invoked, the value of the `outbound` variable, the value of the `header`, `body`, and `attachment` variables for both the request and response messages.

The trace contains similar information for route nodes as for stages. In the case of route nodes, the trace contains the following categories of information:

- The trace for service invocations on the request path
- The trace for the routed service
- The trace for the service invocations on the response path
- Changes made to the message context between the entry point of the route node (on the request path) and the exit point (on the response path)

Note:

To see tracing in the log file or standard out (server console), WebLogic Server logging must be set to the following severity levels:

- Minimum severity to log: Info
- Log file: Info
- Standard out: Info

For information on setting log severity levels, see "Using Log Severity Levels" in *Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server*.

Business Service Testing

The input test data for a business service should be in the form expected by the business service as it would come from the pipeline, split-join, or proxy service that invokes the business service. For example, this could be data from a route node or a service callout action of a pipeline. The Test Console functions in the role of the invoking service when you use it to test a business service. When testing business services, messages are always routed through the transport layer.

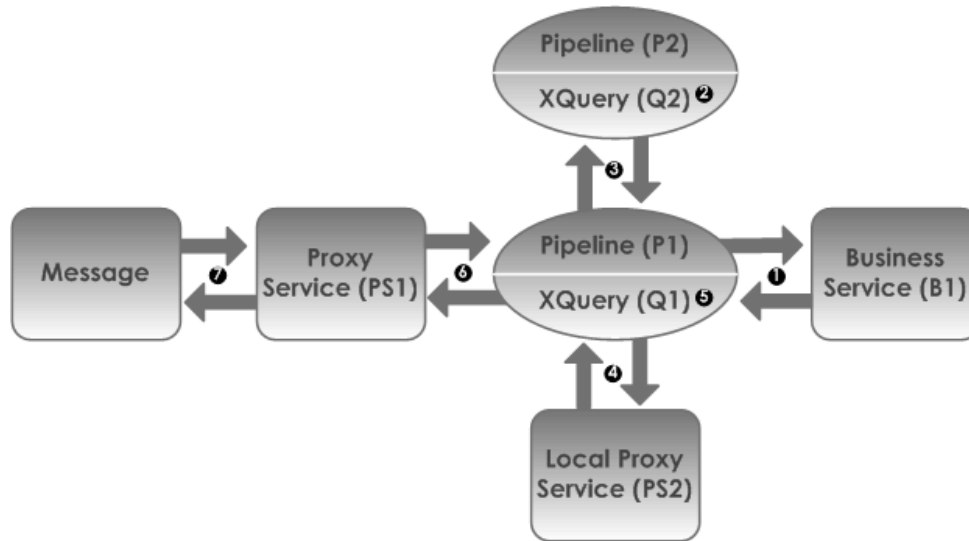
When you test a business service, ensure that the user name and password you specify in the Test Console exists in the local Service Bus domain even if the business service

being tested is in a remote domain. The test service performs a local authentication before invoking any proxy or business service.

Recommended Approaches to Testing Services

In the scenario depicted in [Figure 58-1](#), a client invokes the proxy service PS1, which in turn invokes the pipeline P1. The pipeline invokes pipeline P2, then business service B1, and then local proxy service PS2 before returning a message to the client. Interfaces are identified by number.

Figure 58-1 Test Scenario Example



There are many valid test strategies for this scenario. Oracle recommends the following:

- Complete the testing of interfaces other than the initial proxy service first. In the sample scenario illustrated in [Figure 58-1](#), this means that you complete the testing of interfaces 1 through 6 first, then test interface 7.

Generally, you want to test in the reverse order of the way a message would flow through the system. In this way, the message flow logic in the pipelines can be iteratively changed and tested knowing that the downstream interfaces function correctly.

- Test the pipeline (P1) to business service (B1) interface (1).
- Validate and test all the XQuery expressions in the pipelines prior to testing the pipelines themselves. In [Figure 58-1](#), interface 2 and 5 refer to XQuery expression tests.
- Test the pipeline (P1) to pipeline (P2) interface (3).
- Test the pipeline (P1) to local proxy service (PS2) interface (4).
- Test the initial proxy service (PS1) to pipeline (P1) interface (6).
- Make your final *system* test simulate the client invoking the proxy service PS1. This test is represented by interface 7 in [Figure 58-1](#).
- Save the message state after executing successful interface tests to facilitate future troubleshooting efforts on the system. Testing interface 6 is in fact a test of the

complete system. Knowing that all other interfaces in the system work correctly can help narrow the troubleshooting effort when system errors arise.

HTTP Requests

When you test proxy services, the Test Console never sends an HTTP request over the network, therefore, transport-level access control is not applied. Transport-level access control is achieved through the web application layer. For information about message processing in the transport layer, see [Message Processing](#). For information about transport settings, see [How the Runtime Uses the Transport Settings in the Test Console](#).

Accessing the Test Console

There are multiple ways to access the Test Console to test a specific Service Bus service, from the Oracle Service Bus Console, JDeveloper, or Fusion Middleware Control.

Proxy services, business services, pipelines, and split-joins can only be tested outside a session. Transformations can be tested from outside or inside a session.

Note:

If you receive an error saying the Test Console service is not running, try setting the Admin server listen address to a specific valid address, such as localhost. In the Oracle WebLogic Server Administration Console, go to **Environment** > **Servers** > *admin_server_name* > **Configuration** > **General** to set the Listen Address. Also, in a cluster, make sure all managed nodes are running.

Prerequisites

The following must be in place before you can use the Test Console:

- Service Bus must be running and, if you are testing a service, the session that contains the resource you want to test must be activated. Services can only be tested from outside a session, but transformations can be tested from outside or inside a session.
- Pop-up blockers must be disabled in your browser for XQuery testing to work. If you have toolbars in the Internet Explorer browser, this may mean disabling pop-up blockers from under the **Options** menu as well as for all toolbars that are configured to block them. XQuery testing is done only in the design-time environment (in an active session).
- If you want the Test Console to generate and send SAML tokens to a proxy service, you must configure the proxy service to require SAML tokens and to be a relying party. For more information on creating a SAML relying party, see "Create a SAML 1.1 Relying Party" in the *Oracle WebLogic Server Administration Console Online Help*.

Note:

When creating a SAML relying party:

- Only WSS/Sender-Vouches and WSS/Holder-of-Key SAML profiles are applicable to a proxy service.
 - When you configure the relying party, provide the *URI* of the proxy service for the target URL value. To view the URI of the proxy service, click the open the proxy service in the Oracle Service Bus Console and click the Transport subtab.
-
-

How to Access the Test Console from the Oracle Service Bus Console

On the Oracle Service Bus Console, you can access the Test Console for a service or transformation from that component's definition editor or from the Project or Folder Definition Editor.

Accessing the Test Console from a Component's Definition Editor

To access the Test Console from a component's definition editor:

1. In the Oracle Service Bus Console Project Navigator, expand the project and folders containing the service or transformation you want to test.
2. Click the resource to open it in a definition editor.
3. In the upper right section of the definition editor, click **Launch Test Console** (the green arrow icon).

The Test Console appears in a new browser window, ready to test the selected component.

Accessing the Test Console from the Project or Folder Definition Editor

To access the Test Console from the Project or Folder Definition Editor:

1. In the Oracle Service Bus Console Project Navigator, click the name of the project or folder that contains the component you want to test.

The definition editor for that folder or project appears.

2. Locate the component you want to test in the resources table.
3. In the Actions column, click **Launch Test Console** (the green arrow icon).

The Test Console appears in a new browser window, ready to test the selected component.

How to Access the Test Console from Fusion Middleware Control

In Fusion Middleware Control, you can access the Test Console for a service from that service's Dashboard page. You can only test services from Fusion Middleware Control, and not transformations.

To access the Test Console from Fusion Middleware Control:

1. In the Fusion Middleware Control target navigator, expand **SOA**, expand **service-bus**, and click the project that contains the component you want to test.

The Service Health page for that project appears.

2. If the services associated with the project do not appear in the Services table, perform a search for the service to test.

Note:

Only services that have monitoring enabled are displayed on this page.

3. In the Services table, click the name of the service to test.

The Dashboard for the service appears.

4. To the right of the service name, click **Launch Test Console** (the green arrow icon).

The Test Console appears in a new browser window, ready to test the selected component.

How to Access the Test Console from JDeveloper

In JDeveloper, you can access the Test Console for a service or transformation from that service's definition editor or from its context menu. Transformations provide multiple options for testing, including the Test Console and JDeveloper XSLT tester.

Accessing the Test Console from JDeveloper

To access the Test Console for a service from JDeveloper:

1. In JDeveloper expand the Service Bus project containing the service you want to test.
2. Do one of the following:
 - Double-click the service to display its editor, and then click the **Run** icon in the JDeveloper toolbar.
 - Right-click the service and select **Run**.

The Test Console appears in a new tabbed window, ready to test the selected service.

Accessing the Test Console for a Transformation from JDeveloper

To access the Test Console for a transformation from JDeveloper:

1. In JDeveloper expand the Service Bus project containing the transformation you want to test.
2. Do one of the following:
 - Click the transformation to display its editor, and then click the **Run** icon in the JDeveloper toolbar.
 - Right-click the transformation and select **Run**.

Note:

Clicking **Debug** also launches the Test Console, but is typically used in conjunction with breakpoints so you can step through the message flow in sections.

3. If a dialog appears, select **In Service Bus Test Console** as the choice for how the target should be started.

The Test Console appears in a new tabbed window, ready to test the selected service.

Testing Proxy Services, Business Services, Pipelines, and Split-Joins

When testing Service Bus services, you cannot be in an active session. The Test Console lets you test input, transport headers, attachments, and certain security configurations. For more information about testing services, see [Introduction to the Test Console](#).

Note:

- In a clustered domain, you cannot use the Test Console to test any configured business service or proxy service which routes to a business service.
 - When the Test Console invokes a proxy with HTTP custom token authentication, the authentication check is not performed.
-
-

How to Test Service Bus Services

When you launch the Test Console for a service, you can configure the test input, including the operation to test, the message payload, transport headers, and security information. For pipelines, you can trace message processing through the different components. The following figure shows an example of a Test Console page.

Figure 58-2 Pipeline Testing Page on the Test Console

Pipeline Testing - LoanGateway1 Help

Execute Execute-Save Reset Close

Service Operation

Operation: processLoanApp

Test Configuration

Include Tracing:

Request Document

Form XML

SOAP Header: `<soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
</soap:Header>`

loanRequest: `<loanRequest xmlns:java="java:normal.client">
<java:Name>Elizabeth Miller</java:Name>
<java:SSN>1234567890</java:SSN>
<java:Rate>4.9</java:Rate>
<java:Amount>15</java:Amount>`

To test a Service Bus service:

1. If applicable, make sure the session in the Oracle Service Bus Console is activated.
2. Locate the proxy service, business service, pipeline, or split-join you want to test, and launch the Test Console, as described in [Accessing the Test Console](#).
3. For SOAP and XML services, select the WSDL operation you want to test.
4. In the Request Document section, enter the test data to use. This must be the data that the service expects to receive.

For information about filling in this section, see [Request Document Test Console Properties](#).

Note:

A *secured* SOAP message is displayed with extra white spaces. Because white spaces can affect the semantics of the document, this SOAP message cannot always be used as the literal data. For example, digital signatures are white space-sensitive and can become invalid.

5. Configure the remaining sections of the Test Console. For information about the properties you can set, see [Test Console Page Reference for Services](#).
6. Click **Execute**.

The Test Console displays the request message and the service's response message and metadata. For information about interpreting the test results, see [How to View Service Test Results](#).

7. To run the test again, click **Back**. Repeat steps 3 through 6.

How to Test Attachments in Services

You can test message attachments with proxy services, business services, and pipelines.

To test services with attachments:

1. If applicable, make sure the session in the Oracle Service Bus Console is activated.
2. Locate the proxy service, business service, or pipeline you want to test, and launch the Test Console, as described in [Accessing the Test Console](#).
3. For SOAP and XML services, select the WSDL operation you want to test.
4. For pipelines, specify whether to enable tracing. For more information, see [How To Trace Pipeline Processing](#).
5. In the Request Document section, enter the payload for the test message.

As an example, the following input uses a `submitAttachment` operation to send a ZIP file as an attachment in a SOAP message.

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <m:submitAttachment xmlns:m="http://www.alsb.com/SOAPwithAttachment/">
      <submitAttachment>
        <fileName>c:\yourfile.zip</fileName>
      </submitAttachment>
      <zipFile href="cid:zipFile"/>
    </m:submitAttachment>
  </env:Body>
</env:Envelope>
```

For information about filling in this section, see [Request Document Test Console Properties](#).

6. In the Attachment section of the Test Console, enter values for the attachment headers, as described in [Attachment Test Console Properties](#). You must select the file to use as an attachment.
7. Configure the remaining sections of the Test Console. For information about the properties you can set, see [Test Console Page Reference for Services](#).
8. Click **Execute**.

To confirm success of the sent attachment, check the server console for a message similar to the following, which is logged in our example by the `submitAttachment` operation:

```
WS called - received the following properties:
submitAttachment is:
    com.alsb.soapwithattachment.SubmitAttachmentRequestType@e2abb0
zipFile is: javax.activation.DataHandler@175cf0a
```

How To Trace Pipeline Processing

When you enable tracing for a pipeline, the test results include the details of the trace. Use tracing to track problems in the system and to isolate them for correction. The trace information shows the path taken through the request and response paths in the code. The following figure shows the results of a pipeline trace.

While viewing the trace you can also view the message flow in the Oracle Service Bus Console or JDeveloper. This helps you relate the trace to the stages and actions in the message flow. You can modify the message flow and run the trace again to view the output.

Figure 58-3 Pipeline Invocation Trace

```

Invocation Trace
├── (receiving request)
│   └── Initial Message Context
│       ├── added $body
│       ├── added $header
│       ├── added $inbound
│       └── added $messageID
├── RouteNode1
│   └── Routed Service
│       ├── Route to: "ManagerLoanReview"
│       ├── $outbound:
│       ├── $body (request):
│       ├── $header (request):
│       └── $attachments (request):
│           └── Message Context Changes
│               └── added $outbound
│                   └── <con:endpoint name="BusinessService$MortgageBroker$BusinessService$ManagerLoanReview"
│                       xmlns:con="http://www.bea.com/wli/sb/context">
│                           <con:service>
│                               <con:operation>processLoanApp</con:operation>
│                           </con:service>
│                           <con:transport>
│                               <con:uri>
│                                   http://localhost:7001/mjws_basic_ejb/ManagerSimpleBean
│                               </con:uri>
│                               <con:mode>request-response</con:mode>
│                               <con:qualityOfService>best-effort</con:qualityOfService>
│                               <con:request xsi:type="http:HttpRequestMetaData"
│                                   xmlns:http="http://www.bea.com/wli/sb/transports/http"
│                                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

```

To trace a message through a pipeline:

1. If applicable, make sure the session in the Oracle Service Bus Console is activated.
2. Locate the pipeline you want to test, and launch the Test Console, as described in [Accessing the Test Console](#).
3. For SOAP and XML services, select the WSDL operation you want to test from the available options in the Service Operation section.
4. In the Test Configuration section, make sure **Include Tracing** is selected.
5. In the Request Document section, enter the test data to use. This must be the data that the pipeline expects to receive.

For information about filling in this section, see [Request Document Test Console Properties](#).

6. Configure the remaining sections of the Test Console. For information about the properties you can set, see [Test Console Page Reference for Services](#).
7. Click **Execute**.

The Test Console displays the request message and the service's response message and metadata. For information about interpreting the test results, see [How to View Service Test Results](#).

8. Scroll down to the Invocation Trace section.

This section displays a representation the message flow. You can trace the message through the service and view the state of the message at pre-selected points in the trace. The trace points are automatically set.

9. Click the + plus sign to expand the message flow to view more detail.

How to View Service Test Results

When you test a proxy service, business service, pipeline, or split-join, the Test Console displays the results in several sections. [Table 58-1](#) describes the testing results sections.

Table 58-1 Testing Results for Proxy Services

Section	Description
Request Document	<p>The request message sent to the service by the Test Console. This section is initially collapsed if the Test Console did not modify the request message. This section is initially expanded for SOAP services configured using the Form tab or if WS-Security has been applied.</p> <p>If WS-Security has been applied, this section contains two SOAP messages. The first message is the <i>clear text</i> message; the second is the <i>secured</i> SOAP message.</p>
Response Document	<p>The message response generated by the service. This section also indicates if any errors occurred.</p> <p>For a SOAP service with a WS-Security response, this section contains two SOAP messages. The first SOAP message is the <i>secured</i> message as received by the client. The second SOAP message is the corresponding <i>clear text</i> message.</p>
Response Metadata	The metadata returned with the message response.
Invocation Trace	Tracing is available only for pipelines, and this section shows the state of the message as it passes through the system. This is only performed when you select Include Tracing prior to executing the test.

Testing MFL Transformations

A Message Format Language (MFL) document is a specialized XML document used to describe the layout of binary data. MFL resources support the following transformations:

- XML to binary: There is one required input (XML) and one output (binary).
- binary to XML: There is one required input (binary) and one output (XML).

Each transformation accepts only one input and provides a single output. You can test MFL transformations using the Test Console, accessed from either JDeveloper or the Oracle Service Bus Console, or using the tester in the Format Builder in JDeveloper.

How to Test MFL Transformations in the Test Console

You can test transformations after activating a session or during a session to ensure that the resources operate with the expected behavior. If you do not active the session, the testing is done at design time using your local changes.

In JDeveloper, you can also use the Format Builder's built-in testing tool. For information about running tests from the Format Builder, see [Testing Format Definitions](#).

To test MFL transformations in the Test Console:

1. To test the runtime, activate the current session. To test the design time, do not activate the session.
2. Locate the MFL resource you want to test, and launch the Test Console, as described in [Accessing the Test Console](#).
3. Configure the test data for the MFL resource. For more information, see [Table 58-2](#).
4. Click **Execute**.

The Test Console displays the results.

5. To retest, click **Back**. You can close the Test Console, modify, and retest the resource.

Table 58-2 MFL Test Console Properties

Section	Description
Supported transformations	To select a specific transformation to test, select the transformation name.
Input Document	<ul style="list-style-type: none"> • XML Input: Required for XML to binary transformations: The XML schema for the MFL document can be inferred. A sample XML document is automatically entered in the text field. The XML input can be file-based or text-based. Referencing a file for input takes precedence over textual input. Browse and select the file you want to use in your test. • Binary Input: Required for binary to XML transformations: The binary input can be file-based or text-based. Referencing a file for input takes precedence over textual input. Browse and select the file you want to use in your test.

MFL Test Console Example

The following example illustrates testing an MFL transformation, and shows the contents of the MFL file, the test input, and the test results. The Test Console generates a sample XML document from the MFL file, and you execute the test using the XML

input. A successful test results in the transformation of the message content of the input XML document to binary format.

Example - Contents of the MFL File

Below is an example of an MFL file.

```
<?xml version='1.0' encoding='windows-1252'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
  <MessageFormat name='StockPrices' version='2.01'>
    <StructFormat name='PriceQuote' repeat='*'>
      <FieldFormat name='StockSymbol' type='String' delim=':' codepage='windows-1252' />
      <FieldFormat name='StockPrice' type='String' delim='|' codepage='windows-1252' />
    </StructFormat>
  </MessageFormat>
```

Example - Test Console XML Input

The XML document generated by the Test Console to test the MFL file in the previous example is shown below.

```
<StockPrices>
  <PriceQuote>
    <StockSymbol>StockSymbol_31</StockSymbol>
    <StockPrice>StockPrice_17</StockPrice>
  </PriceQuote>
</StockPrices>
```

Example - MFL Test Console Results

When you click **Execute** in the Test Console to run this test, the console displays the following data (the stock symbol and stock price in binary format).

```
00000000:53 74 6F 63 6B 53 79 6D 62 6F 6C 5F 33 31 3A 53 StockSymbol_31:S
00000010:74 6F 63 6B 50 72 69 63 65 5F 31 37 7C .. .. StockPrice_17|...
```

Testing XSLT Transformations (Resources)

Extensible Stylesheet Language Transformation (XSLT) describes XML-to-XML mappings in Service Bus. You can use XSLT transformations in XQuery expressions in message flows.

To test an XSLT resource, you must supply an input XML document. The Test Console returns the output XML document. An XSLT transformation can include multiple parameters to assist with a transformation. All parameters required by the transformation are displayed on the Test Console. Default values are available but you can override them. XSLT parameters accept either primitive values or XML document values. You cannot identify the types of parameters from the XSL transformation.

You can access the Test Console for XSLT transformations from the Oracle Service Bus Console or from JDeveloper. In JDeveloper, you can also use the XSLT mapper's built-in tester.

How to Test XSLT Transformations Using the Test Console

XSLT transformations can be tested after activating a session or during a session to ensure that the resources operate with the expected behavior. You must activate the session to test the runtime, otherwise the testing is done at design time using your local changes.

To test an XSLT transformation using the Test Console:

1. To test the runtime, activate the current session. To test the design time, do not activate the session.
2. Locate the XSLT resource you want to test, and launch the Test Console, as described in [Accessing the Test Console](#).
3. Configure the test data for the resource by entering the following information.
 - **XML Input:** The XML input can be file-based or text-based. If you select a file for input, it takes precedence over textual input. Browse and select the file you want to use in your test. XML input is required.
 - ***param_name*:** A named XSLT parameter. There are two types of input: XML and primitive (String, integer, float, and so on). The default input type is String. Select the `as XML` check box associated with the parameter name to identify a parameter of type XML. For more information about this option, see [Testing XQuery Transformations \(Resources\)](#).
4. Click **Execute**.
The Test Console displays the results.
5. To retest, click **Back**. You can close the Test Console, modify, and retest the resource.

How to Test XSLT Transformations Using the JDeveloper XSLT Mapper

While you can access the Test Console for XSLT transformations from JDeveloper, JDeveloper provides its own tester for XSLT transformations that you can access directly from the XSLT Mapper. Custom XPath functions cannot be tested in the mapper.

For information and instructions about running XSLT transformation tests in JDeveloper, see "Testing the Map" in *Developing SOA Applications with Oracle SOA Suite*.

Testing XQuery Transformations (Resources)

XQuery uses the structure of XML to express queries across different kinds of data, whether physically stored in XML or viewed as XML. An XQuery transformation can take multiple inputs and returns one output. The inputs expected by an XQuery transformation are variable values to bind to each of the XQuery external variables defined. The value of an XQuery input variable can be a primitive value (String, integer, date), an XML document, or a sequence of the previous types. The output value can be a primitive value (String, integer, date), an XML document, or a sequence of the previous types.

XQuery is a typed language, which means every external variable is given a type. The types can be categorized into the following groups:

- Simple/primitive type (string, int, float, and so on)
- XML nodes
- Untyped

In the Test Console, all three variables are listed in the Variables section of the Test Console, and you configure their input when performing the test. By default, XML is selected for the untyped variable as it is the most typical case.

XQuery Transformation Testing Prerequisites and Guidelines

You must disable the pop-up blockers in your browser for the XQuery testing to work. Note that if you have toolbars in the Internet Explorer browser, you may need to disable pop-up blockers from under the browser **Options** menu as well as for all toolbars that are configured to block them.

When performing XQuery testing in the Test Console, use the **Back** button to execute a new test. However, if you want to execute a new test after making changes to the XQuery, you must close and re-open the Test Console for the changes to take effect.

How to Test XQuery Transformations in the Test Console

XQuery maps can describe XML-to-XML, XML to non-XML, and non-XML to XML mappings. The Test Console does not support sequences on input. You can test transformations after activating a session or during a session to ensure that the resources operate with the expected behavior. You must activate the session to test the runtime, otherwise the testing is done at design time using your local changes.

To test an XQuery transformation using the Test Console:

1. To test the runtime, activate the current session. To test the design time, do not activate the session.
2. Locate the XQuery resource you want to test, and launch the Test Console, as described in [Accessing the Test Console](#).
3. Configure the test data for the resource by entering the XQuery input. The input is based on the external variables, as described below.
 - There is one input field named for each of the XQuery external variables.
 - A single-line edit box is displayed if the type is a simple type. A multi-line edit box is displayed if the data is XML.
 - A combination input field is used when the variable is not typed. You must declare the variable type. Select the `as XML` check box to identify a parameter of type XML.
 - An XML input can be file-based or text-based. Referencing a file for input takes precedence over textual input. Browse and select the file you want to use in your test.
 - Input in the Test Console is rendered based on the type to make it easier to understand the type of data you must enter. When untyped, the default type is XML.
4. Click **Execute**.

The Test Console displays the results.
5. To retest, click **Back**. You can close the Test Console, modify, and retest the resource.

Testing Inline Expressions

You can test expressions in a message flow action from the XQuery/XSLT Expression Editor, the XQuery Condition Editor, and the XPath Expression Editor when you

create or modify an expression or condition in the Oracle Service Bus Console. You can also access the Test Console from the Expression Builders in JDeveloper. Testing takes the same form for both the XQuery/XSLT expression and condition editors, but is slightly different for the XPath Expression Editor.

How to Test XQuery Expressions

You can test XQuery expressions directly from the XQuery/XSLT Expression Editor and the XQuery Condition Editor.

To test XQuery expressions:

1. Create or update the expression in the pipeline action.
2. Do one of the following:
 - In JDeveloper: Click the **Test Expression** icon.
 - In the Oracle Service Bus Console: Click **Validate** to make sure the expression is valid, and then click **Test**.

The Test Console appears, and displays the expression being tested.

3. Configure the test data for the resource by entering the XQuery input in the Data Inputs section. The input is based on the external variables, as described below.
 - There is one input field named for each of the XQuery external variables.
 - A single-line edit box is displayed if the type is a simple type. A multi-line edit box is displayed if the data is XML.
 - A combination input field is used when the variable is not typed. You must declare the variable type. Select the `as XML` check box to identify a parameter of type XML.
 - An XML input can be file-based or text-based. Referencing a file for input takes precedence over text input. Browse and select the file you want to use in your test.
 - Input in the Test Console is rendered based on the type to make it easier to understand the type of data you must enter. When untyped, the default type is XML.
4. Click **Execute**.

The Test Console displays the results.

5. To retest, click **Back**. You can close the Test Console, modify, and retest the resource.

To execute a new test after making changes to the XQuery, you must close and reopen the Test Console for the changes to take effect.

How to Test XPath Expressions

You use XPath expressions to select a subset of an XML message context variable. You can use the Test Console in the XPath Expression Editor to test the definition of the XPath expression. An XPath expression takes a single XML document as input and generates a sequence of XML documents, primitives types, or both as output.

To test an XPath expression:

1. Create or update the XPath expression in the message flow action.
2. When you are done defining the expression, click **Validate** to make sure the expression is valid.
3. Click **Test**.

The Test Console appears, and displays the expression being tested.

4. Configure the test data for the resource by entering the XML input in the Data Inputs section.
 - This section contains a single input field corresponding to the XML document against which this XPath expression is being tested.
 - The XML input can be file-based or text-based. Referencing a file for input takes precedence over textual input. Browse and select the file you want to use in your test.
5. Click **Execute**.

The Test Console displays the results.

6. To retest, click **Back**. You can close the Test Console, modify, and retest the resource.

To execute a new test after making changes to the XPath expression, you must close and reopen the Test Console for the changes to take effect.

Testing Services With OWSM Security

The Test Console supports testing proxy services and business services protected with OWSM security policies. When a service has OWSM policies attached, the message exchange between the Test Console and the service is protected by the mechanisms of the policy. According to the policy, the test service digitally signs or encrypts the message (more precisely, parts of the message) and includes any applicable security tokens. You specify the input to the digital signature and encryption operations in the clear-text SOAP envelope specified as described in [Request Document Test Console Properties](#).

If you specify a service key provider in the Test Console, all client-side PKI key-pair credentials required by WS-Security are retrieved from the service key provider. You use the user name and password fields when an operation's request policy specifies an Identity Assertion and Username Token as one of the supported token types.

[Table 58-3](#) and [Table 58-4](#) describe security scenarios.

Table 58-3 Digital Signature and Encryption Scenarios

Scenario	Is Service Key Provider Required?
The request policy has a Confidentiality assertion.	<p>No. The test service encrypts the request with the service's public key. When testing a proxy service, the test service automatically retrieves the public key from the encryption certificate assigned to the service key provider of the proxy service.</p> <p>When testing a business service, the encryption certificate is embedded in the WSDL file of the business service. The test service automatically retrieves this WSDL file from the WSDL repository and extracts the encryption certificate from the WSDL file.</p>
The response policy has a Confidentiality assertion.	<p>Yes. In this scenario, the operation policy requires the client to send its certificate to the service. The service will use the public key from this certificate to encrypt the response to the client. A service key provider <i>must</i> be specified and <i>must</i> have an associated encryption credential.</p> <p>If both request and response encryption are supported, different credentials must be used.</p>
The request policy has an Integrity assertion.	<p>Yes. The client must sign the request. A service key provider <i>must</i> be specified and <i>must</i> have an associated digital signature credential.</p> <p>Furthermore, if this is a SAML holder-of-key integrity assertion, a user name and password is needed in addition to the service key provider.</p>
The response policy has an Integrity assertion.	<p>No. In this case, the policy specifies that the service must sign the response. The service signs the response with its private key. The Test Console simply verifies this signature.</p> <p>When testing a proxy service, this is the private key associated to the service key provider's digital signature credential for the proxy service.</p> <p>When testing a business service, the service signing key-pair is configured in a product-specific way on the system hosting the service.</p> <p>In the case that the current security realm is configured to do a Certificate Lookup and Validation, the certificate that maps to the service key provider must be registered and valid in the certificate lookup and validation framework.</p> <p>For more information on Certificate Lookup and Validation, see "Configuring the Certificate Lookup and Validation Framework" in <i>Administering Security for Oracle WebLogic Server</i>.</p>

Table 58-4 Identity Policy Scenarios (Assuming that the Policy has an Identity Assertion)

Supported Token Types (From the Identity Assertion inside the request policy)	Description	Comments
UNT	The service only accepts WSS user name tokens	You must specify a user name and password in the Security panel.
X.509	The service only accepts WSS X.509 tokens	You must specify a service key provider in the Security panel and the service key provider must have an associated WSS X.509 credential.
SAML	The service only accepts WSS SAML tokens	You must specify a user name and password in the Security panel <i>or</i> a user name and password in the Transport panel. If both are specified, the one from the Security panel is used as the identity in the SAML token.
UNT, X.509	The service accepts UNT or X.509 tokens	You must specify a user name and password in the Security panel <i>or</i> a service key provider in the Security panel with an associated WSS X.509 credential. If both are specified, only a UNT token is generated.
UNT, SAML	The service accepts UNT or SAML tokens	You must specify a user name and password in the Security panel <i>or</i> a user name and password in the Transport panel. If both are specified, only a UNT token is sent.
X.509, SAML	The service accepts X.509 or SAML tokens	You must specify one of the following: <ul style="list-style-type: none"> • user name and password in the Security panel • user name and password in the Transport panel • service key provider with an associated WSS X.509 credential
UNT, X.509, SAML	The service accepts UNT, X.509 or SAML tokens	You must specify one of the following: <ul style="list-style-type: none"> • user name and password in the Security panel • user name and password in the Transport panel • service key provider with an associated WSS X.509 credential

Limitations for Services and Policies

The following limitations exist for testing proxy services with SAML policies and business services with SAML holder-of-key policies:

- Testing proxy services with inbound SAML policies is not supported.
- Testing business services with a SAML holder-of-key policy is a special case. The SAML holder-of-key scenario can be configured in two ways:

- As an integrity policy (this is the recommended approach)
- As an identity policy

In both cases, you must specify a user name and password; the SAML assertion will be on behalf of this user. If SAML holder-of-key is configured as an integrity policy, you must also specify a service key provider. The service key provider must have a digital signature credential assigned to it. This case is special because this is the only case where a user name and password must be specified even if there is not an identity policy.

Note:

After executing a test in the Test Console, the envelope generated with WSS is not always a valid envelope; the results page in the Test Console includes white spaces for improved readability. That is, the secured SOAP message is displayed with extra white spaces. Because white spaces can affect the semantics of the document, this SOAP message cannot always be used as the literal data. For example, digital signatures are white-space sensitive and can become invalid.

About Security and Transports

When using the Test Console to test HTTP business services with basic authentication, the Test Console authenticates the user name and password from the service account of the business service. Similarly, when testing JMS, email, or FTP business services that require authentication, the Test Console authenticates the service account associated with the business service.

When you test proxy services, the Test Console never sends a HTTP request over the network. Therefore, transport-level access control is not applied.

Undeploying the Test Console

Oracle recommends that you not use the test framework in production systems. For example, testing pipelines bypasses some important security steps, including access control.

When you create a Service Bus domain, the Configuration Wizard, by default, includes the ALSB Test Framework (Test Console) as a target on the Admin Server and any Managed Servers. The following section describe different options for undeploying the Test Console:

- [Untargeting the Test Console Before Domain Creation](#)
- [Untargeting the Test Console when the Server is Running](#)
- [Untargeting the Test Console when the Server is Not Running](#)

Untargeting the Test Console Before Domain Creation

To untarget the Test Console in the Oracle Fusion Middleware Configuration Wizard before a domain is created:

1. When creating a Service Bus domain with the Configuration Wizard, select the optional configuration for Deployments and Services.

2. In the related wizard pages that follow, for each server, select **Service Bus Test Framework** in the Targets panel and click the left-facing arrow to move it to the Deployments panel.

When the wizard creates the domain, the Test Console (`OSB_ORACLE_HOME\lib\apps\TestFwk.ear`) is not deployed.

Untargeting the Test Console when the Server is Running

You can undeploy the Test Console if it has already been deployed to the Service Bus server and the server is already running.

To undeploy the Test Console on a running Service Bus domain:

1. Start the Oracle WebLogic Server Administration Console and log in.
2. In the left navigation pane under Domain Structure, click **Deployments**.
The Summary of Deployments page appears.
3. In the Deployments table, click **Service Bus Test Framework**.
The Overview page for the Service Bus Test Framework appears.
4. Click the **Targets** tab.
5. Select the check box next to Service Bus Test Framework in the Target Assignments table to select all the test framework resources.
6. Click **Change Targets**.
7. On the Target Deployments page, clear the check boxes next to the Admin Server and all Managed Servers.
8. Click **Yes**.

A message is displayed indicating that the settings have been successfully updated.

Untargeting the Test Console when the Server is Not Running

If a Service Bus domain is not running, you can use the WebLogic Scripting Tool (WLST) to untarget the Test Console from the domain..

To untarget the Test Console using WLST:

1. If you have not already set up your environment to use WLST, see Main Steps for Using WLST in Interactive or Script Mode in *Understanding the WebLogic Scripting Tool*.
2. Run the following command to invoke WLST Offline.

```
C:>java weblogic.WLST
```

Note:

You must have your environment set up properly for this command. Depending on your operating system, run `setWLSEnv.cmd` or `setWLSEnv.sh` from `WL_HOME/server/bin`.

3. To read the domain that was created using the Configuration Wizard, run the following command:

```
wls:/offline>readDomain('C:/oracle/user_projects/domains/domain_name')
```

4. To untarget the Service Bus Test Framework application, run the following command:

```
wls:/offline/domain_name>unassign("AppDeployment", "Service Bus Test Framework", "Target", "AdminServer", "ManagedServer_1", "ManagedServer_2")
```

Include the names of all managed servers in the command.

5. To update the domain, run the following command:

```
wls:/offline/base_domain>updateDomain()
```

6. To close the domain, run the following command:

```
wls:/offline/base_domain>closeDomain()
```

7. Run the following command to exit from the WLST command prompt:

```
wls:/offline>exit()
```

Test Console Page Reference for Services

The following sections describe each section displayed on the Oracle Service Bus Test Console when you test a Service Bus service. Not all sections are displayed for all components, and some sections vary depending on the component being tested.

The Test Console request page includes a combination of the following sections, depending on the type of service you are testing and the type of messaging it uses:

- [Test Configuration Test Console Properties](#)
- [Service Operation Test Console Properties](#)
- [Request Document Test Console Properties](#)
- [Security Test Console Properties](#)
- [Authentication Test Console Properties](#)
- [Transport Test Console Properties](#)
- [Attachment Test Console Properties](#)

Test Configuration Test Console Properties

This section only appears when testing pipelines, and it includes one field: Include Tracing. Select this check box to include a trace of the state of the message at each stage of processing.

Service Operation Test Console Properties

This section appears when the proxy service, business service, pipeline, or split-join being tested is based on a WSDL file with operations and when testing a RESTful proxy or business service. The fields that appear in this section vary depending on

whether the service is WSDL-based or RESTful, and whether you select the SOAP or REST view for a RESTful business service.

For RESTful business services, the REST view displays information as a pure REST service. The SOAP view displays information as the caller (pipeline) would see the data (that is, wrapped as a WSDL SOAP service). The fields for the SOAP view are the same as those for a WSDL-based service.

Table 58-5 Test Console Properties- Service Operation

Property	Description
Operation	Select the WSDL operation to test from the list of operations associated with the service. This field is available for WSDL-based services and on the SOAP view for RESTful business services.
Resource	Select an existing URL resource path from the list. A resource is any source of specific information that can be addressed. This field is only available when testing RESTful services.
Method	Select the REST operation to perform. This field is only available when testing RESTful services.
Accept	Select the accepted media type for the test input from the list of options. Options vary depending on the service configuration. This field is only available when testing RESTful services.

Request Document Test Console Properties

The input fields generate the request message that is sent to the proxy service, business service, pipeline, or split-join. The values you enter in the Request Document section are service-type specific. The service types and a description of the input required by each are listed below.

The Invocation Mode option is only displayed when testing any SOAP or any XML proxy service. Clear the **Request/Response** check box for one-way service invocations. Select the check box for request/response invocations.

Table 58-6 Test Console Properties - Request Document

Service Type	Description
WSDL	If the service is a WSDL-based service with multiple operations defined, the Test Console generates and provides a sample document to use when testing the service. Use this sample data directly, edit it, and then run the test, or provide your own test data.
Any XML	Enter the request input in the form of a payload. The payload is the content of the message being sent. The content is expected to be an XML document. You can browse to a file or enter the message content directly in the text box.
Any SOAP	Enter the request input in the form of a payload. The payload is the content of the message being sent. The content is expected to be the SOAP envelope. You can browse to a file or enter the message content in the text box.

Table 58-6 (Cont.) Test Console Properties - Request Document

Service Type	Description
Messaging	<p>Enter a single input, either file-based or text-based. Messaging services define several different input types, including Binary, Java, MFL, XML, and text. For the type none, no input is required.</p> <p>Oracle recommends entering binary input from a file. Data entered in the text area is converted to binary input using the system encoding.</p> <p>Data entered from files for text services must be converted to text. The encoding input field specifies the encoding to apply during the conversion. The system encoding is used if this field is not configured. By default, the encoding field is initialized with the encoding value configured on the proxy service endpoint.</p>
Soap Document	<p>For a SOAP document, the SOAP envelope is usually composed of zero or more headers and one body payload. The Form and XML tabs provide alternative ways to specify the content.</p> <p>The Form tab contains a SOAP Header field and a SOAP Body field. The content of the SOAP Header field is expected to be a SOAP Header tag (this allows for the definition of more than one header). The SOAP Body field contains the data that is actually sent as part of the message. The content is expected to be an XML document. Both the header and the body are used to generate the SOAP envelope.</p>
SOAP RPC	<p>For SOAP RPC, the SOAP envelope is composed of zero or more headers, and zero or more arguments. The Form and XML tabs provide alternative ways to specify the content.</p> <p>The Form tab contains a single input for SOAP headers, and one input field for each argument (the name of the input field corresponds to the name of the argument). The content of the SOAP Header field is expected to be a SOAP Header tag (this allows for the definition of more than one header).</p> <p>The Test Console uses the WSDL file to detect the type of each argument. A single-line input field is used for primitive types; a multi-line input field is used for XML types. A sample document is automatically generated to facilitate testing.</p> <p>The headers and arguments are used by the Test Console to generate the SOAP envelope.</p> <p>The XML tab contains a single input field. The content of this field is expected to be the SOAP envelope being sent. The payload (XML input) can be file-based or text-based. Referencing a file for input takes precedence over textual input. Browse and select the file you want to use in your test.</p>

Table 58-6 (Cont.) Test Console Properties - Request Document

Service Type	Description
REST	<p>The input for a RESTful service varies depending on the resource being tested and whether you are using the SOAP view or the REST view.</p> <p>In the SOAP view, enter the request input in the form of a payload. The payload is the content of the message being sent. Click Browse to navigate to and select a file or enter the message content in the text box.</p> <p>In the REST view, the request input can be in the form of REST parameters or in the form of a payload. If parameter fields are displayed, enter the input for each parameter. If the Payload field is displayed, select the media type from the list of options, and then click Browse to navigate to and select a file or enter the message content in the text box.</p>

Security Test Console Properties

If the service being tested is secured using OWSM policies, the Security panel appears on the test console. You can modify override values and add or remove security policies. To test with additional policies, click the **Add** button and select the policies to test. For more information, see [Securing Oracle Service Bus with Oracle Web Services Manager](#).

Table 58-7 Test Console Properties - Security

Property	Description
Policy Name	Displays the name of the OWSM policies attached to the service being tested.
Property	Displays the name of each available override property for the listed policies. The available properties vary based on the policies that are attached.
Default Value	Displays the default value, if any, for each override property.
Override Value	Enter the value you want to use for the override property when testing the service.
Actions	Click the Delete icon in this column to remove an attached policy for testing purposes.

Authentication Test Console Properties

If the service being tested routes the message to a business service that expects a SAML token, this is the identity that will be represented by the token. For more information, see [Using SAML with Oracle Service Bus](#).

Table 58-8 Test Console Properties - Authentication

Property	Description
Username	Enter the user name for setting the security context used by the test service when invoking the service. Do not confuse this field with the Web Services Security (WSS) user name field. This must be a valid user name and password in the local security realm. An invalid user name or invalid password will cause a client-side error on the test service. Note: When the Test Console invokes a proxy with HTTP custom token authentication, the authentication check is not done.
Password	Enter password associated with the user name.
Service Key Provider	Enter the service key provider to use for authentication. This field is used when testing an HTTPS business service with client certificate authentication. The service provider must have an associated SSL client credential. The test service will use that credential during the SSL handshake.

Transport Test Console Properties

Use the Transport panel in the Test Console to specify the metadata and transport headers for messages in your test system. The following sections describe the transport settings and how they affect processing:

- [Test Console Transport Settings](#)
- [How the Runtime Uses the Transport Settings in the Test Console](#)

Test Console Transport Settings

The Transport section of the Test Console varies depending on the type of transport being tested. [Figure 58-4](#) shows the Transport section for a WSDL-based proxy service.

Figure 58-4 Transport Panel in the Test Console

Transport

Username:

Password:

encoding:

relative-URI:

query-string:

query-parameters:

client-host:

client-address:

http-method:

Accept:

Accept-Encoding:

Accept-Language:

Connection:

Content-Encoding:

Content-Type:

Cookie:

Host:

SOAPAction:

User-Agent:

User Headers: Name:

Value:

User Metadata: Name:

Value:

By setting the metadata and the transport headers in the pipeline, you influence the actions of the outbound transport. You can test the metadata, the message, and the headers so that you can view the pipeline output. The fields that are displayed in the Transport panel when testing a proxy service represent those headers and metadata that are available in the pipeline. The Test Console cannot filter the fields it displays depending on the proxy service. The same set of transport parameters are displayed for every HTTP-based request.

The **Username** and **Password** fields are used to implement basic authentication for the user that is running the proxy service. The **Username** and **Password** fields are not specifically transport related.

Metadata fields are located below the **Username** and **Password** fields and above the transport header fields. The fields displayed are based on the transport type of the service. Certain fields are pre-populated depending on the operation selection algorithm you selected for the service when you defined it. For more information about the selection algorithms, see [Modeling Message Flow in Oracle Service Bus](#).

Specify the values in the Transport panel fields according to the type of service being tested. When testing a pipeline or split-join, the test data should represent the message in the state expected at the point it leaves the caller and enters the service being tested. When testing a business service, the test data represents the data that is sent from a route node or a service callout.

The following properties are common for most transport types. For information about specific headers and metadata and how they are handled by the test framework, see [How the Runtime Uses the Transport Settings in the Test Console](#).

Table 58-9 Test Console Properties - Transport

Property	Description
Username	Enter a user name to implement basic authentication for the user that is running the test. This is not transport-specific.
Password	Enter the password for the user name entered above.
Request/Response	Clear the Request/Response check box for one-way service invocations. Select the check box for request/response invocations. This option is only displayed when testing any SOAP or any XML proxy service.
Service Key Provider	Enter the service key provider to use for authentication. This field is used when testing an HTTPS business service with client certificate authentication. The service provider must have an associated SSL client credential. The test service will use that credential during the SSL handshake.

How the Runtime Uses the Transport Settings in the Test Console

[Transport Test Console Properties](#) describes how you configure the values of the transport headers, transport metadata, and transport-related security data for outbound requests when you test proxy services or business services in the Test Console. However, some specifications you can make in the Test Console are not honored at runtime. That is, the values of certain headers or metadata are overwritten, or ignored by Service Bus at runtime when the test is executed. The headers and metadata for which there are limitations are described in [Table 58-10](#).

Table 58-10 Limitations to Transport Header and Metadata Values You Specify in the Test Console

Transport	Service Type	Description of Limitation	Transport Headers Affected
HTTP(S) Note: When you test proxy services, the Test Console never sends a HTTP request over the network, therefore transport-level access control is not applied.	Proxy Services	All transport headers and other fields you set are preserved at runtime.	All

Table 58-10 (Cont.) Limitations to Transport Header and Metadata Values You Specify in the Test Console

Transport	Service Type	Description of Limitation	Transport Headers Affected
HTTP(S)	Business Services	The Service Bus runtime overrides any values you set for these parameters.	Content-Length Content-Type relative-URI client-host client-address
JMS	Proxy Services	The same limitations apply as those for a transport header action configuration.	See the limitations for JMS transport headers described in Table 12-8 .
JMS	Business Services	The same limitations apply as those for a transport header action configuration.	See the limitations for JMS transport headers described in Table 12-8 .
email	Proxy Services	No limitations. Any transport headers and other fields you set are honored at runtime.	None
email	Business Services	The Service Bus runtime overrides any values you set for these parameters	Content-Type
File	Proxy Services	No limitations. Any transport headers and other fields you set are honored at runtime. For example, <code>FileName</code> (Transport metadata)—the value you assign is appended to the output file name. For example, 1698922710078805308- b3fc544.1073968e0ab.-7e8e- { <code>FileName</code> }.	None
File	Business Services	No limitations	None
FTP	Proxy Services	No limitations. Any transport headers and other fields you set are honored at runtime.	None
FTP	Business Services	No limitations	None

Attachment Test Console Properties

Use the Attachment section of the Test Console to include attachments as part of the test input. [Table 58-11](#) described the properties in this section.

Table 58-11 Proxy Service Test Console Properties

Property	Description
Content-Type	Enter globally unique reference that identifies the attachment.

Table 58-11 (Cont.) Proxy Service Test Console Properties

Property	Description
Content-ID	Enter the media type and sub-type of the attachment.
Content-Description	Enter brief description of the content.
Content-Disposition	Specify how the attachment should be handled by the proxy service.
Content-Transfer-Encoding	Specify how the attachment is encoded.
File	Browse to and select the file to test as an attachment.
Resp. Attach. Display Limit	The number of characters of the attachment to include in the display of the test results.

Deploying Oracle Service Bus Services

This chapter provides instructions for deploying Service Bus projects and applications to an Oracle WebLogic Server. You can deploy Service Bus components from the Oracle Service Bus Console, JDeveloper, Fusion Middleware Control, and the Service Bus Deployment API. When you deploy to and from different environments, you can use customization files to update environment values like endpoint URIs.

This chapter includes the following sections:

- [Deployment Overview](#)
- [Before You Deploy](#)
- [Deploying from the Oracle Service Bus Console](#)
- [Deploying from JDeveloper](#)
- [Deploying a Service Bus Configuration JAR File in Fusion Middleware Control](#)
- [Updating an Online Configuration](#)
- [Updating an Online Configuration in a Cluster](#)

Deployment Overview

Deployment is the process of packaging Service Bus resources and transferring them to a target application server. You can deploy resources in multiple ways, including the following:

- Activate the current session in the Oracle Service Bus Console.
- Deploy a project or application using the Deploy command in JDeveloper.
- Export a configuration JAR file to an application server in JDeveloper. You can also deploy to the integrated application server included with JDeveloper to run, debug, and test applications and projects.
- Import a configuration JAR file using Fusion Middleware Control.
- Use the Maven plug-in to deploy Service Bus services in a Maven environment. For more information, see [Using the Oracle Service Bus Development Maven Plug-In](#).
- Use WLST commands to activate sessions and to import configurations and environment values. For more information, see [Using the Oracle Service Bus Deployment APIs](#) in *Administering Oracle Service Bus*.

Once you deploy Service Bus resources, you can monitor and manage them in the runtime using Fusion Middleware Control. For information, see [Oracle Service Bus Runtime Management](#) in *Administering Oracle Service Bus*.

Before You Deploy

Regardless of which method you use to deploy Service Bus configurations, review the information in the following sections before deploying.

- [Creating a Service Bus Domain Using the Configuration Wizard](#)
- [Resolving Conflicts](#)
- [Configuring JMS Resources](#)
- [Configuring Security](#)

Creating a Service Bus Domain Using the Configuration Wizard

Before you can deploy a Service Bus configuration, you need to have a running Service Bus domain, to which Service Bus resources will be deployed. The domain must have the required schemas in a running database. For information on creating a Service Bus domain with the Oracle Fusion Middleware Configuration Wizard, see [Configuring the Oracle Service Bus Domain in *Installing and Configuring Oracle Service Bus*](#).

Resolving Conflicts

If there are any conflicts in the Service Bus resources being deployed, the deployment will fail. Before activating resources or exporting them, view and resolve any existing conflicts as described in [Viewing and Resolving Conflicts](#). Deploying from JDeveloper or activating from the console will fail if there are any conflicts in Service Bus resources.

Configuring JMS Resources

In addition to configuring JMS file stores in the Oracle Fusion Middleware Configuration Wizard, proxy services and business services that use JMS require the following resources:

- **JMS connection factories:** You must configure XA or non-XA JMS connection factories for all business services and proxy services implemented using JMS.
- **JMS queues/topics:** Service Bus automatically configures JMS queues for proxy services that are implemented using JMS if they queues are on the same local Service Bus domain. You must configure JMS queues/topics for all business services using JMS, for proxy services that consume messages from a remote queue, and for proxy services that are implemented using non-JMS.

To concentrate all Service Bus JMS resources in a single JMS module, use the WebLogic Server Administration Console to create a new JMS module containing the destination to be used for the proxy services' endpoint. For more information about configuring JMS resources, see "Methods for Configuring JMS Resources" in [Administering JMS Resources for Oracle WebLogic Server](#).

Configuring Security

Service Bus leverages the security features of WebLogic Server to ensure message confidentiality and integrity (message-level security), secure connections between clients and WebLogic Server (transport-level security), and authentication and

authorization (access control). For information on how to configure security for Oracle Service Bus, see [Security](#)

Note:

You must configure security separately for each Service Bus domain. Service Bus does not export or import security configurations.

Deploying from the Oracle Service Bus Console

In the Oracle Service Bus Console, you deploy the services you create by activating your changes to the runtime. Any changes you activate become immediately available in the runtime environment. You can also update components that are activated in the runtime using the Oracle Service Bus Console.

How to Deploy from the Console

Once you have configured your Service Bus domain, secured it, and added any JMS resources required for its services, you are ready to import the JAR file that contains your Service Bus configuration. After you import the configuration metadata, you can update environment-specific information for your domain. All changes to Service Bus configurations require a session, with the exception of security-related changes.

To deploy the contents of configuration JAR file:

1. Create a session in Service Bus.

See [How to Create a Session](#).

2. Import all or selected objects from a configuration JAR file.

See [How to Import Resources from a Configuration JAR File in the Console](#).

3. Update environment-specific information such as service endpoint URIs and directory names.

See "Finding and Replacing Environment Values Using the Oracle Service Bus Console" or "Using Configuration Files to Update Environment Values and Operational Settings" in *Administering Oracle Service Bus*.

4. Verify there are no conflicts (indicated by a **Conflict** icon in the toolbar).

If there are conflicts, resolve them as described in [Viewing and Resolving Conflicts](#).

5. Activate the session.

See [How to Activate a Session](#).

You can also import and update a configuration programmatically using the WebLogic Scripting Tool (WLST) and the Service Bus `deploymentMBean`. For more information, see "Using the Oracle Service Bus Deployment APIs" in *Administering Oracle Service Bus*.

Deploying from JDeveloper

The first time you deploy an application or project to the WebLogic Server, the entire application and all projects are published. On subsequent deployments of the

application, only the resources that were changed are published to the server. If there are any conflicts in the any Service Bus resources in the application, the deployment will fail.

How to Create a Connection to the WebLogic Server

In JDeveloper, you must create a connection to the application servers to which Service Bus applications will be deployed. You only need to perform this task once for each application server hosting Service Bus services. You can connect to the following types of servers:

- **Standalone Server:** A server that is not managed by JDeveloper. Standalone servers include the domains you create using the configuration wizard. Only the Deploy command can be used to deploy to a standalone server, and the server must already be running in order to deploy to it.
- **Integrated Server:** A server that can be managed by JDeveloper, and only used in development and testing. If you opt to allow JDeveloper to manage the lifecycle of the server when you set up the connection, you can deploy to the server directly from JDeveloper using the Run command or the Deploy command. If JDeveloper does not manage the lifecycle, you need to start and stop the server manually. In that case, the Run command can only be used for deployment if the server is already running.

To create an application server connection:

1. If the Application Servers navigator is not visible, click the **Window** main menu, and select **Application Servers**.
2. In the Application Servers navigator, right-click **Application Servers** and select **New Application Server**.

The Create Application Server Connection wizard appears.

3. Select either **Standalone Server** or **Integrated Server**.
4. Click **Next**.
5. Do one of the following:
 - If you selected **Standalone Server**: Enter a name for the connection. Leave the connection type at its default value (WebLogic 12.x).
 - If you selected **Integrated Server**: Enter the connection name. If you want to be able to start the server in Run and Debug mode from JDeveloper, select **Let JDeveloper manage the lifecycle for this Server Instance**, and enter the location of the domain and server instance.
6. Click **Next**.
7. On the Authentication page, enter the user name and password to connect to the WebLogic server, and then click **Next**.
8. On the Configuration page, enter the following information:
 - **WebLogic Hostname (Administration Server):** The name of the server on which the WebLogic server resides.
 - **Port:** The port number used by the WebLogic server.

- **SSL Port:** The SSL port number used by the WebLogic server.
- **WebLogic Domain:** The name of the WebLogic server domain.

For additional information about specifying domains, click **Help**.

9. To use secure socket layer (SSL), select the **Always use SSL** check box.
10. Click **Next**.
11. On the Test page, click **Test Connection** to verify the server connection.

Note:

The test is only successful when performed against a running server.

12. If the connection is successful, click **Finish**. Otherwise, click **Back** to make corrections in the previous dialogs.

Even if the connection test is unsuccessful, a connection is created.

How to Create a Deployment Profile

A deployment profile provides JDeveloper with information about the application server to which a project or application will be deployed. The type of profile indicates whether to deploy only the selected project or the entire Service Bus application. Service Bus attaches a default project deployment profile to each Service Bus project you create, and a default application deployment profile to each Service Bus application. You can define additional deployment profiles, each of which deploy to a different application server.

For additional information about working with deployment profiles, see "How to Create and Edit Deployment Profiles" in *Developing Applications with Oracle JDeveloper*.

To create a deployment profile:

1. In the Application Navigator, right-click the Service Bus project.
2. Point to Deploy, and select **New Deployment Profile**.
The Create Deployment Profile dialog appears.
3. In the **Profile Type** field, do one of the following:
 - To define a profile that deploys just the selected project, select **Service Bus Project**.
 - To define a profile that deploys all projects in the application that contains the selected project, select **Service Bus Configuration**.
4. In the **Deployment Profile Name** field, enter a unique and identifying name for the profile.
5. Click **OK**.
6. Click **Save**.

How to Deploy a Service Bus Project or Application

You can deploy just the selected project to the WebLogic server, or you can deploy the entire application in which the selected project is located. Project-level deployments are not incremental and publish the full project and any of its dependencies. When you deploy a project or application, you select the deployment profile to use for the process. The profile you use determines whether to deploy just the project or the entire application. The default profile generated for each project only deploys the project.

Note:

You can also publish to the server using the Run command, which deploys to the selected server and launches the Service Bus Test Console. Note that you can only use the Run command for integrated-type servers. For more information, see [Accessing the Test Console](#).

Before You Begin

Make sure you have a connection to the application server, as described in [How to Create a Connection to the WebLogic Server](#). If you do not want to use the default deployment profile, create a new profile, as described in [How to Create a Deployment Profile](#).

To deploy a project or application:

1. In the JDeveloper Application Navigator, right-click the project to deploy.
2. Point to **Deploy** and select the name of the deployment profile to use.

Make sure to select the correct profile. To deploy just the project, select a profile with a type of **Service Bus Project**. To deploy the entire application, select a profile configured with a type of **Service Bus Configuration**. You can view deployment profile configurations in the project properties.

3. On the Deploy wizard, select **Deploy to Service Bus Server**, and then click **Next**.
4. On the Select Server page, select the name of the application server connection for the server to which you are deploying.

If the server does not exist in the Application Servers list, click **Add an Application Server** and complete the Create Application Server Connection wizard, as described in [How to Create a Connection to the WebLogic Server](#).

5. To overwrite existing deployed artifacts select **Overwrite modules of the same name**.
6. Click **Next**.
7. Review the summary information for the deployment, and then click **Finish**.

You can view the progress of the deployment in the Deployment - Log window.

How to Deploy a Project or Application Using the Previous Configuration

After you have deployed a project once, you can skip the Deploy wizard in subsequent deployments if you want to use the same configuration. When you skip

the Deploy wizard, Service Bus uses the same deployment profile and same selections you made on the Deploy wizard when you deployed the project the previous time.

To deploy using the previous configuration:

1. In the JDeveloper Application Navigator, right-click the project to deploy.
2. Point to **Deploy** and select **Deploy to Service Bus Server**.

The project is deployed using the last configuration used to deploy the project. You can view the progress of the deployment in the Deployment - Log window.

What Happens When You Deploy Using JDeveloper

If deployment is successful, the deployed project appears in the following locations:

- The Resources window under **Application Server** > *server_connection_name* > **Service Bus** > *Service_Bus_server_name*.
- The Application Server Navigator under *server_connection_name* > **Service Bus** > *Service_Bus_server_name*.

You are now ready to monitor your application from Oracle Enterprise Manager Fusion Middleware Control. For information, see Introduction to the Management and Monitoring Pages in *Administering Oracle Service Bus*.

If deployment is unsuccessful, view the messages that appear in the Deployment log window and take corrective actions.

Deploying a Service Bus Configuration JAR File in Fusion Middleware Control

You can deploy Service Bus configuration JAR files by importing the files into Fusion Middleware Control. Configuration JAR files contain Service Bus resources that were previously exported from a different Service Bus environment. The export and import features let you easily move projects and resources between environments. When you import a JAR file, you can also import a separate configuration file that updates operational settings and environment values in the imported resources to match the new environment. For example, a configuration file can update host names and port numbers, enable or disable monitoring for a service, and so on.

For information about importing resources into Fusion Middleware Control, see "Importing Oracle Service Bus Resources in Fusion Middleware Control" in *Administering Oracle Service Bus*. For information about environment configuration files, see "Using Configuration Files to Update Environment Values and Operational Settings" in *Administering Oracle Service Bus*.

Updating an Online Configuration

Once a configuration is deployed, Service Bus allows you to dynamically change the configuration information for a system without the need to restart the server for changes to take affect. You can change a resource, a project, or a number of resources (related or unrelated) using procedure outlined in [How to Deploy from the Console](#). In step 2, you can modify resources directly in the console, or import all or a subset of objects from a configuration JAR file.

The changes are consolidated and sent to all servers (administration and Managed Servers, if you are working in a cluster environment). These changes update the

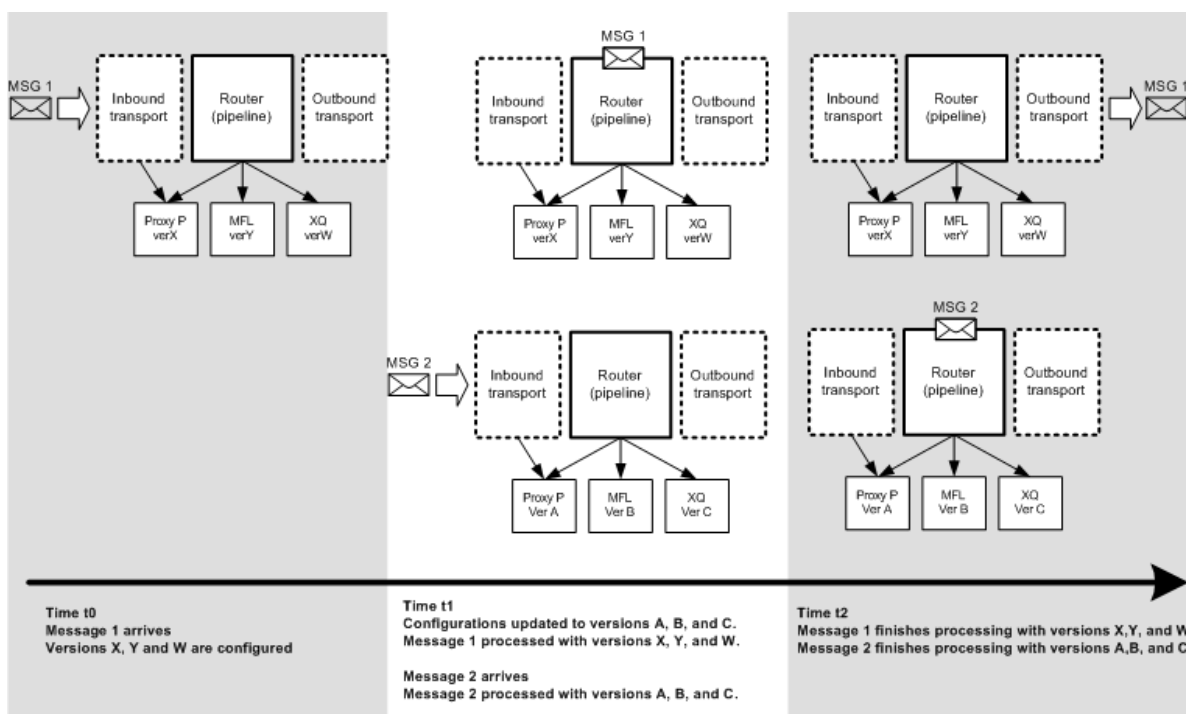
persisted configuration data and also cause other runtime tasks to be performed (such as creating proxy services and JMS queues, compiling XQueries, and so on).

Figure 59-1 illustrates how the system processes messages in the event that the configuration is updated while messages are being processed through the system. Table 59-1 describes the versions for the resources for the sample system illustrated in Figure 59-1.

Table 59-1 Initial and Updated Configuration for a Sample System

Resource	Initial Version	Updated Version
Proxy Service	X	A
MFL	Y	B
XQuery	W	C

Figure 59-1 Sample Online Update Scenario



Note the following characteristics of the message processing illustrated in the preceding figure:

- Message 1 is already in the system at t1 (the time the configuration is updated)
- Message 1 completes processing by the original (pre-update) resources (X, Y, W)
- Message 2 starts and completes processing with the new configuration (resources A, B, C)

Service Bus tries to execute messages with the version of the proxy service and artifacts available when the messages enters the proxy service. This ensures that a message has a consistent view of the artifacts. If the message processor cannot guarantee this behavior for a message, it will reject the message rather than process it incorrectly.

What You Need to Know for Successful Online Configuration Updates

This section describes guidelines to follow and limitations to be aware of when you update a configuration in a running Service Bus system.

- If you are concerned about message rejection by Service Bus, use the JMS transport protocol with retries. With retries, any messages that are rejected because the system cannot guarantee their processing by compatible resources will be retried.
- Update security-related configuration first, and then update the Service Bus resources in your system. Security configuration changes must be made at the WebLogic Server level before they are visible to Service Bus, especially for activation purposes. For instance, you must configure SSL-related options at the domain level (enable SSL port, configure identity and trust for the domain, etc.) before you can enable a proxy service to use SSL. To learn about updating security resources, see *Overview of Security Management in Administering Security for Oracle WebLogic Server*.
- Updates must be compatible with existing clients using the system. See [Changing an Online Proxy Service](#).
- If you are updating the configuration to a cluster, it is possible that the updates are done at different times on different Managed Servers. Consequently, messages could be processed by different versions of a proxy service, depending on which Managed Server gets the message to process. This depends on load balancing across Managed Servers.
- During online deployment, Service Bus checks whether the correct versions of referenced resources are used for message processing. If this is temporarily not true, an error is returned. However, if the interface artifact of an invoked service changes (for example, an MFL or WSDL file), the invoking proxy service may not return an error although it temporarily sees a version of the artifact that does not correlate with the proxy service version.

Changing an Online Business Service

Enterprise information services (EIS) are sometimes phased out, and new instances (possibly with new versions of EIS software, new hardware, and so on) are brought online. When this happens, Service Bus administrators need to gracefully transition to the new EIS instance by modifying any affected Service Bus business services.

For information about using the Oracle Service Bus Console to change an endpoint URI for a business service, see [How to Configure a Business Service Transport](#).

Changing an Online Proxy Service

While the majority of the metadata that defines a proxy service can be deployed without change in a new environment, there is some information you may need to update. For example, definitions of proxy services for File, FTP, and email message types must specify a single Managed Server for deployment of polling runtime components in a cluster.

As your business requirements change, you may need to make changes to your proxy services. If the changes you need to make are backward compatible, you can dynamically make changes online using the Oracle Service Bus Console to create a new version of the proxy service. Changes are backward compatible if they meet one of the following criteria:

- The interface of the changed object is unchanged.
- Old and new clients will work with the interface.

If the changes you need to make are not backward compatible, there are two alternatives to consider that would enable you to make the changes online:

- Create and deploy a new proxy service having a different name and URL from that of the earlier version. Clients upgrade by accessing the new proxy service. This enables you to run the old and new versions of a proxy service in parallel, and supports a gradual migration to the new proxy service.
- Force backwards compatibility by changing the proxy service interface to support both the new interface and the old interface (for example, using XML schema choice) and perform different logic in the message flow based on the document received. Clients continue to access the proxy service by using its original URL.

Service Bus cluster domains have additional system administration requirements for deployment of proxy services that are not backward compatible. For more information, see [Installing a New Version of a Proxy Service in a Cluster](#).

Changing an Online Pipeline

Pipelines route messages to named destinations, such as business services, other proxy services, and so on. Message routing definitions may need to be updated in a new environment.

Updating an Online Configuration in a Cluster

Updating Service Bus components in a cluster requires some additional considerations to those for a non-clustered environment. For information about performing online updates, see [Updating an Online Configuration](#).

Changing a Business Service in a Cluster

The procedure for changing a business service is the same in both non-clustered and cluster environments. However, the procedure for deploying changes to a business service in a cluster depends on the types of changes made to the business service and the nature of any other changes that might be deployed simultaneously. For more information, see the description of installation strategies in the following section.

For information about changing a business service, see [Changing an Online Business Service](#).

Installing a New Version of a Proxy Service in a Cluster

You can make changes to proxy services dynamically online, partially offline, or completely offline. If your changes are backward compatible (that is, you are making no changes to interfaces), you can make your changes dynamically online using the Oracle Service Bus Console. Making other types of changes should be done partially or completely offline, which requires additional system administration steps.

Making changes that include non-backward compatible changes to proxy service interfaces requires complete offline deployment. To install the new version, follow the procedure below while all servers are operational:

1. Quiesce all inbound messages.

2. Confirm all asynchronous backlogged messages have been processed.
3. Make the necessary changes in the proxy service, and test to verify the proxy service operates as required.
4. Resume accepting inbound messages.

For more information about backward compatibility and installation strategies, see [Changing an Online Proxy Service](#).

Using the Oracle Service Bus Development Maven Plug-In

This chapter describes how to use the Oracle Service Bus development Maven plug-in to build and manage Service Bus projects. The Oracle Service Bus development Maven plug-in lets you package and deploy Service Bus projects in a Maven environment.

This chapter includes the following sections:

- [Introduction to the Oracle Service Bus Maven Plug-In](#)
- [Installing and Configuring Maven](#)
- [Using the Oracle Service Bus Development Maven Plug-In](#)
- [Service Bus Development Maven Plug-In Goals](#)
- [Oracle Service Bus Development Maven Plug-In POM File Samples](#)

For more information about using Maven with Oracle Fusion Middleware, see *Developing Applications Using Continuous Integration*. For information on installing and using Maven to build applications and projects, see <http://maven.apache.org/users/index.html>.

Introduction to the Oracle Service Bus Maven Plug-In

Maven is a build automation tool that lets you create and manage runtime projects. The Oracle Service Bus development Maven plug-in provides Maven goals specific to the requirements of Service Bus projects and applications. You can use it to perform tasks such as packaging Service Bus projects or resources and deploying the package to a running server.

Maven Lifecycle Phases and Goals

Lifecycle phases give order to goal execution in a Maven POM file. A goal is a specific task, which can be mapped to one or more phases. The Oracle Service Bus development Maven plug-in provides goals that are specific to Service Bus projects and that can be used in conjunction with standard Maven goals to build and manage Service Bus projects and applications throughout the lifecycle phases.

Service Bus provides two custom goals, `package` and `deploy`. For information about the syntax and parameters for these goals, see [Service Bus Development Maven Plug-In Goals](#). [Table 60-1](#) lists the phases in the default Maven lifecycles for Service Bus, along with the goals mapped to each phase. If you map multiple goals to the same lifecycle phase, they are typically executed in the order you list them.

Table 60-1 Maven Lifecycle Phases for Service Bus

Phase	Description
package	Packages the Service Bus project resources in its distributable format, <i>sbar</i> (Service Bus archive file). The <code>com.oracle.servicebus.plugin:oracle-servicebus-plugin:12.1.3.0.0:package</code> goal is mapped to this phase.
pre-integration-test	Processes and deploys the package if necessary into an environment where integration tests can be run. The <code>com.oracle.servicebus.plugin:oracle-servicebus-plugin:12.1.3.0.0:deploy</code> goal is mapped to this phase.
install	Installs the package into the local repository, for use as a dependency in other projects locally. Service Bus uses the <code>org.apache.maven.plugins:maven-install-plugin:install</code> goal for this phase.

POM Files and Archetypes

Maven projects are defined by a POM file, which describes the project's artifact, the plug-ins to use, inheritance, and dependencies on other artifacts (such as system resources that are required to build Service Bus projects). An archetype is a template for creating a specific type of project. Service Bus provides two different archetypes, one for Service Bus projects and one for Service Bus system resources.

The Service Bus project archetype is named

`com.oracle.servicebus.archetype:oracle-servicebus-project:12.1.3-0-0`. It is defined by the following archetype coordinates:

```
<groupId>com.oracle.servicebus.archetype</groupId>
<artifactId>oracle-servicebus-project</artifactId>
<version>12.1.3-0-0</version>
<name>Oracle Service Bus - Project Archetype</name>
```

The Service Bus system resources archetype is named

`com.oracle.servicebus.archetype:oracle-servicebus-system:12.1.3-0-0`. It is defined by the following archetype coordinates:

```
<groupId>com.oracle.servicebus.archetype</groupId>
<artifactId>oracle-servicebus-system</artifactId>
<version>12.1.3-0-0</version>
<name>Oracle Service Bus - System Resources Archetype</name>
```

Installing and Configuring Maven

A distribution of Maven 3.0.5 is included with Oracle Fusion Middleware in the following location:

```
Middleware_Home/Oracle_Home/oracle_common/modules/org.apache.maven_3.0.5
```

For information about installing and configuring Maven for Oracle Fusion Middleware, see "Installing and Configuring Maven for Build Automation and Dependency Management" in *Developing Applications Using Continuous Integration*. Be sure to follow the setup instructions in Section 5.1, "Setting Up the Maven Distribution" and Section 5.2, "Customizing Maven Settings."

The Oracle Service Bus development Maven plug-in is installed in the following location:

```
Service_Bus_Home/plugins/maven
```

How to Configure the Oracle Service Bus Development Maven Plug-In

The `oracle-servicebus-plugin` plug-in is provided as a pre-built JAR file and accompanying POM file. You use the Maven synchronization plug-in to populate a local or shared Maven repository from an Oracle home, which includes the Service Bus plug-in. For more information and complete instructions for installing and running the synchronization plugin, see "Populating the Maven Repository Manager" in *Developing Applications Using Continuous Integration*. The steps below include links to more specific sections of that guide.

To configure the Oracle Service Bus development Maven plug-in:

1. Navigate to `ORACLE_HOME/oracle_common/plugins/maven/com/oracle/maven/oracle-maven-sync/12.1.3`.
2. Run the following command to install the Maven sync plug-in:

```
mvn install:install-file -DpomFile=oracle-maven-sync-12.1.3.pom -Dfile=oracle-maven-sync-12.1.3.jar
```

For more options, see "Installing Oracle Maven Synchronization Plug-In."

3. Run the following command to seed the Oracle Service Bus development Maven plug-in into the Maven repository:

```
mvn com.oracle.maven:oracle-maven-sync:push -DoracleHome=ORACLE_HOME
```

Where `ORACLE_HOME` is the full path to your Oracle Fusion Middleware installation. For more options, see "Running the Oracle Maven Synchronization Plug-In."

4. Validate whether you have successfully installed the plug-in using the Maven `help:describe` goal.

```
mvn help:describe -DgroupId=com.oracle.servicebus.plugin -DartifactId=oracle-servicebus-plugin -Dversion=12.1.3-0-0
```

The following information confirms the installation of the Service Bus plug-in:

```
Name: Oracle Service Bus - Plugin
Description: (no description available)
Group Id: com.oracle.servicebus.plugin
Artifact Id: oracle-servicebus-plugin
Version: 12.1.3-0-0
Goal Prefix: servicebus
This plugin has 2 goals:
servicebus:deploy
  Description: (no description available)
servicebus:package
  Description: (no description available)
For more information, run 'mvn help:describe [...] -Ddetail'
```

How to Use Maven Online Help

Maven online help provides you with a list of goals and their associated commands. Use the `describe` goal in the `help` plug-in to access online help. For example, enter the following command to obtain online help for the package goal:

```
mvn help:describe -Ddetail -Dcmd=com.oracle.servicebus.plugin:oracle-servicebus-plugin:package
```

See the Apache [help plug-in describe goal](#) documentation for additional information.

Using the Oracle Service Bus Development Maven Plug-In

You invoke Maven goals either through a Maven project POM file or from the command line. The preferred and recommended way is to use a Maven POM file, which can be created for both Service Bus projects and applications.

For more information about working with Maven in JDeveloper, see *Building and Running with Apache Maven in Developing Applications with Oracle JDeveloper*.

How to Generate a Service Bus Project POM File

When you create a Service Bus project in JDeveloper, a POM file is automatically created for that project. You can also manually generate a POM file for a Service Bus project using JDeveloper.

To generate a Service Bus project POM file in JDeveloper:

1. In the Application Navigator, right-click the project, point to **New** and select **From Gallery**.
2. Under the General category on the New Gallery dialog, select **Maven**.
3. Under Items, select **Maven POM for Project** and click **OK**.
4. Enter a name for the POM file and the directory where the POM will be located.
5. Enter a unique group ID, artifact ID and version for the project.

Together, these values form the fully qualified artifact name in the form of `<groupId>:<artifactId>:<version>`.

6. Enter a description for the project.
7. In the Packaging field, select **sbar**.
8. Clear or select **Use this POM as the default for the project**, depending on your requirements.
9. Click **OK**.

How to Generate a Service Bus Project POM File from an Archetype

You can generate a POM file for a project from the Service Bus *project* archetype in either JDeveloper or from a command line. Archetypes are templates for creating Maven projects.

Creating a Service Bus Project POM File from an Archetype in JDeveloper

To generate a Service Bus project POM file from an archetype in JDeveloper:

1. In the Application Navigator, right-click the project, point to **New** and select **From Gallery**.
2. Under the General category on the New Gallery dialog, select **Maven**.
3. Under Items, select **Generate from Archetype** and click **OK**.

The Create Project From Archetype dialog appears.

4. Enter a unique group ID, artifact ID and version for the project.

Together, these values form the fully qualified artifact name in the form of `<groupId>:<artifactId>:<version>`.

For more information about the properties on this dialog, see the online help and [Parameters for Generating a POM File](#). Note that these are not the *archetype* group and artifact IDs.

5. Enter the default package name and the directory where you want to store the project.
6. In the **Maven Archetype** field, click the browse icon to search for and select the `oracle-servicebus-project` archetype.

For information about searching for archetypes, see "How to Create Maven Projects Using Maven Archetypes" in *Developing Applications with Oracle JDeveloper*.

7. When you are done, click **Finish** on the Create Project From Archetype dialog.

Generating a Service Bus Project POM File from an Archetype Using a Command Line

To generate a Service Bus project POM file from an archetype using a command line:

1. From the directory for the Service Bus project for which you want to create the POM file, run the following command:

```
mvn archetype:generate
-DarchetypeGroupId=com.oracle.servicebus.archetype
-DarchetypeArtifactId=oracle-servicebus-project
-DarchetypeVersion=12.1.3-0-0
-DarchetypeRepository=Repository_Home
-DgroupId=Group_ID
-DartifactId=Artifact_ID
-Dversion=1.0-SNAPSHOT
```

Where:

- *Repository_Home* is the path to the Maven repository to use. Instead of entering the path, you can simply enter **local**, which tells Maven not to look in any other repository it knows about and is more efficient.
- *Group_ID* is a unique, identifying name for the project to build.
- *Artifact_ID* is the name of the subdirectory in the current directory in which the project artifacts are generated.

For more information about the parameters for this command, see [Parameters for Generating a POM File](#).

2. When the command line prompts you to confirm the properties configuration, type **Y** to confirm or **N** to cancel. Press **Enter**.

The POM file is generated in the directory from which you ran the command in the subdirectory specified by the artifact ID.

How to Generate a Service Bus System Resources POM File from an Archetype

You can generate a POM file for Service Bus system resources from the Service Bus *system* archetype in either JDeveloper or from a command line. Archetypes are templates for creating Maven projects.

Generating a Service Bus System Resources POM File from an Archetype in JDeveloper

To generate a Service Bus system resources POM file from an archetype in JDeveloper:

1. In the Application Navigator, right-click the project containing the system resources, point to **New** and select **From Gallery**.

Note:

For Service Bus system resources, the location in JDeveloper might be in **Service Bus System Resources** under **Application Resources**.

2. Under the General category on the New Gallery dialog, select **Maven**.
3. Under Items, select **Generate from Archetype** and click **OK**.

The Create Project From Archetype dialog appears.

4. Enter a unique group ID, artifact ID and version for the project.

Together, these values form the fully qualified artifact name in the form of `<groupId> : <artifactId> : <version>`.

For more information about the properties on this dialog, see the online help and [Parameters for Generating a POM File](#). Note that these are not the *archetype* group and artifact IDs.

5. Enter the default package name and the directory where you want to store the project.
6. In the **Maven Archetype** field, click the browse icon to search for and select the `oracle-servicebus-system` archetype.

For information about searching for archetypes, see "How to Create Maven Projects Using Maven Archetypes" in *Developing Applications with Oracle JDeveloper*.

7. When you are done, click **Finish** on the Create Project From Archetype dialog.

Generating a Service Bus System Resources POM File from a Command Line

To generate a Service Bus system resources POM file from a command line:

1. From the directory where the Service Bus system resources are located, run the following command:

```
mvn archetype:generate
-DarchetypeGroupId=com.oracle.servicebus.archetype
-DarchetypeArtifactId=oracle-servicebus-system
-DarchetypeVersion=12.1.3-0-0
-DarchetypeRepository=Repository_Home
-DgroupId=Group_ID
-DartifactId=Artifact_ID
-Dversion=1.0-SNAPSHOT
```

Where:

- *Repository_Home* is the path to the Maven repository to use. Instead of entering the path, you can simply enter **local**, which tells Maven not to look in any other repository it knows about and is more efficient.
- *Group_ID* is a unique, identifying name for the project to build.
- *Artifact_ID* is the name of the subdirectory in the current directory in which the project artifacts are generated.

For more information about these parameters, see [Parameters for Generating a POM File](#).

2. When the command line prompts you to confirm the properties configuration, type **Y** to confirm or **N** to cancel. Press **Enter**.

The POM file is generated in the directory from which you ran the command in the subdirectory specified by the artifact ID.

Parameters for Generating a POM File

When you generate a POM file from a command line, use the following parameters to configure the project.

Parameter	Description
<code>archetypeGroupId</code>	The group ID of the archetype to use. For Service Bus, this is <code>com.oracle.servicebus.archetype</code> .
<code>archetypeArtifactId</code>	The artifact ID of the archetype to use. For a Service Bus project, this is <code>oracle-servicebus-project</code> . For Service Bus system resources, this is <code>oracle-servicebus-system</code> .
<code>archetypeVersion</code>	The version of the archetype to use. The current version is <code>12.1.3-0-0</code> .
<code>archetypeRepository</code>	The Maven repository to use.
<code>groupId</code>	The group ID of the project to build.
<code>artifactId</code>	The artifact ID of the project to build.
<code>version</code>	The version of the project to build.

Service Bus Development Maven Plug-In Goals

The Oracle Service Bus development Maven plug-in provides two goals specific to Service Bus, which are described in the following sections:

- **package**: Packages the Service Bus project resources in its distributable format, *sbar* (Service Bus archive file).
- **deploy**: Deploys Service Bus projects and applications to a running server. This goal supports files in the SBAR format.

package

Full Name

```
com.oracle.servicebus.plugin:oracle-servicebus-plugin:package
```

Description

The package goal creates a configuration JAR file from the resources associated with a POM file, and packages the resources into a Service Bus-specific archive file known as an *.sbar* file. By default, the Maven plug-in assumes the resources being packaged are project resources, but a Service Bus application can also include system resources, which are shared among projects. System resources are packaged differently than project resources, so when you package system resources, you need to set the `system` flag to true.

The Maven plug-in uses the offline export (configuration JAR) tool to package the Service Bus resources. It places the temporary files created by the export tool in `project/.data/maven/configjar`. The settings for the export tool are derived from the Maven project context. When packing project resources, Service Bus performs the export at the project level; when packaging system resources, Service Bus performs the export at the resource level. It uses the default extension mappings. For more information, see [Exporting a Service Bus Configuration Offline](#). The export tool writes information to a log file located at `Project_Home/.maven/configjar/configjar.log`.

The directories that contain Service Bus resources often contain additional files that you might want to exclude from the generated package. For example, you can exclude metadata files used by the versioning system. You can define an exclusion list to make sure these files are not included when the SBAR file is generated. By default, the following files and folders are excluded for the project archetype: `servicebus.sboverview`, `pom.xml`, `.settings/`, and `.data/`. For the system archetype, `pom.xml` and `.data/` are excluded.

Validation errors that occur during packaging are not reported, and packaging does not fail if there are validation errors.

Parameters

Use the following parameters to customize the packaging process.

Table 60-2 Parameters for `servicebus:package` Goal

Name	Type	Description
<code>oracleHome</code>	<code>java.lang.String</code>	Specifies the location to the Oracle Fusion Middleware home directory. You can specify this value as an expression.

Table 60-2 (Cont.) Parameters for servicebus:package Goal

Name	Type	Description
system	java.lang.Boolea n	Specifies whether the resources being packaged are system resources, which are shared by multiple projects within a Service Bus application. The default value is false. You must set this value to true when packaging system resources.
excludes	String[]	Specifies a list of files to exclude from the project. Use this to exclude things like versioning system files.

deploy

Full Name

com.oracle.servicebus.plugin:oracle-servicebus-plugin:deploy

Description

The deploy goal deploys Service Bus projects to a running server. This goal supports the Service Bus deployment format, SBAR. It does not require a local server installation. By default, deploying projects does not apply any updates to environment values. If you want to update the environment values, you can create a configuration file with the new environment values and specify that configuration file when you run deploy.

For each deployment, a new session is created and named in the following format:

Service_Bus_Maven-artifactId-currentTime

The current time extends to milliseconds.

When you package a Service Bus project using Maven, Service Bus generates a configuration JAR file which can then be deployed to a running server using the deploy goal. Service Bus uses the default import plan, which includes dependencies and does not preserve environment variables, security settings, or credentials.

If any resources fail to import, the information is logged and the goal execution fails. If all resources import successfully and you specified a configuration file, Service Bus applies the new environment variables. Once the deployment completes successfully (that is, there are no conflicts in any of the resources), Service Bus activates the session. In case of any failures or conflicts, Service Bus does not activate the changes or apply the new environment variables, but it does retain the session so you can research the failures.

Parameters

Use the following parameters to customize the deployment process. You can specify any of the parameters using an expression.

Table 60-3 Parameters for servicebus:deploy Goal

Name	Type	Description
oracleServerUrl	java.lang.String	Specifies the address and port on which the Administration Server is listening. The default value is: t3://localhost:7001
oracleUsername	java.lang.String	Specifies the administrative user name.
oraclePassword	java.lang.String	Specifies the administrative password.
customization	java.io.File	Specifies the location and name of a Service Bus configuration file that will update environment values for the environment in which the Service Bus archive is being deployed.

Oracle Service Bus Development Maven Plug-In POM File Samples

Service Bus has three different types of POM files, one for Service Bus applications, one for Service Bus projects, and one for Service Bus system resources. The one you use depends on the resources you are working with.

Example - Service Bus Application POM File

Below is an example of a Service Bus application POM file. This is an aggregation file that lists all the projects to compile in an application, allowing you to run Maven against a single POM file instead of each individual project POM file. Maven executes the modules in the order in which they are listed in the application POM file.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd"
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>

  <groupId>OrdersAndPayments</groupId>
  <artifactId>OrdersAndPayments</artifactId>
  <version>1.0-SNAPSHOT</version>

  <packaging>pom</packaging>

  <modules>
    <module>System</module>
    <module>Orders</module>
    <module>Payments</module>
  </modules>

</project>
```

Example - Service Bus Project POM File

Below is an example of a POM file for a Service Bus project. This file inherits from a parent POM named `project-12.1.3` in the Maven repository. With the inheritance, the project POM file is more streamlined.

Note that "System" is a reserved project name in Service Bus specifically for system resources. This should not be used as the artifact ID when using the project archetype.

```
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd"
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>com.oracle.servicebus</groupId>
    <artifactId>project</artifactId>
    <version>12.1.3.0.0</version>
  </parent>

  <groupId>OrdersAndPayments</groupId>
  <artifactId>Orders</artifactId>
  <version>1.0-SNAPSHOT</version>

  <packaging>sbar</packaging>

  <dependencies>
    <dependency>
      <groupId>OrdersAndPayments</groupId>
      <artifactId>System</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
  </dependencies>

</project>
```

Example - Service Bus System Resources POM File

Below is an example of a POM file for Service Bus system resources. This file inherits from a parent POM named `system-12.1.3` in the Maven repository. With the inheritance, the system resources POM file is more streamlined.

"System" is a reserved project name in Service Bus specifically for system resources. The artifact ID must be "System" when using the system archetype.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd"
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>com.oracle.servicebus</groupId>
    <artifactId>system</artifactId>
    <version>12.1.3.0.0</version>
  </parent>

  <groupId>OrdersAndPayments</groupId>
  <artifactId>System</artifactId>
  <version>1.0</version>

  <packaging>sbar</packaging>

</project>
```


Part X

Appendixes

This part contains miscellaneous development information and reference material.

This part contains the following appendixes:

- [Message Context](#)
- [XPath Extension Functions](#)
- [Oracle Service Bus APIs](#)
- [Transport SDK Interfaces and Classes](#)
- [Transport SDK UML Sequence Diagrams](#)
- [XQuery-SQL Mapping Reference](#)
- [Work Managers and Threading](#)

Message Context

This appendix describes the Service Bus message context model and the predefined context variables that are used in message flows.

This chapter includes the following sections:

- [The Message Context Model](#)
- [Predefined Context Variables](#)
- [Message-Related Variables](#)
- [Inbound and Outbound Variables](#)
- [Operation Variable](#)
- [Fault Variable](#)
- [Initializing Context Variables](#)
- [Performing Operations on Context Variables](#)
- [Constructing Messages to Dispatch](#)
- [Message Context Schema](#)

The Message Context Model

The Service Bus message context is a set of properties that hold message content as well as information about messages as they are routed through Service Bus. These properties are referred to as context variables; for example, service endpoints are represented by predefined context variables. Service Bus also supports user-defined context variables.

The message context is defined by an XML schema. You typically use XQuery expressions to manipulate the context variables in the pipeline service.

Predefined Context Variables

[Table A-1](#) describes the predefined context variables. The predefined context variables can be grouped into the following types: message-related variables, inbound and outbound variables, the `$operation` variable, and the fault variable.

Note:

The Message Context Schema specifies the element types for the message context variables.

For information about the element types in message context variables, see [Message Context Schema](#).

Table A-1 *Predefined Context Variables in Service Bus*

Context Variable	Description	See Also
header	For SOAP messages, <code>\$header</code> contains the SOAP header. If the pipeline is SOAP 1.2, <code>\$header</code> contains a SOAP 1.2 Header element. For message types other than SOAP, <code>\$header</code> contains an empty SOAP header element.	Message-Related Variables
body	This varies depending on the message type, as described below: <ul style="list-style-type: none">• SOAP messages: The <code><SOAP:Body></code> part extracted from the SOAP envelope. If the pipeline is SOAP 1.2, the <code>\$body</code> variable contains a SOAP 1.2 Body element.• Non-SOAP, non-binary messages: The entire message content wrapped in a <code><SOAP:Body></code> element.• Binary messages: A <code><SOAP:Body></code> wrapped reference to an in-memory copy of the binary message.• Java objects: A <code><SOAP:Body></code> wrapped reference to an in-memory copy of the Java object.	Message-Related Variables
attachments	The MIME attachments for a given message.	Message-Related Variables
inbound	The inbound transport headers along with information about the proxy service that received a message.	Inbound and Outbound Variables
outbound	The outbound transport headers along with information about the target service to which a message is to be sent.	Inbound and Outbound Variables
operation	The operation being invoked on a pipeline.	Operation Variable
fault	Information about errors that have occurred during the processing of a message.	Fault Variable
messageId	The transport provider-specific message identifier. This ID should uniquely identify the message among other messages going through the Service Bus runtime, but it is not required that this value be unique.	messageID Variable

Message-Related Variables

Together, the message-related variables `$header`, `$body`, and `$attachments` represent the canonical format of a message as it flows through Service Bus. These variables are initialized using the message content received by a pipeline and are used to construct the outgoing messages that are routed or published to other services. If

you want to modify a message as part of processing it, you must modify these variables.

The message payload (that is, a message content exclusive of headers or attachments) is contained in the `$body` variable. The decision about which variable's content to include in an outgoing message is made at the point at which a message is dispatched (published or routed) from Service Bus. That determination is dependent upon whether the target endpoint is expecting a SOAP or a non-SOAP message:

- When a SOAP message is expected, the `$header` and `$body` variables are combined in a SOAP envelope to create the message.
- When a non-SOAP message is expected, the contents of the `Body` element in the `$body` variable constitutes the entire message.
- In either case, if the service expects attachments, a MIME package is created from the resulting message and the `$attachments` variable.

Header Variable

The `$header` variable contains SOAP headers associated with a message. The `$header` variable points to a `<SOAP:Header>` element with headers as sub-elements. Note that if the proxy service is SOAP 1.2, the `$header` variable contains a SOAP 1.2 Header element. In the case of non-SOAP messages or SOAP messages with no headers, the `<SOAP:Header>` element is empty, with no sub-elements.

Body Variable

The `$body` variable represents the core message payload and always points to a `<SOAP:Body>` element. Note that if the proxy service is SOAP 1.2, `$body` contains a SOAP 1.2 Body element. The core payload for both SOAP and non-SOAP messages is available in the same variable and with the same packaging; that is, wrapped in a `<SOAP:Body>` element:

- In the case of SOAP messages, the SOAP body is extracted from the envelope and assigned to the `$body` variable.
- In the case of non-SOAP, non-binary, messages, the full message contents are placed within a newly created `<SOAP:Body>` element.
- In the case of binary messages, rather than inserting the message content into the `$body` variable, a `<binary-content />` reference element is created and inserted into the `<SOAP:Body>` element. To learn how binary content is handled, see [Binary Content in the Body and Attachments Variables](#).
- In the case of Java objects, a `<java-content />` reference element is created and inserted into the `<SOAP:Body>` element. To learn how Java content is handled, see [Java Content in the Body Variable](#).

Attachments Variable

The `$attachments` variable holds the attachments associated with a message. The `attachments` variable is defined by an XML schema. It consists of a single root node, `<ctx:attachments>`, with a `<ctx:attachment>` sub-element for each attachment. The sub-elements contain information about the attachment (derived from MIME headers) as well as the attachment content and any custom headers for the attachment.

As with most of the other message-related variables, `$attachments` is always set, but if there are no attachments, the `$attachments` variable consists of an empty

<ctx:attachments> element. The \$attachments variable contains the following for each attachment:

- The attachment, if the attachment is XML.
- A reference XML, if the attachment is binary.
- Text, if the attachment is text.

Note:

The Message Context Schema specifies the element types for the message context variables.

Each attachment element includes the set of sub-elements described in [Table A-2](#).

Table A-2 Sub-Elements of the Attachments Variable

Elements of the Attachments Variable	Description
Content-ID	A globally-unique reference that identifies the attachment. The type is <code>string</code> .
Content-Type	The media type and sub-type of the attachment. The type is <code>string</code> .
Content-Transfer-Encoding	An indicator of how the attachment is encoded. The type is <code>string</code> .
Content-Description	A textual description of the content. The type is <code>string</code> .
Content-Location	A locally-unique URI-based reference that identifies the attachment. The type is <code>string</code> .
Content-Disposition	An indicator of how the attachment should be handled by the recipient. The type is <code>string</code> .
user-headers	A list of custom MIME headers for the attachment. This element can contain one or more <code>user-header</code> elements that define each custom header.
user-header	A custom MIME header. Specify the header's name in the <code>name</code> attribute, and the header's value in the <code>value</code> attribute. The type for both attributes is <code>string</code> .
body	Holds the attachment data. The type is <code>anyType</code> .

With the exception of the untyped `body` element, all other elements contain string values that are interpreted in the same way as they are interpreted in MIME. For example, valid values for the `Content-Type` element include `text/xml` and `text/xml; charset=utf-8`.

The parsing of attachments is not recursive. If an attachment has a `Content-Type` of `multipart/...`, the `body` element holds the original unpacked MIME content as a stream of bytes and does not contain attachment sub-elements. Because the MIME stream may contain binary data, it is represented by a `<binary-content>` reference

element. To learn how binary content is handled, see [Binary Content in the Body and Attachments Variables](#).

Messages whose `Content-Type` is `multipart/form-data` are constructed at runtime as follows:

- Inbound: All parts of a received inbound `multipart/form-data` type message are assigned to the `$attachments` variable. The `$body` variable is left empty.
- Outbound: The content of an outbound `multipart/form-data` type message is built from the content of the `$attachments` variable. Nothing from `$header` or `$body` is included.

Note:

If the inbound message is of a different `multipart` type than `multipart/form-data` (for example, `multipart/related`) and the outbound message is `multipart/form-data`, you must explicitly preserve the headers and content of the inbound root part, because they will not otherwise be passed through.

Service Bus does not support sending attachments to EJB, Tuxedo, and DSP services.

Message Types and Context Variables

The context variables are wrapper variables that contain the SOAP header elements, the SOAP body element, and the MIME attachments, respectively. The context gives the impression that all messages are SOAP messages, and non-SOAP messages are mapped to this paradigm. The following table lists the mappings for different message types. For information about Java content, see [Java Content in the Body Variable](#).

Table A-3 Message Mappings

Message Type	Mapping
XML	The <code>Body</code> element in <code>\$body</code> contains the XML document. Attachments are in <code>\$attachments</code> .
binary	The <code>Body</code> element in <code>\$body</code> contains a reference XML document. Attachments are in <code>\$attachments</code> .
MFL	The document is transparently converted from and to XML, and appears as an XML document in the <code>Body</code> element in <code>\$body</code> . Attachments are in <code>\$attachments</code> .
text	The <code>Body</code> element in <code>\$body</code> contains the text. Attachments are in <code>\$attachments</code> .
File, FTP, and Email	In the case of pass-by-reference documents, a reference XML document in the <code>Body</code> element in <code>\$body</code> refers to the URI of the document stored in the file system by the transport. Attachments are in <code>\$attachments</code> .
SOAP	The <code>Body</code> element in <code>\$body</code> contains the SOAP body. The <code>Header</code> element in <code>\$header</code> contains the SOAP header. Attachments are in <code>\$attachments</code> .

Binary Content in the Body and Attachments Variables

In the case of both the `$body` and `$attachments` variables, `text`, `XML`, and `MFL` content is placed directly inside an XML element. For binary data, which can contain byte values that are illegal in XML, Service Bus does not place the binary content in the XML element. Consequently, the binary content cannot be manipulated, but it is handled efficiently.

When binary content is received, the Service Bus runtime stores it in an in-memory hash table and a reference to that content is inserted into the XML (`body` or `attachments`) element. This reference is represented by the following XML snippet:

```
<binary-content ref="..." />
```

where the `ref` attribute contains a URI or URN that uniquely identifies the binary content. This XML can be manipulated in a pipeline pair, branch, or route node in the same way any other content can be manipulated, but only the reference and not the underlying binary content is affected.

For example:

- Binary content in the `$body` variable can be copied to an attachment by copying the reference XML to the `body` sub-element of an attachment element.
- Binary content in two different attachments can be swapped by swapping the snippets of reference XML or by swapping the values of the `ref` attributes.

When messages are dispatched from Service Bus, the URI in the reference XML is used to restore the relevant binary content in the outgoing message. For information about how outbound messages are constructed, see [Constructing Messages to Dispatch](#).

Clients and certain transports, notably email, file, and FTP can use this same reference XML to implement pass-by-reference. In this case, the transport or client creates the reference XML rather than the proxy service runtime. Also, the value of the URI in the `ref` attribute is specified by the user who creates the reference XML. For these cases in which the reference XML is not created by the proxy service runtime—specifically, when the URI is not recognized as one referring to internally managed binary content—Service Bus does not de-reference the URI, and the content is not substituted into an outgoing message.

Sending SOAP with Attachments to Business Processes

When you send a SOAP with Attachments (SwA) document from Service Bus to a BPEL business process and use the XPath function `ora:getAttachmentContent('inputVariable','bin','/bin')` to retrieve the multipart attachments in BPEL, make sure to do the following:

- Set the `Content-ID` attachment variable for the multipart attachments.
- Refer to the attachments in the primary soap envelope using an `href` attribute.

Note that in some cases, the `cid:` information might be absent from the `href` attribute.

Below is an example:

```
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
    start="<rootpart@example.com>"
Content-Description: This is the optional message description.
--MIME_boundary
Content-Type: text/xml; charset=UTF-8
```

```

Content-Transfer-Encoding: 8bit
Content-ID: <rootpart@example.com>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body xmlns:types="http://example.com/mimetypes">
    <m:SendClaim xmlns:m="http://example.com/mimewsdl">
      <ClaimDetail>
        <Name>...</Name>
        <!-- additional claim details -->
      </ClaimDetail>
      <ClaimPhoto href="cid:4d7a5fa2-14af-451c-961b5c3abf786796@example.com"/>
    </m:SendClaim>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <4d7a5fa2-14af-451c-961b5c3abf786796@example.com>

...MIME attachment of binary photograph...
--MIME_boundary-

```

Java Content in the Body Variable

The Service Bus pipeline supports Java objects as inputs and outputs to Java callout actions. A POJO returned by a Java callout is cached in the pipeline, and its key is returned wrapped in an XML message of the form `<java-content ref="cid:kkkkeeeeyyyy" />`, where `cid:kkkkeeeeyyyy` is a key automatically generated by the producing action and used to index the object in the pipeline's POJO repository. Any subsequent action then passes that XML unmodified as an argument.

The content of a POJO variable is not directly accessible by pipeline actions at configuration time. Rather, the content can be handled in the following ways:

- The content's metadata (that is, its key) can be handled as any other XML, for example in an XQuery such as `$pojo/java-content/@ref`. This may be useful for logging or debugging, but the content of the object cannot be directly accessed.
- The content can be assigned to a new variable that automatically becomes typed (in the pipeline) as a POJO. The object itself is not touched. The `<java-content . . . />` XML snippet is copied from the source variable to the target variable.
- The content can be passed to another appropriate action (like Java callout) as a variable (for example, `$pojo`). The object itself is not touched. The argument is automatically de-referenced to the actual object.

The Java object is removed from the pipeline's POJO repository when you delete all variables holding the object's key (in `<java-content />`) or when you delete all XPath expressions pointing to the `<java-content />` snippet.

Streaming Body Content

For processing message content, you can specify that the pipeline streams the content rather than loading it into memory. When you enable content streaming for a pipeline, you specify whether to buffer the streamed content to memory or a disk file as an intermediate step during the processing of the message. The creation of these temporary files might affect performance. For information about protecting temporary

files, see "Protection of Temporary Files With Streaming body Content" in *Administering Oracle Service Bus*.

When you enable the streaming option, content streaming applies *only* to the `$body` variable.

In general, use content streaming:

- When processing large content messages. See the guidelines in [Best Practices for Using Content Streaming](#).
- In use cases where Service Bus accesses the payload a small number of times.
- For content-based routing without transformations; content streaming results in better performance due to the benefits from partial parsing.

Best Practices for Using Content Streaming

Use the following guidelines and recommendations for enabling streaming content:

- When you enable streaming for large message processing, you cannot use the `insert`, `replace`, `rename`, `for each`, `validate`, and `delete` actions with respect to the `$body` message context variable, because these actions require the input variable to be fully materialized in memory and full materialization is incompatible with the content streaming option.
- You can use the results of an XQuery or XSL transformation from a very large `$body` with the following pipeline actions:
 - `Assign`, `insert`, and `replace` actions: Use the results to update the value of another context variable (not `$body`). However, you must ensure that the result of the expression is small enough to be fully materialized and stored in the message context.
 - `Java callouts`: Use the results to pass input arguments. All input to `Java callouts` is fully materialized, therefore, the results of expressions used as input must be small enough to be fully materialized.
 - `MFL transformations`: Use the results to transform very large payloads without first materializing the input as an XML Bean. When using a very large `$body` as an input to an MFL transformation, declare a messaging service, binary message type pipeline. If you declare a messaging service, text message type pipeline, `$body` will get fully materialized to obtain an input stream for the transformation.
 - `Alert`, `log`, and `report` actions: Use the results to report the result of an XQuery or XSL transformation on a very large `$body`.
 - `Service callouts`
- For XSL transformations, all input is fully materialized in order to perform the transformation, therefore, you must ensure that the input is small enough so that it can be fully materialized and processed by the XSLT processor.
- With very large MFL input, you should use an MFL service instead of an MFL stage action to perform a MFL-to-XML transformation.
- Do not use the Test Console to test pipelines with very large content messages because the content will be fully materialized, potentially causing an out-of-

memory exception, and displayed, causing a slowdown in the Test Console window.

- When writing XQueries, use proper indexing to achieve partial parsing. For example, instead of using `$body/*:DateTimeStruct`, which would consume the entire input stream, use one of the following:

```
($body/*:DateTimeStruct)[1]
```

or

```
$body[1]/*:DateTimeStruct[1]
```

By using indexing, only content up to and including the first `DateTimeStruct` element will be parsed.

- Because each variable that is accessed by two or more consumers (expressions) is materialized, when writing XQueries, avoid statements such as the following:

```
let $labdata1 := $body/*  
return <HEADER>{ $labdata1/HEADER/@*, $labdata1/HEADER/node() }</HEADER>
```

In this case, `$labdata1` is bound to the whole document without the root element so the XQuery engine runs out of memory when trying to materialize it.

One way of changing this query to avoid excessive materialization would be to move the `/HEADER` path expression inside the `let` clause.

```
let $labdata1 := $body/*/HEADER  
...
```

In this case, the XQuery engine will only materialize the `HEADER` element or elements.

- At runtime, processing large messages is subjected to the limitations and restrictions of the underlying transport; for example, the message size handling limitations of the transport. Be aware of the JVM and RMI settings that limit the capacity of the transport to accept large messages.

Streaming Attachments

For processing message attachments, you can specify that Service Bus page MIME attachments to disk and then stream the contents rather than buffering the attachments in memory and parsing them into XML. This is particularly advantageous when working with large attachments. Using this approach, Service Bus buffers only the headers and exposes the rest of the message payload as a stream from which smaller portions can be read at a time. Streamed transfers can improve the scalability of a service by eliminating the need for large memory buffers.

Note:

Service Bus does not support streaming attachments for email or WS transports.

When enabled for pipelines, the setting applies to the handling of inbound request messages. For HTTP business services, the setting applies in the handling of outbound response messages. The following actions allow outbound messages to be dispatched and support streaming attachments:

-
- Route, Dynamic Route, and Route Table
 - Publish, Dynamic Publish, and Publish Table

When streaming is enabled, all attachments, including binary, text, and XML, are processed as opaque data, which means you cannot run XQueries or XPath expressions based on the XML content of the attachment.

Inbound Message Handling

When Service Bus receives an inbound message and is configured to stream attachments, the contents of each MIME attachment is saved to a separate file on disk. The value of the `$attachments` variable is then set such that standard MIME headers for each attachment, including Content-ID, Content-Type, Content-Transfer-Encoding, Content-Description, Content-Location, and Content-Disposition, when present, are added under the attachment element.

The `body` child element then has a single `binary-content` child element that refers to the corresponding source in the source repository.

For example, an `$attachments` variable might appear as follows:

```
<con:attachments xmlns:con="http://www.oracle.com/wli/sb/context">
  <con:attachment>
    <con:Content-Type>image/jpeg</con:Content-Type>
    <con:Content-ID>
      &lt;l1.urn:uuid:BFB7D745CBAF21BA5B12023554608963@apache.org>
    </con:Content-ID>
    <con:Content-Transfer-Encoding>
      binary
    </con:Content-Transfer-Encoding>
    <con:body>
      <con:binary-content ref="cid:23976580:1183dd6aab9:-7fe0"/>
    </con:body>
  </con:attachment>
</con:attachments>
```

This is done regardless of the attachment content type. This means that `text/xml` attachments, for example, are treated identically to `image/jpeg` attachments, namely as opaque binary data with a binary content element.

This style of message handling differs from when attachments are not streamed, in which case certain types, such as `text/xml` or `text/plain` are recognized and used to initialize the `body` XML element (to contain the XML contents) or the text contents of the attachment respectively.

Outbound Response Message Handling

When Service Bus processes an outbound message and is configured to stream attachments, the contents of each MIME attachment is saved to a separate file on disk. Service Bus then initializes the `$attachments` message context variable in a manner similar to inbound requests (see [Inbound Message Handling](#)).

XOP/MTOM Support

Service Bus enables you to configure pipelines to decode and parse inbound messages in XOP/MTOM format and to send responses using the XOP/MTOM format, when appropriate. Oracle also enables you to configure business services to encode outbound messages in XOP/MTOM format.

Service Bus supports pipelines with the following binding types to accept and decode incoming XOP/MTOM payloads:

-
- Any XML
 - Messaging (XML)
 - Any SOAP
 - WSDL-based

Pipelines of any other service binding type that receive XOP/MTOM payloads treat them the same as any other MIME multipart request without MTOM-specific handling.

In addition, the following Service Bus transports support XOP/MTOM:

- HTTP/S
- Local
- SB (when applicable, such as with chained Service Bus domains)

Service Bus supports all existing actions that allow outbound messages to be dispatched, including the following:

- Route, Dynamic Route and Route Table
- Publish, Dynamic Publish and Publish Table
- Service Callout

Service Bus does not support combining MTOM and SOAP with Attachments (SwA).

XOP/MTOM in Pipelines

You can enable pipelines to decode and parse inbound messages in XOP/MTOM format and to send responses using the XOP/MTOM format, when appropriate. When XOP/MTOM support is enabled, you can further select how to handle binary data in the `$body` message context variables from among the following options:

- Include Binary Data by Reference: (Default) See [Binary by Reference Option](#).
- Include Binary Data by Value: See [Binary by Value Option](#).

Note that if XOP/MTOM Support is enabled for a pipeline, it is not required that every inbound message be in the MTOM format. Instead, this setting specifies that when an MTOM-formatted message arrives, the pipeline should handle it accordingly. Note also that when pipelines that are not enabled for XOP/MTOM support receive an MTOM-formatted message, the service rejects the message and issues a runtime error.

Binary by Reference Option

Use Include Binary Data by Reference when you need direct access to binary data, for example to pass data to a Java callout or Message Format Language (MFL) transformation.

When the Binary by Reference option is selected, Service Bus parses the root of the inbound message checking for the presence of `xop:Include` tags. These tags, when found, are converted to `ctx:binary-content` elements with a reference pointing to the corresponding source in binary repository. The resulting document is represented by the `$body` message context variable. The `$attachments` message context variable, in contrast, does not contain any information (is null).

This means that when pipeline actions access the contents of the `$body` message context variable, the actions do not encounter `xop:Include` elements, but instead work with `ctx:binary-content` elements.

When the pipeline needs to send a response back, the binding layer creates an XOP/MTOM package response by replacing `ctx:binary-content` references with `xop:Include` tags in the root of the message and adding a separate MIME part for each corresponding binary content reference.

Binary by Value Option

Use Include Binary Data by Value in the following cases:

- To bridge between MTOM and non-MTOM services. For example, consider an MTOM-enabled pipeline that receives a request that is then routed to a non-MTOM-enabled service. You could use this option to comply with existing standards for sending binary data in XML in Base64-encoded form.
- To validate the contents of the message against an XML schema that requires a `base64binary` element to be used in place of binary data.

When the Include Binary Data by Value option is selected, Service Bus parses the root of the inbound MIME message checking for the presence of `xop:Include` tags. When found, the body of the corresponding MIME parts (the binary data) are Base64 encoded and the resulting text replaces the `xop:Include` tags in the `$body` message context variable.

The `$attachments` message context variable, in contrast, does not contain any information (is null).

XOP/MTOM in Business Services

You can enable business services to encode outbound messages in XOP/MTOM format. When XOP/MTOM Support is enabled, you can further select how to handle binary data in the `$header` and `$body` message context variables from among the following options:

- **Include Binary Data by Reference:** (Default) In an outbound response message, replace `xop:Include` elements with `ctx:binary-content` elements when setting up the `$body` message context variable.
- **Include Binary Data by Value:** In an outbound response message, replace `xop:Include` elements with Base64-encoded text versions of corresponding binary data when setting up the `$body` message context variable.

Note that if XOP/MTOM support is enabled for a business service, it is not required that every outbound message be in the MTOM format. Instead, this setting specifies that the business service is capable of handling an MTOM payload.

XOP/MTOM in Outbound Messages

When XOP/MTOM support is enabled for a business service, Service Bus examines the contents of `$body` message context variable searching for `ctx:binary-content` elements. If any are present, Service Bus creates an XOP/MTOM MIME package replacing `ctx:binary-content` with `xop:Include` elements and with the corresponding MIME part in the payload.

Service Bus always uses XOP/MTOM when it is enabled and the body has binary content, regardless of the size of the content (for example, even when it is smaller than 1KB). Since Service Bus does not support the combination of MTOM and SwA, the

system throws a runtime exception when Service Bus needs to dispatch an outbound request to a business service and the following conditions are met:

- The business service is XOP/MTOM enabled
- The `$attachments` message context variable is not null

XOP/MTOM Attachments Streaming

With XOP/MTOM enabled on a service, Service Bus automatically modifies elements in the `$header` and `$body` of a message, as described in the previous sections. Because of the need to modify XOP/MTOM message content, Service Bus does not support streaming the message `$header` and `$body`.

However, there may be situations in which you need to stream the binary attachments of XOP/MTOM messages directly to disk rather than to the Service Bus heap (the default), such as when attachments are large and cause out-of-memory errors.

Note:

XOP/MTOM attachments are not sent in the `$attachments` variable of the message. They are included or referenced in the message `$body` as previously described.

To stream XOP/MTOM message attachments directly to disk, use the following settings on your service configuration:

- Use the XOP/MTOM Include Binary Data by Reference option. This option puts the binary data directly in the `$body`, which cannot be streamed for XOP/MTOM support.)
- With Include Binary Data by Reference selected, you can also select the Page Attachments to Disk option (HTTP transport only). With Page Attachments to Disk selected, Service Bus generates the XOP/MTOM message to reference the binary attachment on disk rather than in memory.

Note:

Referencing attachments in memory is recommended for optimum performance, so use the Page Attachments to Disk option only when you have a specific need for doing so, such as when sending large attachments whose size could cause out-of-memory errors.

Custom MIME Headers

Service Bus supports custom MIME headers for both inbound request/outbound response and outbound request/inbound response message patterns. When an inbound message includes a custom header and the message is read in the pipeline, the custom header information appears in the `user-headers` element in the XML, as shown in the example below.

```
<con:user-headers>
  <con:user-header name="MyCustomHeader" value="Custom Header Value" />
</con:user-headers>
```

For an outbound request with an inbound response, Service Bus converts any `user-header` elements to custom headers in the corresponding MIME part when Service Bus constructs the outbound request message. `user-header` elements can be the result of the (unpacked) inbound request or can be added manually in an explicit pipeline action, such as `insert`, `replace` or `assign` actions.

Inbound and Outbound Variables

The `$inbound` and `$outbound` context variables contain information about the inbound and outbound endpoints. The `$inbound` variable contains information about the proxy service that received the request message; the `$outbound` variable contains information about the target business service to which a message is sent.

The `$outbound` variable is set in the route action in route nodes and publish actions. You can modify `$outbound` by configuring request and response actions in route nodes and by configuring request actions in publish actions.

Caution:

Some modifications that you can make for the `$inbound` and `$outbound` context variables are not honored at runtime. That is, the values of certain headers and metadata can be overwritten or ignored by the Service Bus runtime. The same limitations are true when you set the transport headers and metadata using the transport headers and service callout actions, and when you use the Test Console to test your services.

For information about the headers and metadata for which there are limitations, see [How the Runtime Uses the Transport Settings in the Test Console](#). Note also that any modifications you make to `$outbound` in the message flow *outside* of the request or response actions in route nodes and publish actions are ignored. In other words, those modifications are overwritten when `$outbound` is initialized in the route nodes and publish actions.

You cannot modify the `$outbound` variable in service callout actions.

The `$inbound` and `$outbound` variables have the following characteristics:

- Have the same XML schema. The `$inbound` and `$outbound` context variables are instances of the `endpoint` element as described in [Message Context Schema](#).
- Contain a single `name` attribute that identifies the name of the endpoint as it is registered in the service directory. The `name` attribute should be considered read-only for both `$inbound` and `$outbound`.

Caution:

The read-only rule is not enforced. Changing read-only elements can result in unpredictable behavior.

- Contain the `service`, `transport` and `security` sub-elements described in [Sub-Elements of the Inbound and Outbound Variables](#).

Sub-Elements of the Inbound and Outbound Variables

This section describes the sub-elements of the `$inbound` and `$outbound` context variables, including information about whether a given sub-element is initialized at runtime. To learn about how context variables are initialized, see [Initializing Context Variables](#). The sub-elements include the following:

- [service](#)
- [transport](#)
- [security](#)

service

The `service` element is read-only for both `$inbound` and `$outbound`. Sub-elements include `providerName` and `operation`.

Note:

The Message Context Schema specifies the element types for the message context variables.

Table A-4 Sub-Elements of the service Element

Sub-Elements	Description...
<code>providerName</code>	The name of the service key provider. This is initialized based on the configuration of publish and routing actions.
<code>operation</code> (outbound only)	The name of the operation to be invoked on the target business service. This is initialized based on the <code>\$inbound</code> and <code>\$outbound</code> . Note: This element is used for the <code>\$outbound</code> variable only. In the case of inbound messages, the name of the operation to be invoked on the proxy service is specified by the <code>\$operation</code> variable.

transport

The `transport` element is read-only on inbound, except for the `response` element, which you can modify to set the response transport headers. The sub-elements of the `transport` element are described in [Table A-5](#).

Note:

The Message Context Schema specifies the element types for the message context variables.

Table A-5 Sub-Elements of the Transport Element

Sub-Elements	Description...
uri	<p>The URI of the endpoint:</p> <ul style="list-style-type: none">• When used in the <code>\$inbound</code> variable, this is the URI by which the message arrived.• When used in the <code>\$outbound</code> variable, this is the URI to use when sending the message. It overrides any URI value registered in the service directory. <p>Initialization</p> <p>The URI element is initialized as described below:</p> <ul style="list-style-type: none">• Always initialized on the <code>\$inbound</code> variable.• Never initialized on the <code>\$outbound</code> variable. You can set the URI on <code>\$outbound</code> when you want to override the set of URIs in the service configuration. URI failover is not supported if this element is set.

Table A-5 (Cont.) Sub-Elements of the Transport Element

Sub-Elements	Description...
request	<p data-bbox="735 323 1448 464">Transport-specific metadata about the request (including transport headers). The value for this element is defined by the transport protocol (specifically, the <code>RequestMetaData</code> XML defined by the transport SDK). Therefore, the structure of this element depends on the transport being used.</p> <p data-bbox="735 474 1448 585">This element is read-only in the <code>\$inbound</code> variable. You can modify it for the <code>\$outbound</code> variable. Note: The read-only rule is not enforced. Changing read-only elements can result in unpredictable behavior.</p> <p data-bbox="735 596 1448 680">To learn about the transport-specific types for this element, see the appropriate transport schema, which are available in the following directory in your Service Bus installation:</p> <p data-bbox="735 701 1122 726"><code>service_bus-home/config/plugins/</code></p> <p data-bbox="735 758 886 783">Initialization</p> <p data-bbox="735 793 1268 819">The URI element is initialized as described below:</p> <ul data-bbox="735 829 1448 1035" style="list-style-type: none"><li data-bbox="735 829 1448 888">• Initialized on the <code>\$inbound</code> variable using information from the request message received by Service Bus.<li data-bbox="735 898 1448 1035">• On the <code>\$outbound</code> variable, the <code>request</code> element is created with the proper typing. The typing is transport-dependent. The <code>request</code> element is typically initialized as an empty element, with the exception of certain important transport headers; for example, <code>content-type</code> and <code>SOAPAction</code>. <p data-bbox="735 1045 1448 1129">To set a filename for an outbound message using the File transport protocol, configure <code>\$outbound</code> in a route node request action, as described below:</p> <ul data-bbox="735 1140 1448 1423" style="list-style-type: none"><li data-bbox="735 1140 1448 1224">• If the <code>fileName</code> only is specified, a file of that name is stored at the location specified by the endpoint URI of the target business service.<li data-bbox="735 1234 1448 1423">• If <code>isFilePath</code> is set to <code>true</code>, the value of <code>fileName</code> is used as a relative path appended to the endpoint URI of the target business service. For example, if the endpoint URI is <code>file:///service/ob/data</code>, and the <code>fileName</code> header is set to <code>./schema/example.xml</code>, and <code>isFilePath</code> is set to <code>true</code>, the message will be stored at <code>/service/ob/data/schema/example.xml</code>. <p data-bbox="735 1434 1448 1518">If a file already exists with that name, a new name is generated, following the format <code>path/filename_random-number.xml</code>, where <code>random-number</code> is an integer in the range of 0 to 999999.</p>

Table A-5 (Cont.) Sub-Elements of the Transport Element

Sub-Elements	Description...
response	<p>Transport-specific metadata about the response (including transport headers). The value for this element is defined by the transport protocol (specifically, the <code>ResponseMetaData</code> XML defined by the transport SDK). Therefore, the structure of this element depends on the transport being used.</p> <p>This element is read-only in the <code>\$outbound</code> variable. You can modify it for the <code>\$inbound</code> variable.</p> <p>To learn about the transport-specific types for this element, see the appropriate transport schema, which are available in the following directory in your Service Bus installation:</p> <pre>service_bus-home/config/plugins/</pre> <p>Initialization</p> <p>The URI element is initialized as described below:</p> <ul style="list-style-type: none">• Initialized on the <code>\$outbound</code> variable using information from the response message received by Service Bus.• On the <code>\$inbound</code> variable, the <code>response</code> element is created with the proper typing. The typing is transport-dependent. The <code>response</code> element is typically initialized as an empty element, with the exception of certain important transport headers; for example, <code>content-type</code> and <code>SOAPAction</code>. <p>For a description of the standard HTTP headers, see http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html</p> <p>For a description of the standard JMS headers, see "Value-Added Public JMS API Extensions" in <i>Developing JMS Applications for Oracle WebLogic Server</i>.</p> <p>Note: The following MQ headers do not have equivalents in Oracle JMS: <code>ApplOriginData</code>, <code>ApplIdentityData</code>, <code>AccountingToken</code></p>
mode	<p>An indicator of whether the communication style is <code>request</code> (one-way) or <code>request-response</code> (two-way).</p> <p>Initialization</p> <p>Initialized on the <code>\$inbound</code> and <code>\$outbound</code> variables using information from the service and its operations (if applicable). For example, if a request-only operation is being invoked, the <code>mode</code> element is set to <code>request</code>, rather than to <code>request-response</code>.</p>

Table A-5 (Cont.) Sub-Elements of the Transport Element

Sub-Elements	Description...
<p>qualityOfService</p> <p>This element is read only for inbound.</p> <p>You can modify it for the outbound case— in the outbound request actions of a publish or routing action.</p>	<p>The quality of service expected when sending or receiving a message. Valid values include <code>best-effort</code> and <code>exactly-once</code>:</p> <ul style="list-style-type: none"> • best-effort means that each dispatch defines its own transactional context (if the transport is transactional). There is no reliable messaging and no elimination of duplicate messages; however, performance is optimized. <p>For the scenario in which a message is dispatched as a result of a publish action, any dispatch errors are suppressed. For the scenario in which a message is dispatched from a routing node, dispatch errors are not suppressed.</p> <ul style="list-style-type: none"> • exactly-once means that the dispatch is included as part of the inbound transactional context (if one exists and if the outbound transport is transactional) and errors cause processing to abort and trigger the relevant error handler (in the case of both the route and publish scenarios). Exactly once reliability means that messages are delivered from inbound to outbound exactly once, assuming a terminating error does not occur before the outbound message send is initiated. <p>Initialization</p> <p>The <code>qualityOfService</code> element is initialized on the <code>\$inbound</code> and <code>\$outbound</code> variables as described below:</p> <ul style="list-style-type: none"> • In the inbound case, the quality of service (QoS) is dictated by the transport. For example, for the JMS/XA transport, the QoS is exactly once; for the HTTP transport, the QoS is best effort. • In the outbound case, the QoS is set differently for publishing and for routings: <ul style="list-style-type: none"> Routing: When messages are routed to another service from a route node, the QoS is always initialized using the value from the <code>\$inbound</code> context variable. In other words, the outbound QoS is set to exactly once if (and only if) the inbound QoS is exactly once. Otherwise, the outbound QoS is set to best effort. Publishing: When a message is published to another service as the result of a publish action, the quality of service (QoS) is always initialized to best effort regardless of the inbound setting.
<p>retryCount</p> <p>(outbound only)</p>	<p>The number of retries to attempt when sending a message from Service Bus.</p> <p>If <code>retryCount</code> is set, the setting overrides any retry count value configured in the target service configuration.</p>

security

The sub elements of the `security` element are described in [Table A-6](#).

Note:

The Message Context Schema specifies the element types for the message context variables.

Table A-6 Sub-Elements of the Security Element

Sub-Elements	Description...
transportClient (inbound only)	<p>Authenticated transport-level user information. The user information includes a user name and any optional principals. The principals can themselves include zero or more groups, one for each group the subject belongs to.</p> <p>This variable is initialized by Service Bus. The inbound <code>transportClient</code> element is read-only. Note: The read-only rule is not enforced. Changing read-only elements can result in unpredictable behavior. If the subject is anonymous, the user name is anonymous and there are no groups.</p>
messageLevelClient (inbound only)	<p>Specifies authenticated message-level user information. The user information includes a user name and any optional principals. The principals can themselves include zero or more groups, one for each group the subject belongs to.</p> <p>This variable is initialized by Service Bus. The inbound <code>messageLevelClient</code> element is read-only. If the subject is anonymous, then the user name is anonymous and there are no groups.</p>
doOutboundWss (outbound only)	<p>Service Bus sets the value of this element during routing or publishing. Some infrequently used design patterns set the value to <code>false</code> to preempt a service from automatically generating the outbound WS-Security SOAP envelope.</p> <p>For more information, see under Disabling Outbound WS-Security.</p> <p>Note: When one proxy service invokes another proxy service (such as a local proxy) that contains Oracle Web Services Manager service policies, outbound WS-Security processing does not occur. cService Bus handles that behavior automatically and does not use the <code>doOutboundWss</code> property. For more information, see Using OWSM Security with Local Proxy Services..</p>

Related Topics

[Working with Pipeline Actions in Oracle Service Bus Console](#)

[How to Add Route Nodes to Pipelines in the Console](#)

For a description of the standard HTTP headers, see <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>

For a description of the standard JMS headers, see "Understanding WebLogic JMS" in *Developing JMS Applications for Oracle WebLogic Server*.

Operation Variable

The `$operation` variable is a read-only variable. It contains a string that identifies the operation to be invoked on a pipeline. If no operations are defined, the `$operation` variable is not set and returns the equivalent of null.

Service Bus provides the `$operation` variable as a stand-alone variable, rather than as a sub-element of the `$inbound` variable to optimize performance. The computation of the operation may be deferred until the `$operation` variable is explicitly accessed rather than anytime the `$inbound` variable is accessed.

Fault Variable

The `fault` variable holds information about any error that has occurred during message processing. When an error occurs, this variable is populated with information before the appropriate error handler is invoked. This variable is defined only in error handler pipelines and is not set in request and response pipelines or in route or branch nodes.

The `fault` variable includes the sub-elements described in [Table A-7](#).

Note:

The Message Context Schema specifies the element types for the message context variables.

Table A-7 Sub-Elements of the Fault Variable

Elements of the Fault Variables	Description
<code>errorCode</code>	The error code as a string value.
<code>reason</code>	A text description of the error.
<code>details</code>	User-defined XML content related to the error. For more information see Error Codes and Error Details .
<code>location</code>	Identifies the node, pipeline and stage in which the error occurred. Also identifies if the error occurred in an error handler. The sub-elements include the following: <ul style="list-style-type: none"><code>node</code>: The name of the pipeline pair, branch, or route node where an error occurred, in the form of a string.<code>pipeline</code>: The name of the pipeline where an error occurred (if applicable), in the form of a string.<code>stage</code>: The name of the stage where an error occurred (if applicable), in the form of a string.<code>error-handler</code>: A boolean indicator of whether an error occurred from inside an error handler.
<code>java-exception</code>	Information about the exception instance references. This can contain multiple <code>ref</code> elements, each of which define a single instance.
<code>stack-trace</code>	Any stack traces to add to the fault.

Error Codes

The contents of the `fault` variable are modeled after SOAP faults to facilitate fault generation when replying from a SOAP-based service. The values for error codes generated by Service Bus correspond to system error codes and are prefixed with an "OSB" string, unless configured otherwise. The error codes associated with the errors surface inside the element of the `fault` context variable. To access the value, use the following XQuery statement:

```
$fault/ctx:errorCode/text()
```

Service Bus defines generic error codes for the four classes of possible errors. The format of the generic codes is OSB-xxx000, where xxx represents a generic category as follows:

- 380 Transport and Proxy
- 382 Pipeline
- 386 Security
- 394 UDDI

This yields the generic codes as follows:

- OSB-380000—OSB-380999
Indicates a transport or proxy service error (for example, failure to dispatch a message).
- OSB-382000—OSB-382499
Indicates a pipeline runtime error (for example, a stage exception).
- OSB-382500—OSB-382999
Indicates an error in a pipeline action.
- OSB-386000—OSB-386999
Indicates a WS-Security error (for example, authorization failure).
- OSB-394500—OSB-394999
Indicates an error in the UDDI sub system.

Service Bus defines unique codes for specific errors. For example:

- OSB-382030: Indicates a message parsing error (for example, a SOAP service received a non-SOAP message).
- OSB-382500: Reserved for the case in which a service callout action receives a SOAP Fault response.

Error Details

The `details` element of the `fault` variable displays the type of fault, along with relevant information to the fault type. It can include any of the following elements indicating the error type:

- **ErrorResponseDetail:** Indicates that a service callout action received an error response from a transport provider.
- **InvalidEnvelope:** Indicates that a SOAP service received a well-formed XML document that was not an expected SOAP envelope.
- **PayloadDetail:** Indicates that an error occurred when parsing all or parts of the payload as XML.
- **ReceivedFaultDetail:** Indicates that a service callout action received a SOAP fault.
- **UnrecognizedResponseDetail:** Indicates that a service callout action received an unrecognized response from a transport provider.

- **ValidatonFailureDetail:** Indicates that an error occurred in a validation action.
- **WebServiceSecurityFault:** Indicates that a Web Services Security processing-related error occurred.
- **StackTrace:** The exception stack trace for cases not covered above.

You can view the errors schema file in [Errors Schema](#).

XML Parsing Errors (PayloadDetail)

When an error is an XML parsing error, the malformed XML text is included in the `PayloadDetail` element in the error schema, as described in [Error Details](#). Parsing errors are only detected when parsing inbound request messages to the pipeline and outbound response messages to the business services. Actions that explicitly change the payload, such as an assign action, that result in malformed XML are not included in this error detail.

Only the part of the payload that was read by the XML parser is included in the `fault` variable, so the `PayloadDetail` element might not include the entire XML text. By default, there is no limit to the number of characters that can be included in the error detail. To limit the size of the XML text included in the payload detail of the `fault` variable, set the system property `com.bea.wli.sb.FaultPayloadDetailMaxSize` to the size of the text you want to include (in kilobytes). Setting this property to 0 (zero) disables the feature, which means faults will not include the `PayloadDetail` element.

If the pipeline has content streaming enabled, the text placed in the `PayloadDetail` element is truncated to either 10KB or to the max size you set, if it is less than 10KB. If the payload contains binary characters, the payload is base64-encoded and placed in the `base64` child element of the `PayloadDetail` element (up to the maximum payload detail size).

messageID Variable

`messageID` is a transport provider-specific identifier associated with the message. It can be accessed via the `$messageID` system variable.

Initializing Context Variables

The message context and its variables are initialized in the binding layer when a message is received and before message processing begins. [Table A-8](#) summarizes how context variables are initialized.

Table A-8 Initializing Context Variables

Context Variable	How Initialized
<code>outbound</code>	Initialized to null because no routing or errors have yet occurred.
<code>fault</code>	The <code>\$outbound</code> variable is initialized in the route action in route nodes and publish actions. You can modify <code>\$outbound</code> through the request actions in routing nodes and publish actions (also in the response actions in routing nodes). For more information, see Inbound and Outbound Variables . For information about the initialization of sub-elements of <code>\$outbound</code> , see Sub-Elements of the Inbound and Outbound Variables .

Table A-8 (Cont.) Initializing Context Variables

Context Variable	How Initialized
inbound	Initialized with service, transport and security information that is obtained from Service Bus metadata about the registered proxy service and transport-level metadata (transport headers, authenticated user information, and so on) about the specific incoming request. For information about the initialization of sub-elements of <code>\$inbound</code> , see Sub-Elements of the Inbound and Outbound Variables .
header body attachments operation	Initialized using the content of the inbound message. How the initialization is performed depends on the type of service, as described in the subsequent topics in this section: <ul style="list-style-type: none">• Initializing the Attachments Context Variable• Initializing the Header and Body Context Variables The <code>\$header</code> , <code>\$body</code> , and <code>\$attachments</code> variables are re initialized after routing using the content of the response that is received. If no routing is performed or if the communication mode is request-only, then these variables are not re initialized. That is, they are not cleared of any content.

Initializing the Attachments Context Variable

The `$attachments` context variable is initialized with any MIME attachments that accompany the message, but does not include the part representing the main message (whether it is SOAP, XML, MFL, and so on). Each `<attachment>` element is initialized using the MIME headers that accompany each part in the MIME package.

The contents of the `<body>` element in the `<attachment>` can be one of the following depending on the attachment's `Content-Type`:

- XML
- text
- A snippet of reference XML that refers to the attachment content (see [Binary Content in the Body and Attachments Variables](#).)

Initializing the Header and Body Context Variables

This section describes how the initialization of `$header` and `$body` context variables is performed depending on the type of service: [SOAP Services](#), [XML Services \(Non SOAP\)](#), [Messaging Services](#).

SOAP Services

Messages to SOAP-based services are SOAP messages containing XML that is contained in a `<soap:Envelope>` element. In the case that messages include attachments, the content of the inbound message is a MIME package that includes the SOAP envelope as one of the parts, typically the first part or one identified by the top-level `Content-Type` header. The context variables are initialized as follows:

- `$header`: Initialized with the `<soap:Header>` element from the SOAP message
- `$body`: Initialized with the `<soap:Body>` element from the SOAP message

XML Services (Non SOAP)

The messages to XML-based services are XML, but can be of any type allowed by the service configuration. In the case that messages include attachments, the content of the inbound messages is a MIME package that includes the primary XML payload as one of the parts, typically the first part or one identified by the top-level `Content-Type` header.

The context variables are initialized as follows:

- `$header`: Initialized with an empty `<soap:Header/>` element.
- `$body`: Initialized with a `<soap:Body>` element that wraps the entire XML payload.

Messaging Services

Messaging services are those that can receive messages of one data type and respond with messages of a different data type. The supported data types include XML, MFL, text, untyped binary. The context variables are initialized as follows:

- `$header`: Initialized with an empty `<soap:Header/>` element.
- `$body`: Initialized with a `<soap:Body>` element that wraps the entire payload.
 - In the case of XML, MFL, and text content, it is placed directly within the `<soap:Body>` element.
 - In the case of binary content, a piece of reference XML is created and inserted inside the `<soap:Body>` element (see [Binary Content in the Body and Attachments Variables](#)). The binary content cannot be accessed or modified, but the reference XML can be examined, modified, and replaced with inline content.

Performing Operations on Context Variables

You interact with and manipulate the message context through actions in the pipeline pair, branch, or route nodes that define the pipeline. Most actions expose the XQuery language to do so. Each context variable is represented as an XQuery variable of the same name. For example, the `$header` variable is accessible in XQuery as `$header`, the `$body` variable is accessible as `$body`, and so on. The examples in this section show the use of XQuery to examine and manipulate context variables.

`$body`

The `$body` variable includes the `<soap-env:Body> . . . </soap-env:Body>` element. (If the service is SOAP 1.2, the `$body` variable contains a SOAP 1.2 Body element.) For example, if you assign data to the `$body` context variable using the Assign action, you must wrap it with the `<soap-env:Body>` element. In other words, you build the SOAP package by including the `<soap-env:Body>` element in the context variable.

There is an exception to this behavior for the case in which you build the Request Document Variable for the service callout action. Service callout actions work with the core payload (RPC parameters, documents, and so on) and Service Bus builds the SOAP package around the core payload. In other words, when you configure the Request Document Variable for a service callout action, you do not wrap the input document with `<soap-env:Body> . . . </soap-env:Body>`.

For information about configuring the service callout action, see [Adding Service Callout Actions in the Console](#).

\$header

The `$header` variable includes the `<soap-env:Header>...</soap-env:Header>` element. (If the proxy service is SOAP 1.2, the `$header` variable contains a SOAP 1.2 Header element.) For example, if you assign data to the `$header` context variable using the Assign action, you must wrap it with the `<soap-env:Header>` element. In other words, you build the SOAP package by including the `<soap-env:Header>` element in the context variable. This is true for all manipulations of `$header`, including the case in which you can set one or more SOAP headers for a service callout request. For information about configuring SOAP headers for a service callout action, see [Adding Service Callout Actions in the Console](#).

Extract the WS-Addressing Header—From: `$header/wsa:From`

Extract the Payload From a Non-SOAP Message: `$body/*`

Extract the user-header From an Outbound Response Message: `$outbound/ctx:transport/ctx:response/tp:user-header[@name='myheader']/@value`

When creating a `$body` input variable that is used for the request parameter in a service callout to a SOAP Service, you would define that variable's contents using `$body/*` (to remove the wrapper `soap-env:Body`), not `$body` (which results in keeping the `soap-env:Body` wrapper).

Assign Variable Contents for Request Parameter in a Service Callout: `$body/*`

Related Topics

For more information about handling context variables using the XQuery and XPath editors, see:

- [Working with Variable Structures](#)
- [Working With Expression Editors in Oracle Service Bus Console](#)

Constructing Messages to Dispatch

When Service Bus publishes or routes a message, the content of the message is constructed using the values of variables in the message context. For example, transport headers and other transport-specific metadata are taken from `$outbound/transport/request`. As is the case with initialization of the context, the message content for outbound messages is handled differently depending upon the type of the target service. How the outbound message content is created depends on the type of the target service, as described in the following topics:

- [SOAP Services](#)
- [XML Services \(Non SOAP\)](#)
- [Messaging Services](#)

SOAP Services

An outgoing SOAP message is constructed by wrapping the contents of the `$header` and `$body` variables inside a `<soap:Envelope>` element. If the invoked service is a SOAP 1.2 service, the envelope created is a SOAP 1.2 envelope. If the invoked service

is a SOAP 1.1 service, the envelope created is a SOAP 1.1 envelope. If the `$body` variable contains a piece of reference XML, it is sent as is; in other words, the referenced content is not substituted into the message.

If attachments are defined in the `$attachments` variable, a MIME package is created from the main message and the attachment data. The handling of the content for each attachment part is similar to how content is handled for messaging services.

XML Services (Non SOAP)

The messages to XML-based services from Service Bus are constructed from the contents of the `$body` variable:

- If the `$body` variable is empty, then a zero-size message is sent.
- If the `$body` variable contains multiple XML snippets, then only the first snippet is used in the outbound message. For example, if `<soap:Body>` contains `<abc /><xyz />`, only `<abc />` is sent.
- If the content of the `$body` variable is text and not XML, an error is thrown.
- If the `$body` variable contains a piece of reference XML, it is sent as is; in other words, the referenced content is not substituted into the message.
- If attachments are defined in the `$attachments` variable, a MIME package is created from the XML message and the attachment data. In the case of a null XML message, the corresponding MIME body part is empty. The handling of the content for each attachment part is similar to how content is handled for messaging services.

Regardless of any data it contains, the `$header` variable does not contribute any content to the outbound message. For examples of how messages are constructed for service callout actions, see [Working with Pipeline Actions in Oracle Service Bus Console](#).

Messaging Services

The messages to messaging services from Service Bus are constructed from the contents of the `$body` variable.

- If the `$body` variable is empty, then a zero-size message is sent, regardless of the outgoing message type.
- If the outgoing message type is XML, then the message is constructed in the same way as it is for [XML Services \(Non SOAP\)](#).
- If the outgoing message type is MFL, then the behavior is similar to that for XML message types except that the extracted XML is converted to MFL. An error occurs if the XML > MFL conversion cannot be performed.
- If the target service requires text messages, the contents of the `$body` variable are interpreted as text and sent. In this way, it is possible for Service Bus to handle incoming XML messages that must be delivered to a target service as text. In other words, you do not need to configure the message flow to handle such messages.
- For target services that expect binary messages, the `$body` variable must contain a piece of reference XML—the reference URI references the binary data stored in the Service Bus in-memory hash table. The referenced content is sent to the target service.

For cases in which a client, a transport, or the designer of a service specifies the reference URI, the referenced data is not stored in Service Bus and thus cannot be de referenced to populate the outbound message. Consequently, the reference XML is sent in the message.

If the `$body` variable contains a piece of reference XML, and the target service requires a message type other than binary, the reference XML inside the `$body` variable is treated as content. In other words, it is sent as XML, converted to text, or converted to MFL. This is true regardless of the URI in the reference XML.

Regardless of any data it contains, the `$header` variable does not contribute any content to the outbound message.

For examples of how messages are constructed for service callout actions, see [Working with Pipeline Actions in Oracle Service Bus Console](#).

About Sending Binary Content in Email Messages

For binary messages, Service Bus does not insert the message content into the `$body` variable. Instead, a `<binary-content />` reference element is created and inserted into the `<SOAP:Body>` element (see [Message-Related Variables](#)). However, the email standard does not support sending binary content type as the main part of a message. If you want to send binary messages by email to a messaging service that accepts text or XML documents and optional attachments, you can do so as follows:

1. Transfer the `binary-content` reference XML from `$body` to `$attachments`.
2. Replace the content of `$body` with text or XML wrapped in a `<SOAP:Body>` element.

For the case in which the outgoing message type is MFL, the contents of `$body` is converted from XML to text or binary based on the MFL transformation:

- If the target service expects to receive text message, you can set the `content-type` (the default is binary for MFL message type) as `text/plain` in `$outbound`
- If the target service expects to receive binary messages, it is not possible to send MFL content using the email transport.

To learn more about how binary content is handled, see [Binary Content in the Body and Attachments Variables](#).

Related Topics

[Message Context Schema](#)

[Adding Service Callout Actions in the Console](#)

[Adding Transport Header Actions in the Console](#)

[How to Add Route Nodes to Pipelines in the Console](#)

Message Context Schema

The following example shows the message context schema (`MessageContext.xsd`), which specifies the types for the message context variables.

When working with the message context variables, you need to reference `MessageContext.xsd` which is available in a JAR file, `OSB_ORACLE_HOME/lib/servicebus-schemas.jar`, and the transport-specific schemas, which are available at `OSB_ORACLE_HOME/config/plugins/`.

Example - MessageContext.xsd

```
<schema targetNamespace="http://www.bea.com/wli/sb/context"
  xmlns:mc="http://www.bea.com/wli/sb/context"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <!-- =====
-->

  <!-- The context variable 'fault' is an instance of this element -->
  <element name="fault" type="mc:FaultType"/>

  <!-- The context variables 'inbound' and 'outbound' are instances of this
  element -->
  <element name="endpoint" type="mc:EndpointType"/>

  <!-- The three sub-elements within the 'inbound' and 'outbound' variables -->
  <element name="service" type="mc:ServiceType"/>
  <element name="transport" type="mc:TransportType"/>
  <element name="security" type="mc:SecurityType"/>

  <!-- The context variable 'attachments' is an instance of this element -->
  <element name="attachments" type="mc:AttachmentsType"/>

  <!-- Each attachment in the 'attachments' variable is represented by an
  instance of this element -->
  <element name="attachment" type="mc:AttachmentType"/>

  <!-- Used to represent binary payloads and pass-by reference content -->
  <element name="binary-content" type="mc:BinaryContentType"/>

  <!-- Element used to represent POJOs stored in object repository -->
  <element name="java-content" type="mc:JavaContentType"/>

  <!-- =====
-->

  <!-- The schema type for -->
  <complexType name="AttachmentsType">
    <sequence>
      <!-- the 'attachments' variable is just a series of attachment
      elements -->
      <element ref="mc:attachment" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>

  <complexType name="AttachmentType">
    <all>
      <!-- Set of MIME headers associated with attachment -->
      <element name="Content-ID" type="string" minOccurs="0"/>
      <element name="Content-Type" type="string" minOccurs="0"/>
      <element name="Content-Transfer-Encoding" type="string"
        minOccurs="0"/>
      <element name="Content-Description" type="string" minOccurs="0"/>
      <element name="Content-Location" type="string" minOccurs="0"/>
      <element name="Content-Disposition" type="string" minOccurs="0"/>

      <!-- any (custom) headers not specified above will be here -->
```

```

        <element name="user-headers" minOccurs="0">
            <complexType>
                <sequence>
                    <element name="user-header" maxOccurs="unbounded">
                        <complexType>
                            <attribute name="name" type="string"/>
                            <attribute name="value" type="string"/>
                        </complexType>
                    </element>
                </sequence>
            </complexType>
        </element>

        <!-- Contains the attachment content itself, either in-lined or as
<binary-content/> -->
        <element name="body" type="anyType"/>
    </all>
</complexType>

<complexType name="BinaryContentType">
    <!-- URI reference to the binary or pass-by-reference payload -->
    <attribute name="ref" type="anyURI" use="required"/>
</complexType>

<complexType name="JavaContentType">
    <!-- URI reference to POJOs in object repository -->
    <attribute name="ref" type="anyURI" use="required"/>
</complexType>

<!-- =====
-->

<complexType name="EndpointType">
    <all>
        <!-- Sub-elements holding service, transport and security details for
the endpoint -->
        <element ref="mc:service" minOccurs="0" />
        <element ref="mc:transport" minOccurs="0" />
        <element ref="mc:security" minOccurs="0" />
    </all>

    <!-- Fully-qualified name of the service represented by this endpoint -->
    <attribute name="name" type="string" use="required"/>
</complexType>

<!-- =====
-->

<complexType name="ServiceType">
    <all >
        <!-- name of service provider -->
        <element name="providerName" type="string" minOccurs="0"/>

        <!-- the service operation being invoked -->
        <element name="operation" type="string" minOccurs="0"/>
    </all>
</complexType>

<!-- =====
-->

```



```

<complexType name="TransportType">
  <all>
    <!-- URI of endpoint -->
    <element name="uri" type="anyURI" minOccurs="0" />

    <!-- Transport-specific metadata for request and response (includes
    transport headers) -->
    <element name="request" type="anyType" minOccurs="0"/>
    <element name="response" type="anyType" minOccurs="0" />

    <!-- Indicates one-way (request only) or bi-directional
    (request/response) communication -->
    <element name="mode" type="mc:ModeType" minOccurs="0" />

    <!-- Specifies the quality of service -->
    <element name="qualityOfService" type="mc:QoSType" minOccurs="0" />

    <!-- Retry values (outbound only) -->
    <element name="retryInterval" type="integer" minOccurs="0" />
    <element name="retryCount" type="integer" minOccurs="0" />

    <!-- Throttling priority (outbound only) -->
    <element name="priority" type="positiveInteger" minOccurs="0" />
  </all>
</complexType>

<simpleType name="ModeType">
  <restriction base="string">
    <enumeration value="request"/>
    <enumeration value="request-response"/>
  </restriction>
</simpleType>

<simpleType name="QoSType">
  <restriction base="string">
    <enumeration value="best-effort"/>
    <enumeration value="exactly-once"/>
  </restriction>
</simpleType>

<!-- =====
-->

<complexType name="SecurityType">
  <all>
    <!-- Transport-level client information (inbound only) -->
    <element name="transportClient" type="mc:SubjectType" minOccurs="0"/>

    <!-- Message-level client information (inbound only) -->
    <element name="messageLevelClient" type="mc:SubjectType"
      minOccurs="0"/>

    <!-- Boolean flag used to disable outbound WSS processing (outbound
    only) -->
    <element name="doOutboundWss" type="boolean" minOccurs="0"/>
  </all>
</complexType>

<complexType name="SubjectType">

```

```

<sequence>
  <!-- User name associated with this tranport- or message-level subject --
>
  <element name="username" type="string"/>
  <element name="principals" minOccurs="0">
    <complexType>
      <sequence>
        <!-- There is an element for each group this subject
              belongs to, as determined by the authentication
              providers -->
        <element name="group" type="string" minOccurs="0"
              maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
  <element name="subject-properties" minOccurs="0" maxOccurs="1">
    <complexType>
      <sequence>
        <element name="property" minOccurs="0"
              maxOccurs="unbounded"
              type="mc:SubjectPropertyType"/>
      </sequence>
    </complexType>
  </element>
</sequence>
</complexType>

<complexType name="SubjectPropertyType" mixed="true">
  <sequence>
    <any minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
  </sequence>
  <attribute name="name" type="string" use="required"/>
  <anyAttribute processContents="lax"/>
</complexType>

<!-- =====
-->

<complexType name="FaultType">
  <all>
    <!-- A short string identifying the error (e.g. BEA38229) -->
    <element name="errorCode" type="string"/>

    <!-- Descriptive text explaining the reason for the error -->
    <element name="reason" type="string" minOccurs="0" />

    <!-- Any additional details about the error -->
    <element name="details" type="anyType" minOccurs="0" />

    <!-- Information about where the error occurred in the proxy -->
    <element name="location" type="mc:LocationType" minOccurs="0" />

    <!-- Information about the exception instance reference stored in the
          object repository -->
    <element name="java-exception" minOccurs="0">
      <complexType>
        <sequence>
          <element ref="mc:java-content" />
        </sequence>
      </complexType>
    </element>
  </all>

```

```

    </all>
  </complexType>

  <simpleType name="PipelinePathType">
    <restriction base="string">
      <enumeration value="request-pipeline"/>
      <enumeration value="response-pipeline"/>
    </restriction>
  </simpleType>

  <complexType name="LocationType">
    <all>
      <!-- Name of the Pipeline/Branch/Route node where error occurred -->
      <element name="node" type="string" minOccurs="0" />

      <!-- Name of the Pipeline where error occurred (if applicable) -->
      <element name="pipeline" type="string" minOccurs="0" />

      <!-- Name of the Stage where error occurred (if applicable) -->
      <element name="stage" type="string" minOccurs="0" />

      <!-- Indicates if error occurred from inside an error handler -->
      <element name="error-handler" type="boolean" minOccurs="0" />

      <!-- Indicates whether or not error occurred in request or
      response path -->
      <element name="path" type="mc:PipelinePathType" minOccurs="0" />
    </all>
  </complexType>

  <!-- Encapsulates any stack-traces that may be added to a fault <details> -->
  <element name="stack-trace" type="string"/>

</schema>

```

Errors Schema

The following example shows the error schema (`Errors.xsd`), which specifies the structure for errors. This file is available in a JAR file, `OSB_ORACLE_HOME/lib/servicebus-schemas.jar`.

Example - Errors.xsd

```

<schema targetNamespace="http://www.bea.com/wli/sb/errors"
  xmlns:err="http://www.bea.com/wli/sb/errors"
  xmlns:tc="http://www.bea.com/wli/sb/transports"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <import namespace="http://www.bea.com/wli/sb/transports"
    schemaLocation="TransportCommon.xsd" />

  <element name="InvalidEnvelope">
    <complexType>
      <sequence>
        <element name="localpart" type="NCName"/>
        <element name="namespace" type="anyURI" minOccurs="0"/>
      </sequence>
    </complexType>
  </element>

```

```

<element name="WebServiceSecurityFault">
  <complexType>
    <sequence>
      <element name="faultcode" type="QName"/>
      <element name="faultstring" type="string"/>
      <element name="detail" minOccurs="0">
        <complexType mixed="true">
          <sequence>
            <any namespace="##any" minOccurs="0"
maxOccurs="unbounded" processContents="lax"/>
          </sequence>
          <anyAttribute namespace="##any" processContents="lax"/>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>

<!-- Publish error details -->
<element name="ErrorResponseDetail" type="err:ErrorResponseDetail"/>
<complexType name="ErrorResponseDetail">
  <sequence>
    <!-- Response metadata -->
    <element name="response-metadata" type="tc:ResponseMetaDataXML"
minOccurs="0" />
  </sequence>
</complexType>

<!-- payload information in $fault details when parsing message into XML raises
an error -->
<element name="PayloadDetail" type="err:PayloadDetail"/>
<complexType name="PayloadDetail">
  <choice>
    <element name="text" minOccurs="0" type="string" />
    <element name="base64" minOccurs="0" type="base64Binary" />
  </choice>
</complexType>

<!-- REST default (system) faults -->
<element name="RestError" type="err:RestErrorType"/>
<complexType name="RestErrorType">
  <sequence>
    <element name="errorMessage" minOccurs="0" type="string" />
    <element name="errorCode" minOccurs="0" type="string" />
  </sequence>
</complexType>

</schema>

```

XPath Extension Functions

This appendix describes the XPath extension functions for working with cross references and domain value maps in Service Bus. It also describes how to create user-defined XPath extension functions. Oracle provides XPath functions that use the capabilities built into Service Bus and XPath standards for adding new functions.

This appendix includes the following sections:

- [Cross-Reference Functions](#)
- [Domain Value Map Functions](#)
- [Creating Custom XPath Functions](#)

Cross-Reference Functions

Using cross references, you can dynamically map values for an entity in one application to the equivalent values in other applications. This is useful for when you update information for an object in one application, and that information needs to be propagated to the same object in other applications. Each application may have its own way of identifying that object.

Cross references are stored in a lookup table, and the cross reference XPath functions let you monitor and manage the data in the table. Use these functions to add, update, and lookup data in the cross reference mapping tables. A global transaction should be available during these function calls. If the transaction is not available, a new transaction is initiated.

Note:

Most of the cross reference functions take the location and name of the Service Bus cross reference resource as an argument. Service Bus does not support a direct reference to resources in the Oracle Metadata Services (MDS) Repository.

For complete information about cross references, see "Working with Cross References" in *Developing SOA Applications with Oracle SOA Suite*. For information about cross references in Service Bus, see [Mapping Data with Cross-References](#).

lookupPopulatedColumns

This function looks up all the populated columns for a given cross reference table, cross reference column, and value. It returns a node-set, with each node containing a column name and the corresponding value. For a more detailed explanation of this function, see "About the xref:lookupPopulatedColumns Function" in *Developing SOA Applications with Oracle SOA Suite*.

Signature

`xref:lookupPopulatedColumns(xref-location, column, value, need-exception)`

Arguments

Unless otherwise noted, all argument are String values.

- `xref-location`: The full path and name of the cross-reference resource.
- `column`: The name of the reference column.
- `value`: The value corresponding to the reference column.
- `need-exception`: A boolean value indicating whether to throw an exception if the value is not found. When this is set to `true`, the function throws an exception if the value is not found. Otherwise, it returns an empty value.

At runtime, an exception can occur for 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.

Property IDs

- `namespace-uri`: `http://www.oracle.com/osb/xpath-functions/xref`
- `namespace-prefix`: `xref`

Example

```
xref:lookupPopulatedColumns('/BookSellers/XRefPublishers', 'HARPER', 'H_1500',  
true())
```

lookupXRef

This function looks up a cross reference column for a value that corresponds to a specific value in a reference column. For a more detailed explanation of this function, see "About the `xref:lookupXRef` Function" in *Developing SOA Applications with Oracle SOA Suite*.

Signature

`xref:lookupXRef(xref-location, ref-column, ref-value, column-name, need-exception)`

Arguments

Unless otherwise noted, all argument are String values.

- `xref-location`: The full path and name of the cross-reference resource.
- `ref-column`: The name of the reference column.
- `ref-value`: The value corresponding to the reference column.
- `column-name`: The name of the column in which to look up the value.
- `need-exception`: A boolean value indicating whether to throw an exception if the value is not found. When this is set to `true`, the function throws an exception if the value is not found. Otherwise, it returns an empty value.

At runtime, an exception can occur for 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.

Property IDs

- `namespace-uri`: `http://www.oracle.com/osb/xpath-functions/xref`
- `namespace-prefix`: `xref`

Example

```
xref:lookupXRef('/BookSellers/XRefPublishers', 'HARPER', 'H_1500', 'PENGUIN',  
false())
```

lookupXRef1M

This function looks up a cross reference column for multiple values that correspond to a specific value in a reference column. It returns a node-set containing multiple nodes, and each node contains a value. For a more detailed explanation of this function, see "About the `xref:lookupXRef1M` Function" in *Developing SOA Applications with Oracle SOA Suite*.

Signature

```
xref:lookupXRef1M(xref-location, ref-column, ref-value, column-name, need-exception)
```

Arguments

Unless otherwise noted, all argument are String values.

- `xref-location`: The full path and name of the cross-reference resource.
- `ref-column`: The name of the reference column.
- `ref-value`: The value corresponding to the reference column.
- `column-name`: The name of the column in which to look up the values.
- `need-exception`: A boolean value indicating whether to throw an exception if the values are not found. When this is set to `true`, the function throws an exception if the values are not found. Otherwise, it returns an empty value.

At runtime, an exception can occur for 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.

Property IDs

- `namespace-uri`: `http://www.oracle.com/osb/xpath-functions/xref`
- `namespace-prefix`: `xref`

Example

```
xref:lookupXRef1M('/BookSellers/XRefPublishers', 'HARPER', 'H_1500', 'PENGUIN',
true())
```

markForDelete

This function deletes a value in a cross reference table. The column value passed to the function is deleted from the `XREF_DATA` table and moved to the `XREF_DELETED_DATA` table. This function returns `true` if the deletion is successful. Otherwise, it returns `false`. If there is only one value left in the row after the deletion, that value is also deleted since there are no cross references remaining.

For a more detailed explanation of this function, see "Deleting a Cross Reference Table Value" in *Developing SOA Applications with Oracle SOA Suite*.

Signature

```
xref:markForDelete(xref-location, column, value)
```

Arguments

Unless otherwise noted, all argument are String values.

- `xref-location`: The full path and name of the cross-reference resource.
- `column`: The name of the column that contains the value to be deleted.
- `value`: The value to be deleted.

Property IDs

- `namespace-uri`: `http://www.oracle.com/osb/xpath-functions/xref`
- `namespace-prefix`: `xref`

Example

```
xref:markForDelete('/BookSellers/XRefPublishers', 'HARPER', 'H_1500')
```

populateLookupXRefRow

This function populates a column or columns in the cross-reference table with a single value, depending on the mode in which it is run. Use this function to add a new row or to add a value to a column in an existing row. This function returns a string value, which is the value being populated. Unlike the [populateXRefRow](#) function, the `populateLookupXRefRow` function does not throw a unique constraint violation error when records with the same ID are added simultaneously. Instead, it behaves as a lookup and returns the existing source value that caused the error and does not stop the processing flow. Use this function to resolve any concurrency issues that could arise when using the `populateXRefRow` function.

For a more detailed explanation of this function, see "About the `xref:populateLookupXRefRow` Function" in *Developing SOA Applications with Oracle SOA Suite*.

Signature

```
xref:populateLookupXRefRow(xref-location, ref-column, ref-value, column, value,
mode)
```

Arguments

All argument are String values.

-
- `xref-location`: The full path and name of the cross-reference resource.
 - `ref-column`: The name of the reference column.
 - `ref-value`: The value that corresponds to the reference column.
 - `column`: The name of the column to be populated.
 - `value`: The value to be populated in the above column.
 - `mode`: The XREF population mode. This can be `ADD` or `LINK`, and must be entered in all uppercase letters. For more information about these modes, see "xref:populateLookupXRefRow Function Results with Different Modes" in *Developing SOA Applications with Oracle SOA Suite*.

Property IDs

- `namespace-uri`: `http://www.oracle.com/osb/xpath-functions/xref`
- `namespace-prefix`: `xref`

Example

```
xref:populateLookupXRefRow('/BookSellers/XRefPublishers', 'HARPER', 'H_1500',  
'PENGUIN', 'PEN_2001', 'ADD')
```

populateXRefRow

This function populates one or two columns in the cross-reference table with a single value, depending on the mode in which it is run. Use this function to add a new row, or to update or add a value in a column in an existing row. This function returns a string value, which is the value being populated. For a more detailed explanation of this function, see "About the xref:populateXRefRow Function" in *Developing SOA Applications with Oracle SOA Suite*.

Note:

If you find you have concurrency issues when using this function, you can also use the [populateLookupXRefRow](#) function, which should only be used in cases where simultaneous updates are being made that result in unique constraint violations.

Signature

```
xref:populateXRefRow(xref-location, ref-column, ref-value, column, value, mode)
```

Arguments

All argument are String values.

- `xref-location`: The full path and name of the cross-reference resource.
- `ref-column`: The name of the reference column.
- `ref-value`: The value that corresponds to the reference column.
- `column`: The name of the column to be populated.
- `value`: The value to be populated in the above column.

-
- `mode`: The XREF population mode. This can be `ADD`, `LINK`, or `UPDATE`, and must be entered in all uppercase letters. For more information about these modes, see "xref:populateXRefRow Function Modes" in *Developing SOA Applications with Oracle SOA Suite*.

Property IDs

- `namespace-uri`: `http://www.oracle.com/osb/xpath-functions/xref`
- `namespace-prefix`: `xref`

Example

```
xref:populateXRefRow('/BookSellers/XRefPublishers', 'HARPER', 'H_1500', 'PENGUIN', 'PEN_2001', 'ADD')
```

populateXRefRow1M

This function populates a column or columns in the cross-reference table with multiple values, depending on the mode in which it is run. Use this function to add a new row or to add values to a column in an existing row. Two values in one external system can correspond to a single value in another system. In such a scenario, use this function to populate a cross reference column with a value. This function returns a string value, which is the cross reference value being populated.

For a more detailed explanation of this function, see "About the xref:populateXRefRow1M Function" in *Developing SOA Applications with Oracle SOA Suite*.

Signature

```
xref:populateXRefRow1M(ref-location, ref-column, ref-value, column, value, mode)
```

Arguments

All argument are String values.

- `xref-location`: The full path and name of the cross-reference resource.
- `ref-column`: The name of the reference column.
- `ref-value`: The value that corresponds to the reference column.
- `column`: The name of the column to be populated.
- `value`: The value to be populated in the above column.
- `mode`: The XREF population mode. This can be `ADD` or `LINK`, and must be entered in all uppercase letters. For more information about these modes, see "xref:populateXRefRow1M Function Results with Different Modes" in *Developing SOA Applications with Oracle SOA Suite*.

Property IDs

- `namespace-uri`: `http://www.oracle.com/osb/xpath-functions/xref`
- `namespace-prefix`: `xref`

Example

```
xref:populateXRefRow1M('/BookSellers/XRefPublishers', 'HARPER', 'H_1500', 'PENGUIN', 'PEN_2001', 'LINK')
```

Domain Value Map Functions

Using domain value maps, you can map the terms used by different domains to describe the same entity, so values used by one domain for specific fields map to the values used by other domains for the same fields. For example, you can map country or state codes between applications. Domain value maps are stored in a lookup table, and the domain value map XPath functions let you look up the data in the table.

For complete information about domain value maps, see "Using Domain Value Map Functions" in *Developing SOA Applications with Oracle SOA Suite*. For information about cross references in Service Bus, see [Mapping Data with Domain Value Maps](#)..

lookup

This function returns a string by looking up the value for the target column in a domain value map, where the source column contains the given source value.

Signature

```
dvm:lookup(dvm-location, src-column, src-value, target-column, default-value)
```

Arguments

All arguments are String values.

- `dvm-location`: The domain value map URI.
- `src-column`: The source column name.
- `src-value`: The source value (an XPath expression bound to the source document of the XSLT transformation).
- `target-column`: The target column name.
- `default-value`: If the value is not found, this default value is returned.

Property IDs

- `namespace-uri`: `http://www.oracle.com/osb/xpath-functions/dvm`
- `namespace-prefix`: `dvm`

Example

The following example looks for the value in the CityNames table that corresponds to the value "BO" in the CityCodes table. If no matching value is found, the function returns "CouldNotBeFound" instead.

```
dvm:lookup ('/CityMapsDVM/cityMap', 'CityCodes', 'BO', 'CityNames', 'CouldNotBeFound')
```

lookupValue

This function returns a string by looking up the value for the target column in a domain value map, where the source column contains the given source value. You can specify a qualifying column and value to help narrow down this lookup. For a more detailed explanation of this function, see "Using Domain Value Map Functions" in *Developing SOA Applications with Oracle SOA Suite*.

Signature

```
dvm:lookupValue(dvm-location, src-column, src-value, target-column, default-value, qualifiers[])
```

Arguments

All arguments are String values.

- `dvm-location`: The full path and name of the DVM resource.
- `src-column`: The source column name.
- `src-value`: The source value (an XPath expression bound to the source document of the XSLT transformation).
- `target-column`: The target column name.
- `default-value`: If the value is not found, this default value is returned.
- `qualifier-column`: A column that, along with the qualifier value, helps to narrow down the lookup. For more information about qualifiers, see "Qualifier Domains" in *Developing SOA Applications with Oracle SOA Suite*.
- `qualifier-value`: A value in the qualifier column.

Property IDs

- `namespace-uri`: `http://www.oracle.com/osb/xpath-functions/dvm`
- `namespace-prefix`: `dvm`

Example

The following example looks for the value in the CityNames table that corresponds to the value "BO" in the CityCodes table. The qualifier "Massachusetts" in the State table helps to narrow down the city name to just that state. If no matching value is found, the function returns "CouldNotBeFound" instead.

```
dvm:lookupValue ('/CityMapsDVM/cityMap', 'CityCodes', 'BO', 'CityNames',  
'CouldNotBeFound', 'State', 'Massachusetts')
```

lookupValue1M

This function returns an XML document fragment containing values for multiple target columns of a domain value map, where the value for the source column is equal to the source value. When using this function in an expression, if the returned value is directly assigned to a variable, the first occurrence is assigned to the variable. The returned value should be iterated to make the proper assignment.

For a more detailed explanation of this function, see "Using Domain Value Map Functions" in *Developing SOA Applications with Oracle SOA Suite*.

Signature

```
dvm:lookupValue1M(dvm-location, src-column, src-value, target-columns[])
```

Arguments

All arguments are String values.

- `dvm-location`: The full path and name of the DVM resource.
- `src-column`: The source column name.
- `src-value`: The source value (an XPath expression bound to the source document of the XSLT transformation).

-
- `target-columns`: The names of the target columns. At least one column name should be specified. You can specify multiple target column names, each as a separate argument.

Property IDs

- `namespace-uri`: `http://www.oracle.com/osb/xpath-functions/dvm`
- `namespace-prefix`: `dvm`

Example

The following example looks for the values in the `CityNames` and `CityNickNames` tables that correspond to the value "Garden City" in the `CityCodes` table.

```
dvm:lookupValue1M ('/CityMapsDVM/cityMap', 'CityCodes', 'Garden City', 'CityNames', 'CityNickName')
```

Creating Custom XPath Functions

In addition to the standard XPath functions provided with Service Bus, you can create and register custom XPath functions to use in your expressions. Service Bus provides an extensible framework for creating custom XPath functions you can use in the XQuery expression editors in the development or runtime tooling, such as in pipelines, split-joins, and XQuery Mapper transformations.

This section includes the following topics:

- [Registering Custom Functions with Service Bus](#)
- [Creating and Packaging the Custom Function Java Classes](#)
- [Using Custom Functions](#)
- [Deploying Custom Functions in a Cluster](#)

Note:

Service Bus does not support custom functions that have side effects; for example, updating a database table or participating in a global transaction. Create custom functions that contribute only to an XQuery result, and perform side-effect behavior with other features such as Java Callouts.

Registering Custom Functions with Service Bus

Custom functions are available to all Service Bus projects and services within a WebLogic Server domain. To register a custom function, you create an XML file with an optional properties file for localization. The built-in functions that Service Bus provides use this function framework, so you can use those existing registration resources as a guide. Those files are located in the following directory:

```
/service_bus_home/config/xpath-functions
```

The Service Bus functions file is called `osb-built-in.xml`. In that file, keys wrapped in % symbols, such as `%OSB_FUNCTIONS%`, get their value from the corresponding `.properties` file. To extend custom functions to a new domain, you need to copy the registration files (XML and properties files) to the new domain in the following location:

`/domain_home/config/osb/xpath-functions`

You might need to create the `xpath-functions` subdirectory.

Below is an outline of the basic structure of a custom function registration file, followed by descriptions of the elements.

```
category id
  group id
    function
      name
      comment
      namespaceURI
      className
      method
      isDeterministic
      scope
```

Elements have an `xpf: prefix`.

The following table lists and describes each element in the custom registration file.

Table B-1 Custom Function Registration File Properties

Element	Description
category	The group that physically categorizes your group of functions in the expression editors, such as "Service Bus Functions." Use the <code>id</code> attribute to provide the name. If you are using a corresponding <code>.properties</code> file for localization, enter the key that contains the text value in the properties file; for example, <code>%MY_FUNCTIONS%</code> . Category IDs, which include the properties file key name and the actual name value, must be unique.
group	A subcategory for grouping functions in the user interface, such as "General" or "Accessors." Use the <code>id</code> attribute to provide the name. The naming guidelines for category ID apply to group ID. The group element is optional.
name	The name of the function as it appears in XQuery expressions. Function names, which include the namespaceURI and prefix, must be unique. Service Bus does not support function overloading with different method arguments. Identical function names that have different namespaces, and thus different prefixes, are allowed.
comment	A description of the function. While the description does not appear in the Service Bus user interface, you should provide guidance that shows how to invoke the function with meaningful argument names.
namespaceURI	The namespace of the function. For example, the Service Bus functions namespace is <code>http://www.bea.com/xquery/xquery-functions</code> . Namespaces and namespace prefixes must be unique. Custom namespaces that you provide appear in the default namespaces list in the XQuery editor.
className	The fully qualified custom Java class that implements the function.

Table B-1 (Cont.) Custom Function Registration File Properties

Element	Description
method	The custom Java method that implements the function, preceded by the return type; for example, <code>boolean isUserInGroup(java.lang.String, java.lang.String)</code> . If your method uses a single-dimensional array, see Using Single-Dimensional Arrays for guidance in making the entry in the XML file.
isDeterministic	A value of <code>true</code> or <code>false</code> declaring whether or not the function is deterministic. Deterministic functions always provide the same results; for example, a function that concatenates Strings. Non-deterministic functions return unique results; for example, a function that returns the time of day. Though you can use non-deterministic functions, the XQuery standard recommends that functions be deterministic to ensure XQuery engine optimization.
scope	The type of Service Bus resource to which the function applies, such as Pipeline or SplitJoin. You can define multiple scope elements.

Creating and Packaging the Custom Function Java Classes

Your custom functions do not appear in the expression editor until Service Bus can find your custom class. Use the following guidelines to create and package the class for a custom XPath function so Service Bus can locate and use it.

- [Creating the Class and Method](#)
- [Packaging the Custom Function Class](#)

Creating the Class and Method

Use the following guidelines for creating the Java class and method for a custom function.

- class: The class must be public.
- method: The method must be public and static.
- arguments and return values: The following table lists the supported types for method arguments and return values. If a type is not listed, it is not supported. Inner classes and multi-dimensional arrays are not supported.

Table B-2 Supported Java Method Types for Custom Functions

Java Type	XQuery Type	XSLT Type
<code>java.lang.String</code>	<code>xs:string</code>	string
<code>int, java.lang.Integer</code>	<code>xs:int</code>	number
<code>boolean, java.lang.Boolean</code>	<code>xs:boolean</code>	boolean

Table B-2 (Cont.) Supported Java Method Types for Custom Functions

Java Type	XQuery Type	XSLT Type
long, java.lang.Long	xs:long	number
short, java.lang.Short	xs:short	number
byte, java.lang.Byte	xs:byte	number
double, java.lang.Double	xs:double	number
float, java.lang.Float	xs:float	number
char, java.lang.Char	xs:string	object
java.math.BigInteger	xs:integer	number
java.math.BigDecimal	xs:decimal	number
java.util.Date	xs:datetime	See footnote ¹
java.sql.Date	xs:date	See footnote
java.sql.Time	xs:time	See footnote
javax.xml.namespace.QName	xs:Qname	See footnote
org.apache.xmlbeans.XmlObject	element()	See footnote
org.w3c.dom.Element	element()	The XSLT node-set type is not supported with custom XPath functions.

¹ Converted to a string, then passed back as its original type.

Using Single-Dimensional Arrays

Single-dimension arrays (using supported Java types) are mapped to corresponding XQuery types with an asterisk *, which is a wild card to imply the multiple cardinality of the array. For example:

```
public static XmlObject[] getArrayOfXmlObjects(XmlObject[] a)
```

is mapped to

```
namespace:getArrayOfXmlObjects($arg1 as element(*) as element()*)
```

In function signatures that have single-dimensional array input arguments or return values, you must use the type encoding described at <http://docs.oracle.com/javase/7/docs/api/java/lang/Class.html#getName%28%29>. The following examples show how to specify single-dimensional array methods in your custom function XML file using the required array encoding:

Java Method	Entry in Custom Function XML File
public static String[] myUppercaseStringArray(String[] arg)	Ljava.lang.String; myUppercaseStringArray([Ljava.lang.String;)

Java Method	Entry in Custom Function XML File
<pre>public static int[] myAddInts(int[] arg)</pre>	<pre>[I myAddInts([I)</pre>

Packaging the Custom Function Class

Service Bus must know about your custom function class in order to include your custom functions in the XQuery editors and let you use those functions. Package your custom function class in a JAR file, then place the JAR in one of the following directories:

- *service_bus_home*/config/xpath-functions/
- *domain_home*/config/osb/xpath-functions/

where *service_bus_home* is the location where Service Bus is installed and *domain_home* is the directory where the Service Bus domain is installed.

At IDE and server start-up, Service Bus looks for custom function classes in these directories to find the available custom functions. Be sure to correctly reference your custom class and method in the custom function XML file, described in [Registering Custom Functions with Service Bus](#).

After you add new custom functions, you must restart the IDE and any servers that will use the new functions.

Using Custom Functions

This section describes how to use custom functions in Service Bus XQuery expressions and resources.

- [Custom Functions In Inline XQuery Expressions and XQuery Resources](#)
- [Custom Functions In XSLT Resources](#)

Custom Functions In Inline XQuery Expressions and XQuery Resources

You can include custom functions in both inline XQuery expressions and in XQuery resources just as you would use functions provided by Service Bus.

Custom Functions In XSLT Resources

The syntax for invoking a custom function from within an XSLT resource varies by the XSLT engine you use with Service Bus. Given the following custom function code, the Syntax for Invoking a Custom Function with the Xalan Engine example, shown below, shows the syntax for invoking a custom function using the Xalan XSLT engine (the default on Microsoft Windows with the Oracle JDK).

```
package tests.pipeline;
public class CustomXQFunctions
{
    public static String myUppercaseString(String arg)
    {
        return arg.toUpperCase();
    }
}
```

Example - Syntax for Invoking a Custom Function with the Xalan Engine

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:param name="arg-string"/>
  <xsl:template name="myUppercaseString"
    xmlns:ns0="xalan://tests.pipeline.CustomXQFunctions">
    <xsl:variable name="upcase" select="ns0:myUppercaseString($arg-string)" />
    <originalInput>
      <xsl:value-of select="$arg-string" />
    </originalInput>
    <result>
      <xsl:value-of select="$upcase" />
    </result>
  </xsl:template>
  <xsl:template match="*">
    <xsl:copy>
      <xsl:call-template name="myUppercaseString"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

With an input document of `<example />` and an input `arg-string` value of `hello`, the transformation becomes:

```
<example>
  <originalInput>hello</originalInput>
  <result>HELLO</result>
</example>
```

Deploying Custom Functions in a Cluster

In a multiple-server environment with multiple Oracle Fusion Middleware product homes, you must manually add all custom function resources to any of those environments where the custom functions will be used. Clustering does not automatically distribute custom function resources across Managed Servers.

Oracle Service Bus APIs

This appendix provides an overview of the Service Bus APIs for updating and customizing resources, deploying resources, and managing and monitoring those resources in the runtime. Service Bus exposes APIs to allow customizing resources and to provide external access to monitoring data and deployment.

This appendix includes the following sections:

- [Resource Update and Customization](#)
- [Management and Monitoring](#)
- [Deployment](#)

Javadoc for the Oracle Service Bus APIs is provided in the *Java API Reference for Oracle Service Bus*.

Resource Update and Customization

Several APIs are exposed to allow customization of service definitions, WSDL documents, schemas, XQueries, and other design-time resources through programmatic interfaces. The supporting APIs allow loading ZIP files containing resources, in addition to moving, renaming, cloning, or deleting resources, folders, and projects. A typical use case is one in which you have a prototypical proxy service from which you make a number of copies; each copy can be modified programmatically.

Numerous customization options can be applied during deployment. For example, environment variables allow you to preserve or tailor settings when moving from one environment to another.

The available APIs include:

- `ProxyServiceConfigurationMBean`: Enable and disable proxy services, and enable and disable monitoring for a proxy service.
- `BusinessServiceConfigurationMBean`: Enable and disable business services, monitoring for a business service, throttling, offline URIs, and result caching, as well as detach a service from a UDDI registry.
- `PipelineServiceConfigurationMBean`: Enable and disable pipeline and SLA alerts for a pipeline.
- `CommonServiceConfigurationMBean`: Enable and disable a business or proxy service, and enable and disable message tracing.
- `FlowServiceConfigurationMBean`: Enable and disable SLA alerts and monitoring for a split join.

-
- `ALSBConfigurationMBean`: Manage resources in a Service Bus domain by performing the following tasks:
 - Query, export, and import resources
 - Obtain validation errors
 - Get and set environment values
 - Modify references inside resources to new references
 - Move, rename, clone, and delete resources
 - `ResultCacheRuntimeMBean`: Manage the result cache with methods to delete a single entry and delete all entries that belong to a specific business service.
 - `Customization`: Customize the Service Bus runtime by performing the following tasks:
 - Find and replace environment values
 - Assign environment values
 - Map references found in resources to other references

Management and Monitoring

The JMX Monitoring API in Oracle Service Bus provides external access to monitoring data. Java Management Extensions (JMX) technology was used for the implementation. Service Bus resources within a domain use JMX Managed Beans (MBeans) to expose their management functions. An MBean is a concrete Java class that is developed according to JMX specifications.

For more information, see "JMX Monitoring API" in *Administering Oracle Service Bus*.

Deployment

You can use the Service Bus MBeans in Java programs and WLST scripts to automate the promotion of Service Bus configurations from development environments through testing, staging, and finally to production environments. Numerous customization options can be applied during deployment. For example, an extended list of environment variables allows you to preserve or tailor settings when moving from one environment to another.

For information, see "Using the Deployment APIs" in *Administering Oracle Service Bus*.

Transport SDK Interfaces and Classes

This appendix lists and summarizes the classes and interfaces provided in the Service Bus Transport SDK.

For information on which interfaces are required to develop a custom transport provider, see [Developing Custom Transport Providers](#).

This appendix includes the following sections:

- [Introduction](#)
- [Schema-Generated Interfaces](#)
- [General Classes and Interfaces](#)
- [Source and Transformer Classes and Interfaces](#)
- [Metadata and Header Representation for Request and Response Messages](#)
- [User Interface Configuration](#)

Introduction

The Transport SDK classes and interfaces discussed in this chapter are located in `OSB_ORACLE_HOME/lib/modules/oracle.servicebus.kernel-api.jar` unless otherwise noted. `OSB_ORACLE_HOME` is the location in which you installed Service Bus.

For details on classes and methods, see the *Java API Reference for Oracle Service Bus*.

Schema-Generated Interfaces

A number of Transport SDK interfaces are generated from XML Schema by an XML Schema compiler tool. The source (XML Schema) for the following interfaces is provided in the file `TransportCommon.xsd`. This file is the base schema definition file for service endpoint configurations. This file is located in `OSB_ORACLE_HOME/lib/modules/oracle.servicebus.kernel-api.jar`.

The following interfaces are schema-generated:

- **EndPointConfiguration:** The base type for endpoint configuration. An endpoint is a Service Bus resource where messages are originated or targeted. `EndPointConfiguration` describes the complete set of parameters necessary for the deployment and operation of an inbound or outbound endpoint.
- **RequestMetaDataXML:** The base type for the metadata of an inbound or outbound request. Metadata is not carried in the payload of the message, but separately and is used as the context for processing the message. Examples of information that might be transmitted in the metadata are the Content-Type header, security information, or locale information.

-
- **RequestHeadersXML**: The base type for a set of inbound or outbound request headers.
 - **ResponseMetaDataXML**: The base type for response metadata for an inbound or outbound message.
 - **ResponseHeadersXML**: The base type for a set of response headers.
 - **TransportProviderConfiguration**: Allows you to configure (a) whether this provider generates a service description (for example, WSDL) for its endpoints; (b) whether this provider supports inbound (proxy) endpoints; or (c) whether this provider supports outbound (business service) endpoints.

General Classes and Interfaces

This section summarizes the general Transport SDK classes and interfaces that you use when developing your own custom transports. For detailed information on each class and interface listed in this section, refer to the *Java API Reference for Oracle Service Bus*.

- [Summary of General Classes](#)
- [Summary of General Interfaces](#)

Summary of General Classes

class TransportManagerHelper: Helper class that allows the client to execute some common tasks with respect to the transport subsystem.

class ServiceInfo: Wrapper class that describes information about a service, such as its transport configuration and its binding type.

class TransportOptions: Supplies options for sending or receiving a message. There are two styles for using `TransportOptions`: multiline setup and single-line use.

class EndPointOperations: Describes different types of transport endpoint lifecycle-related events by which the transport provider is notified. Nested classes include: `CommonOperation`, `Create`, `Delete`, `EndPointOperationTypeEnum`, `Resume`, `Suspend`, and `Update`.

class Ref: Uniquely represents a resource, project or folder that is managed by the configuration system. This class is located in `OSB_ORACLE_HOME/lib/modules/oracle.servicebus.configfwk.jar`.

class TransportValidationContext: Container that supplies information to transport providers that can be used when implementing validation checks of endpoint configuration.

class Diagnostics: Contains a collection of `Diagnostic` entries relevant to a particular resource. This class is located in `OSB_ORACLE_HOME/lib/modules/oracle.servicebus.configfwk.jar`.

class Diagnostic: Represents a particular validation message related to a resource. `Diagnostic` objects are generated as a result of validation that is performed when a resource changes. Such changes in the system trigger validation for the changed resource, as well as all other resources that (transitively) depend on the changed resource. This class is located in `OSB_ORACLE_HOME/lib/modules/oracle.servicebus.configfwk.jar`.

class NonQualifiedEnvValue: Represents an instance of an environment-dependent value in configuration data. Environment-dependent values normally change when moving the configuration from one domain to another. For example the URI of a

service could be different on test domain and production domains. This class is located in `OSB_ORACLE_HOME/lib/modules/oracle.servicebus.configfwk.jar`.

Summary of General Interfaces

interface TransportManager: A singleton object that provides the main point of centralization for managing different transport providers, endpoint registration, control, processing of inbound and outbound messages, and other points.

interface TransportProvider: Represents the central point for management of transport protocol-specific configuration and runtime properties. There is a single instance of `TransportProvider` for every supported protocol. For example, there is a single instance of HTTP transport provider, JMS transport provider, and so on.

interface BindingTypeInfo: Describes the binding details of the service. The implementation is a convenience wrapper class around several internal Service Bus structures. Additional methods can be added as needed by transport providers.

interface TransportWLSArtifactDeployer: The plug-in interface for modules that need to deploy, undeploy, or modify Oracle WebLogic Server related artifacts along with a Service Bus deployment. For example, in certain cases, Oracle WebLogic Server queues need to be deployed in response to the creation of a service.

Tip:

For more information, see [When to Implement TransportWLSArtifactDeployer](#).

interface SelfDescribedTransportProvider: Extends `TransportProvider`. The transport providers that generate a service binding type description from a given transport endpoint need to implement this interface. An example is the EJB transport provider.

interface SelfDescribedBindingTypeInfo: Extends the `BindingTypeInfo` interface for services that are self-described, such as EJB services.

interface WsdlDescription: Describes the WSDL document associated with a registered Service Bus service.

interface TransportCustomBindingProvider: Represents a class responsible for facilitating the generation of the extensible elements of SOAP binding or all the elements of custom Service Bus binding used during the generation of effective WSDL documents for WSDL-based transports. This includes information such as the transport URI for the `<soap:binding>` element and the location attribute for `<soap:address>`.

interface ServiceTransportSender: Sends outbound messages to a registered service associated with a transport endpoint. `TransportProvider.sendMessageAsync()` gets an instance of `ServiceTransportSender` from which the provider can retrieve the payload and metadata for outbound requests. This interface extends `TransportSender`.

interface CredentialCallback: Transport providers get an instance of this callback interface from Service Bus. The transport provider can call its methods to fetch a credential used for outbound authentication.

interface TransportEndPoint: A transport endpoint is an Service Bus entity or resource where service messages are originated or targeted.

Source and Transformer Classes and Interfaces

This section provides descriptions of the base `Source` and `Transformer` interfaces, along with several concrete `Sources` provided with Service Bus and some supporting classes. For more information, see [Designing for Message Content](#).

- [Summary of Source and Transformer Interfaces](#)
- [Summary of Source and Transformer Classes](#)

Summary of Source and Transformer Interfaces

interface `Source`: Represents source content in some form. `Sources` may be transformed into other `Sources` through a `Transformer` instance. At minimum, a `Source` must natively support conversion to a byte-based stream using the two methods defined in this interface. `Source` may or may not take into account various `TransformOptions`, such as character-set encoding, during serialization.

interface `SingleUseSource`: A marker interface indicating that a type of `Source` can only be consumed once. It also provides one helper method that can be used to determine if the `Source` is still consumable, or *valid*.

If you create a `Source` class that implements the `Source` interface, Service Bus is free to call the `getInputStream()` method multiple times, each time retrieving the input stream from the beginning. If the `Source` class implements `SingleUseSource`, Service Bus calls `getInputStream()` only once; however, Service Bus buffers the entire message in memory in this case.

interface `Transformer`: Transforms one type of `Source` to another. The instance is responsible for indicating what types of sources it can convert between. Note that a transformer is required to support the full cross-product of transformations implied by the supported input and output sources. In other words, a transformer must support transforming any supported input source to any supported output source.

Summary of Source and Transformer Classes

class `StreamSource`: A byte-stream `Source` whose content comes from an `InputStream`. As a byte-stream source, the serialization methods do not heed any transformation options.

Note:

Because this stream is backed by an `InputStream`, this is a single-use `Source`. Both serialization methods pull from the same underlying `InputStream`, and once that content is consumed, it is gone. The push-based `writeTo()` method results in all data being consumed immediately, assuming no error occurs. The pull-based `getInputStream()` actually gives the underlying `InputStream` directly to the caller.

class `ByteArraySource`: A byte-stream `Source` whose content comes from a byte array. As a byte-stream source, the serialization methods do not heed any transformation options.

class `StringSource`: A `Source` that is backed by a single `String`. Serialization is simply a character-set encoded version of the character data.

class XmlObjectSource: Apache XBean *Source* content is represented as an Apache XBean. The XBean may be typed and so may be accompanied by a *SchemaType* object and an associated *ClassLoader*. However, both of these are entirely optional and the XBean can be untyped XML.

class DOMSource: A *Source* whose content comes from a DOM node. The referenced node may be a full-fledged `org.w3c.dom.Document`, but it may also be an internal node in a larger document.

class MFLSource: Represents MFL content. MFL data is essentially binary data that has some logical structure imposed on it by an MFL definition. CSV is a simple example of MFL data, but the structure can be arbitrarily complex. The logical/in-memory representation of the data is an XML document, but its serialized representation is the raw unstructured binary data.

class SAAJSource: A *Source* that is backed by a SAAJ *SOAPMessage* object. A *SAAJSource* is typically converted to and from *MessageContextSource* and *MimeSource*.

class MimeSource: A *Source* representing arbitrary content with headers. Essentially this is a *Source* that represents a MIME part. Headers must conform to RFC822 whereas the *Source* can be of any type. The serialization format for this *Source* is a fully-compliant MIME package. This *Source* is also aware of Content-Transfer-Encoding, and it will perform the proper encoding of the underlying content stream if the header is present. Note that this means that the *Source* provided to the constructor should be in raw form and not be already encoded.

class MessageContextSource: A *Source* that represents all message content. The *Source* for the message and attachments are left untyped to allow for deferred processing. Eventually, however, the attachments source will likely be converted into an object and the message source will likely be converted to a specific typed source such as an *XmlObjectSource* or a *StringSource*.

Note:

The serialization format of a *MessageContextSource* is always a MIME multipart/related package, irrespective of the native serializations of the message and attachment sources. However, if this serialized object is needed more than once, it is best to transform the *Source* into a *MimeSource*.

class TransformOptions: Represents a set of transformation options. Instances of this class are used in conjunction with the *Transformer* class to influence how an input source is converted to an output source (for example, a change in character-set encoding from SHIFT_JIS to EUC-JP). This class is also used by the *InputStream* and *OutputStream* methods of the *Source* interface, since that is effectively also a transformation between the *Source* and the byte-level representation in the *InputStream* and *OutputStream*.

class JavaObjectSource: Represents the payload carried by Service Bus transports that provide a Java messaging type, such as the JMS transport. The objects that make up this payload are registered in the pipeline Java object repository by the binding layer, and their contents are visible in message context variables through `<ctx:java-content ref='jcid:xyz' xmlns:ctx="http://www.bea.com/wli/sb/context" />` XML elements. In this example, `ref` points to the unique ID of the object in the Java object repository.

class `JavaXmlSource`: Represents the payload carried by the services that supports Java objects as the arguments, such as the JEJB transport. `JavaXmlSource` is made up of an XML representation that defines the shape of the message body in the pipeline and a map containing Java objects against unique keys. In the XML representation, Java object arguments are substituted by `<ctx:java-content ref='jcid:xyz' xmlns:ctx="http://www.bea.com/wli/sb/context" />` elements, where the `ref` attribute equals a key in the `JavaObjects` map that indexes the replaced Java object. The map contains the objects to be registered in pipeline Java object repository against the unique IDs in the XML representation.

Metadata and Header Representation for Request and Response Messages

This section lists classes and interfaces that deal with request and response message metadata representation. For additional information, see [Handling Messages](#) and [Designing for Message Content](#).

- [Runtime Representation of Message Contents](#)
- [Interfaces](#)

Runtime Representation of Message Contents

abstract class `CoLocatedMessageContext`: Needs to be extended by a transport provider that implements optimization for co-located outbound calls to go through a Java method invocation instead of the transport layer. For an example implementation, see the class

`com.bea.alsb.transports.sock.SocketCoLocatedMessageContext.java`, which is part of the Sample Socket Transport described in [Creating a Sample Socket Transport Provider](#). For additional information, see [Co-Located Calls](#)."

abstract class `RequestHeaders`: Represents a union of standard and user-defined headers in a given inbound or outbound request message. The set of standard headers is specific to each transport provider. This is an abstract class to be extended by each transport provider to implement its version of request headers.

abstract class `RequestMetaData<T extends RequestHeaders>`: Represents inbound or outbound request message metadata information (for example, headers, request character set encoding, and so on.) Transport providers provide an extension of this class that adds metadata information applicable to the transport provider. For example, HTTP transport provider adds `get/setQueryString()`, `getClientHost()`, and other methods.

abstract class `ResponseHeaders`: Represents a union of standard and user-defined headers in a given inbound or outbound response message. The set of standard headers is specific to each transport provider. This is an abstract class to be extended by each transport provider to implement their version of response headers.

abstract class `ResponseMetaData<T extends ResponseHeaders>`: Represents inbound or outbound response message metadata information (such as headers, request character set encoding, and so on.) Transport providers provide an extension of this class that adds metadata information applicable to the transport provider. For example, HTTP transport provider adds `get/setHttpResponseCode()` and other methods.

Interfaces

interface `TransportMessageContext`: Most message-oriented middleware (MOM) products treat messages as lightweight entities that consist of a header and a payload. The header contains fields used for message routing and identification; the payload

contains the application data being sent. In general, the transport-level message context consists of a message ID, `RequestMetadata`, request payload, `ResponseMetaData`, response payload and related properties.

interface `InboundTransportMessageContext`: Implements the message context abstraction for incoming messages.

interface `OutboundTransportMessageContext`: Implements the message context abstraction for outgoing messages.

interface `ServiceTransportSender`: Sends outbound messages to a registered service. The service is associated with a transport endpoint.

interface `TransportSendListener`: This is the callback object supplied to the outbound transport allowing it to signal to the system that response processing can proceed. This callback object should be invoked on a separate thread from the request message.

User Interface Configuration

Each transport provider can decide on a list of service endpoint specific configuration properties to persist, so a flexible user interface is required that allows the user to enter provider-specific configuration properties for each new service endpoint. The set of classes and interfaces described in this section allow each transport provider to expose its own properties for the user to enter as part of the service definition editors in the Oracle Service Bus Console. Use these interfaces and classes to develop the user interface for a new transport.

- [Summary of UI Interfaces](#)
- [Summary of UI Classes](#)

Summary of UI Interfaces

interface `TransportProviderFactory`: This interface registers the new transport provider with the transport manager.

interface `TransportUIBinding`: Represents an object responsible for rendering provider-specific UI pages used for defining the service, providing a summary, and validating transport provider specific endpoint configurations.

interface `CustomHelpProvider`: Lets you provide context-sensitive help for the functionality you add to the Oracle Service Bus Console, such as custom transports. For implementation details, see [Creating Help for Custom Transports](#).

Summary of UI Classes

class `TransportUIContext`: Supplies options for portions of the user interface that are specific to the transport provider. It is passed by the Oracle Service Bus Console to each transport provider.

class `TransportUIGenericInfo`: Holds transport-specific UI information for the common transport page in the service definition editors.

class `TransportUIFactory`: Provides factory methods for creating a Transport Edit Field and different kinds of Transport UI objects associated with the field. Also provides some helper methods for accessing values in these objects.

class `TransportEditField`: Represents a single editable UI element in the provider-specific portion of the Oracle Service Bus Console service definition editors.

class `TransportViewField`: Represents a single read-only UI element in the provider-specific portion of the service summary page.

class TransportUIError: Returns validation errors to the Oracle Service Bus Console.

Transport SDK UML Sequence Diagrams

This appendix contains UML sequence diagrams that describe the flow of method calls through the Service Bus runtime.

This appendix includes the following sections:

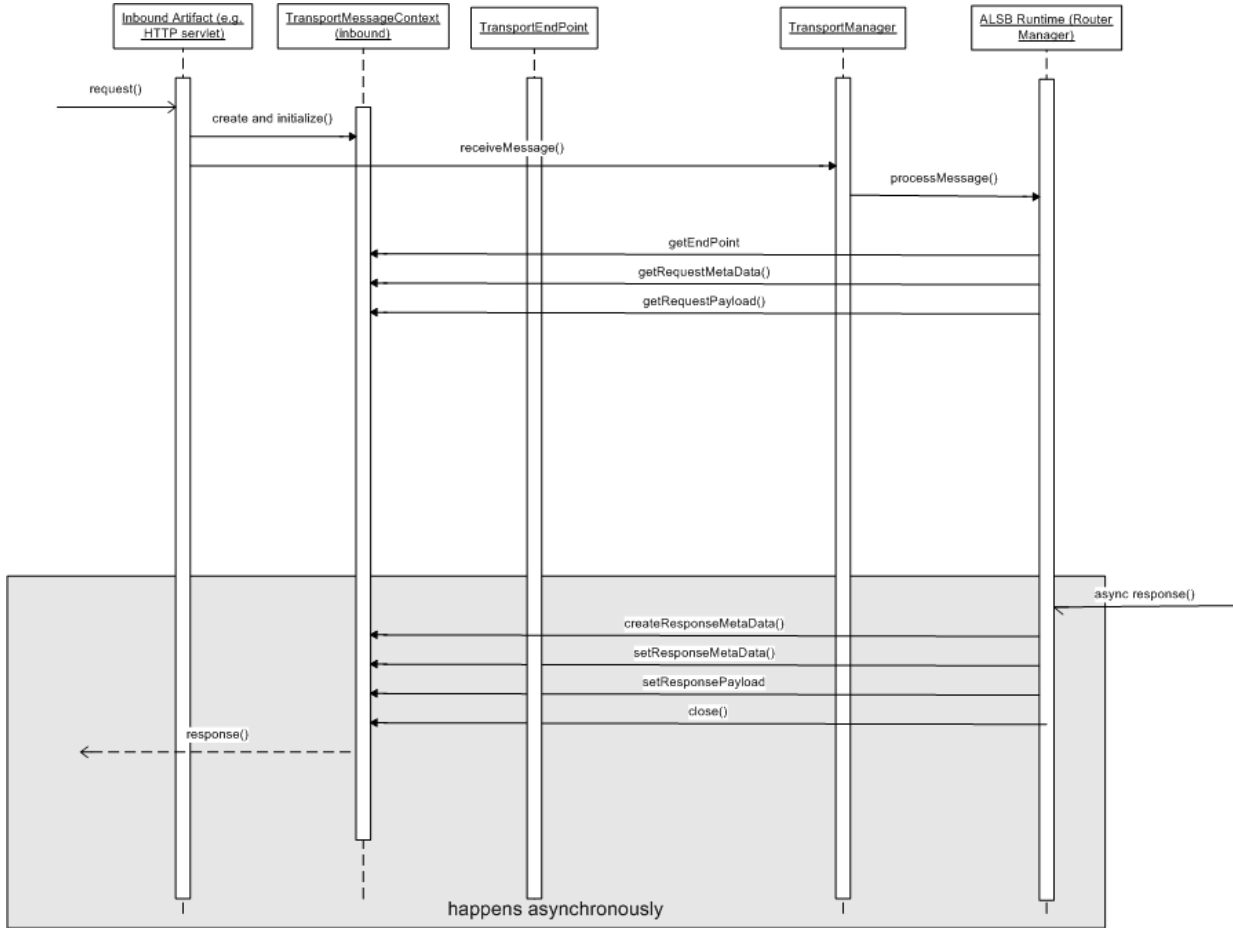
- [Service Bus Runtime Inbound Messages](#)
- [Service Bus Runtime Outbound Messages](#)
- [Design Time Service Registration](#)

Service Bus Runtime Inbound Messages

The sequence diagram in [Figure E-1](#) describes the flow of inbound messages through Service Bus at runtime.

First, an inbound artifact, such as an HTTP Servlet, intercepts a client request. The transport provider creates a data structure called `InboundTransportMessageContext`. The message context packages headers from the request into a metadata object, converting the payload from an HTTP stream into a specific Service Bus source object. The transport provider calls the transport manager to receive the message. The transport manager preprocesses the message and passes the message to the Service Bus runtime for processing. The runtime asks for the message context's service, service version, and other information. It also asks about the metadata and payload, which are required for processing. The runtime asks the `MessageContext` to create the response metadata and the response payload, and then calls `close()`. The response is sent back to the client.

Figure E-1 Inbound Messages at Runtime



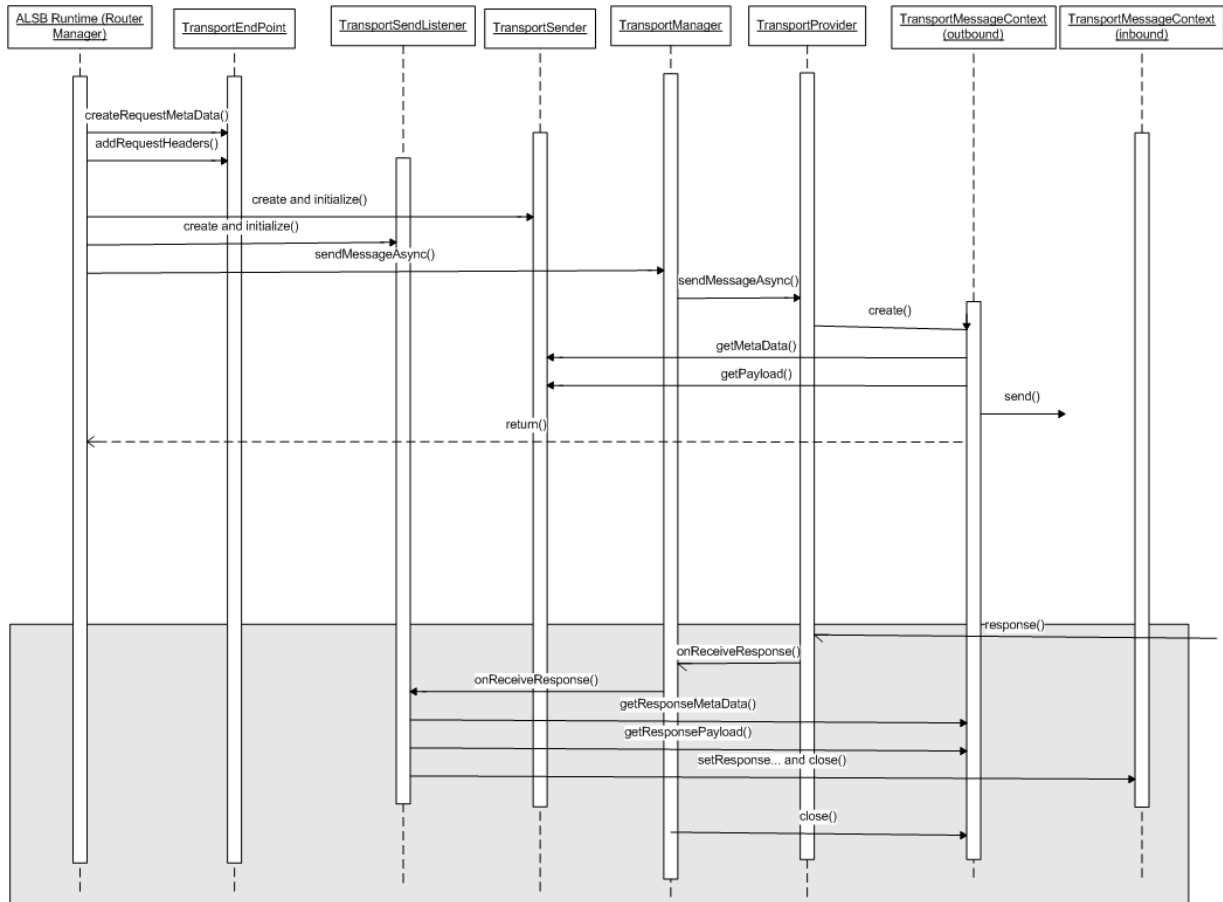
Service Bus Runtime Outbound Messages

The sequence diagram shown in [Figure E-2](#) describes the flow of outbound messages through the Service Bus runtime.

The Service Bus runtime routes the message to an external service. The transport provider creates metadata for the request and creates a `TransportSender` object, which includes information about the payload and quality of service and retry information. Next, the provider calls `TransportManager` (the central hub for the transport subsystem) to send the message asynchronously. `TransportManager` calls the transport provider to send the message. The transport provider creates an `OutboundTransportMessageContext`. The transport provider then asks about the metadata and payload and other information and takes appropriate action. For example, for a JMS message, the transport provider uses the JMS API to populate the headers and the payload and calls the protocol-specific send operation.

When a response comes in, the transport provider calls the `TransportSendListener` object. Eventually the transport manager invokes the response pipeline. After pipeline actions are executed, the outbound endpoint is closed.

Figure E-2 Outbound Messages at Runtime

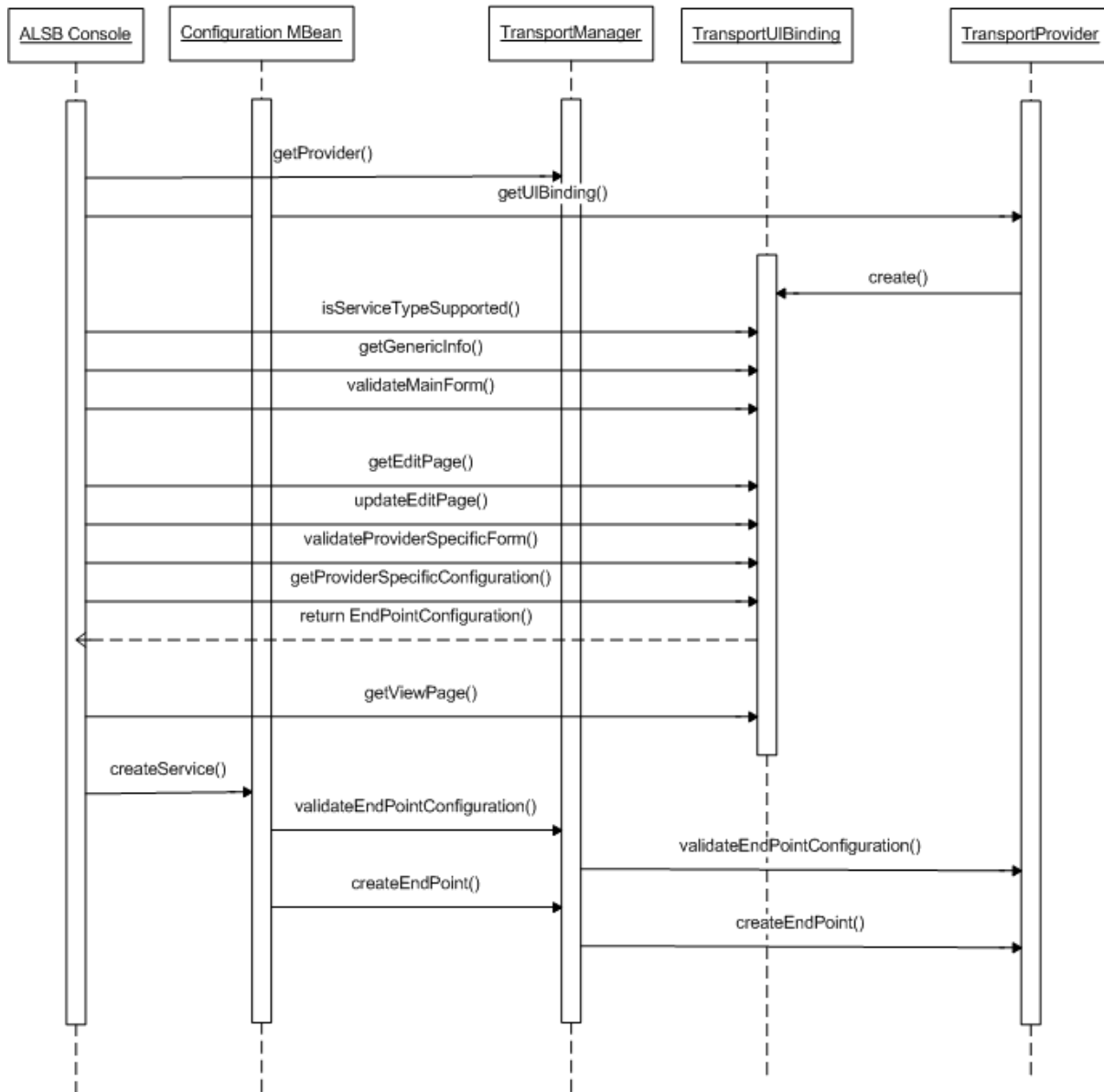


Design Time Service Registration

When you create a service, a wizard guides you through a number of Oracle Service Bus Console pages. When you select a transport type, the Oracle Service Bus Console calls the transport manager to retrieve an object for each one of these entries in the list and gets a binding from each transport provider. This binding answers questions requested by the console, such as what is or is not supported. This step allows the console page to be populated with appropriate information. [Figure E-3](#) describes the service creation process. Below are the basic steps for a transport-based service:

1. Specify the name of the service.
2. Select from a list of transport providers (protocols).
3. Select and optionally configure the service type.
4. Create the service.
5. Review the configuration and save any changes you make. If you created a proxy service, you must specify a target service, which can be a pipeline or another proxy service.

Figure E-3 Service Registration



XQuery-SQL Mapping Reference

This chapter provides information about the native RDBMS Data Type support and XQuery mappings that the Oracle XQuery engine generates or supports.

This chapter includes the following sections:

- Core RDBMS Data Type Mapping:
 - [IBM DB2/NT 8](#)
 - [Microsoft SQL Server](#)
 - [Oracle8i, 8.1.x](#)
 - [Oracle 9i and Later](#)
 - [Sybase 12.5.2 \(and higher\)](#)
- [Base \(Generic\) RDBMS Data Type Mapping](#)

For information about using these mappings in Oracle Service Bus XQueries, see [Accessing Databases Using XQuery](#).

For complete information about database and JDBC drivers support in Oracle Service Bus, see *Oracle Fusion Middleware Supported System Configurations* at:

<http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html>

IBM DB2/NT 8

The following table lists the data type mappings that the XQuery engine generates or supports for IBM DB2/NT 8.

Table F-1 IBM DB2 Data Type Mappings

DB2 Data Type	XQuery Type
BIGINT	xs:long
BLOB	xs:hexBinary
CHAR	xs:string
CHAR() FOR BIT DATA	xs:hexBinary
CLOB ¹	xs:string
DATE	xs:date

Table F-1 (Cont.) IBM DB2 Data Type Mappings

DB2 Data Type	XQuery Type
DOUBLE	xs:double
DECIMAL(p,s) ² (NUMERIC)	xs:decimal (if s > 0), xs:integer (if s = 0)
INTEGER	xs:int
LONG VARCHAR ¹	xs:string
LONG VARCHAR FOR BIT DATA	xs:hexBinary
REAL	xs:float
SMALLINT	xs:short
TIME ³	xs:time
TIMESTAMP ⁵	xs:dateTime ⁴
VARCHAR	xs:string ⁴
VARCHAR() FOR BIT DATA	xs:hexBinary

¹ Pushed down in project list only.

² Where *p* is precision (total number of digits, both to the right and left of decimal point) and *s* is scale (total number of digits to the right of decimal point).

³ Accurate to 1 second.

⁴ Values converted to local time zone (timezone information removed) due to TIME and TIMESTAMP limitations.

⁵ Precision limited to milliseconds.

Microsoft SQL Server

The following table lists the data type mappings that the XQuery engine generates or supports for Microsoft SQL Server.

Table F-2 SQL Server 2000 Data Type Mapping

SQL Data Type	XQuery Type
BIGINT	xs:long
BINARY	xs:hexBinary
BIT	xs:boolean
CHAR	xs:string
DATETIME ¹	xs:dateTime ²
DECIMAL(p,s) ³ (NUMERIC)	xs:decimal (if s > 0), xs:integer (if s = 0)
FLOAT	xs:double

Table F-2 (Cont.) SQL Server 2000 Data Type Mapping

SQL Data Type	XQuery Type
IMAGE	xs:hexBinary
INTEGER	xs:int
MONEY	xs:decimal
NCHAR	xs:string
NTEXT ⁴	xs:string
NVARCHAR	xs:string
REAL	xs:float
SMALLDATETIME ⁵	xs:dateTime
SMALLINT	xs:short
SMALLMONEY	xs:decimal
SQL_VARIANT	xs:string
TEXT ⁴	xs:string
TIMESTAMP	xs:hexBinary
TINYINT	xs:short
VARBINARY	xs:hexBinary
VARCHAR	xs:string
UNIQUEIDENTIFIER	xs:string

¹ Fractional-second-precision up to 3 digits (milliseconds). No timezone.

² Values converted to local time zone (timezone information removed) and fractional seconds truncated to milliseconds due to DATETIME limitations.

³ Where *p* is precision (total number of digits, both to the right and left of decimal point) and *s* is scale (total number of digits to the right of decimal point).

⁴ Pushed down in project list only.

⁵ Accuracy of 1 minute.

Oracle8i, 8.1.x

The following table lists the data types that the XQuery engine generates or supports for Oracle 8.1.x (Oracle 8i).

Table F-3 Oracle 8.1.x Data Type Mapping

Oracle 8 Data Type	XQuery Type
BFILE	not supported
BLOB	xs:hexBinary

Table F-3 (Cont.) Oracle 8.1.x Data Type Mapping

Oracle 8 Data Type	XQuery Type
CHAR	xs:string
CLOB ¹	xs:string
DATE ²	xs:dateTime
FLOAT	xs:double
LONG ¹	xs:string
LONG RAW	xs:hexBinary
NCHAR	xs:string
NCLOB ¹	xs:string
NUMBER	xs:double
NUMBER(p,s) ³	xs:decimal (if s > 0), xs:integer (if s <=0)
NVARCHAR2	xs:string
RAW	xs:hexBinary
ROWID	xs:string
UROWID	xs:string

¹ Pushed down in project list only.

² Does not support fractional seconds.

³ Where *p* is precision (total number of digits, both to the right and left of decimal point) and *s* is scale (total number of digits to the right of decimal point).

Oracle 9i and Later

The following table lists the data type and other mappings that the XQuery engine generates or supports for Oracle Database 9i, 10g, 11g, and 12c. Note that Oracle treats empty strings as NULLs, which deviates from XQuery semantics and may lead to unexpected results for expressions that are pushed down.

Table F-4 Oracle 9i and later Data Type Mapping

Oracle Data Type	XQuery Type
BFILE	not supported
BLOB	xs:hexBinary
CHAR	xs:string
CLOB ¹	xs:string
DATE	xs:dateTime ²
FLOAT	xs:double

Table F-4 (Cont.) Oracle 9i and later Data Type Mapping

Oracle Data Type	XQuery Type
INTERVAL DAY TO SECOND	xdt:dayTimeDuration
INTERVAL YEAR TO MONTH	xdt:yearMonthDuration
LONG ¹	xs:string
LONG RAW	xs:hexBinary
NCHAR	xs:string
NCLOB ¹	xs:string
NUMBER	xs:double
NUMBER(p,s)	xs:decimal (if s > 0), xs:integer (if s <=0)
NVARCHAR2	xs:string
RAW	xs:hexBinary
ROWID	xs:string
TIMESTAMP	xs:dateTime ³
TIMESTAMP WITH LOCAL TIMEZONE	xs:dateTime
TIMESTAMP WITH TIMEZONE	xs:dateTime
VARCHAR2	xs:string
UROWID	xs:string

¹ Pushed down in project list only.

² When SDO stores xs:dateTime value in Oracle DATE type, it is converted to local time zone and fractional seconds are truncated due to DATE limitations.

³ XQuery engine maps XQuery xs:dateTime to either TIMESTAMP or TIMESTAMP WITH TIMEZONE data type, depending on presence of timezone information. Storing xs:dateTime using SDO may result in loss of precision for fractional seconds, depending on the SQL type definition.

Sybase 12.5.2 (and higher)

The following table lists the data types that the XQuery engine generates or supports for Sybase 12.5.2 (and higher).

Note:

Sybase deviates from XQuery semantics (which ignores empty strings) and treats empty strings as a single-space string.

Table F-5 Sybase 12.5.2 Data Type Mapping

Sybase Data Type	XQuery Type
BINARY	xs:hexBinary
BIT	xs:boolean
CHAR	xs:string
DATE	xs:date
DATETIME ¹	xs:dateTime ²
DECIMAL(p,s) ³ (NUMERIC)	xs:decimal (if s > 0), xs:integer (if s == 0)
DOUBLE PRECISION	xs:double
FLOAT	xs:double
IMAGE	xs:hexBinary
INT (INTEGER)	xs:int
MONEY	xs:decimal
NCHAR	xs:string
NVARCHAR	xs:string
REAL	xs:float
SMALLDATETIME ⁴	xs:dateTime
SMALLINT	xs:short
SMALLMONEY	xs:decimal
SYSNAME	xs:string
TEXT ⁵	xs:string
TIME	xs:time
TINYINT	xs:short
VARBINARY	xs:hexBinary
VARCHAR	xs:string

¹ Supports fractional seconds up to 3 digits (milliseconds) precision; no timezone information.

² When SDO stores xs:dateTime value in Oracle DATE type, it is converted to local time zone and fractional seconds are truncated due to DATE limitations.

³ Where *p* is precision (total number of digits, both to the right and left of decimal point) and *s* is scale (total number of digits to the right of decimal point).

⁴ Accurate to 1 minute.

⁵ Expressions returning text are pushed down in the project list only.

Base (Generic) RDBMS Data Type Mapping

When mapping SQL to XQuery data types, the XQuery engine first checks the JDBC typecode. If the typecode has a corresponding XQuery type, the XQuery engine uses the matching native type name. If no matching typecode or type name is available, the column is ignored. [Table F-6](#) shows this mapping.

Table F-6 RDBMS Data Type Mapping

JDBC Data Type	Typecode	XQuery Data Type
BIGINT	-5	xs:long
BINARY	-2	xs:string
BIT	-7	xs:boolean
BLOB	2004	xs:hexBinary
BOOLEAN	16	xs:boolean
CHAR	1	xs:string
CLOB ¹	2005	xs:string
DATE	91	xs:date
DECIMAL (p,s)	3	xs:decimal (if s > 0), xs:integer (if s =0)
DOUBLE	8	xs:double
FLOAT	6	xs:double
INTEGER	4	xs:int
LONGVARBINARY	-4	xs:hexBinary
LONGVARCHAR ¹	-1	xs:string
NUMERIC (p,s) ³	2	xs:decimal (if s > 0), xs:integer (if s =0)
REAL	7	xs:float
SMALLINT	5	xs:short
TIME ⁴	92	xs:time ⁴
TIMESTAMP ⁴	93	xs:dateTime ²
TINYINT	-6	xs:short
VARBINARY	-3	xs:hexBinary
VARCHAR	12	xs:string
OTHER vendor-specific JDBC type codes	1111	Oracle Service Bus uses native data type name to map to an appropriate XQuery data type.

-
- ¹ Pushed down in project list only.
 - ² Values converted to local time zone (timezone information removed) due to DATE limitations.
 - ³ Where p is precision (total number of digits, both to the right and left of decimal point) and s is scale (total number of digits to the right of decimal point).
 - ⁴ Precision of underlying RDBMS determines the precision of TIME data type and how much truncation, if any, will occur in translating xs:time to TIME.

Work Managers and Threading

This appendix describes the internal threading model used by Oracle Service Bus and its implications regarding performance and server stability. It focuses on the HTTP transport.

This appendix includes the following sections:

- [Key Threading Concepts](#)
- [Pipeline Actions](#)
- [Work Managers](#)
- [Designating Work Managers](#)

Key Threading Concepts

The following concepts are important to consider when assigning Work Managers to services.

- Request and response pipelines always execute in separate threads. While the request thread originates from the proxy service transport, the response thread originates from the business service transport.
- When external services are invoked, threads can be blocking or non-blocking, depending on the pipeline action, the Quality of Service (QoS) configuration, and the transport being used.
- When using blocking calls, a Work Manager with a minimum thread constraint must be associated with the response in order to prevent server deadlocks.

Service Bus optimizes invocations between proxy services. When one HTTP proxy service calls a second HTTP proxy service, the transport layer is bypassed. The request message is not sent using a network socket, so the transport overhead is eliminated. Instead, the thread processing the initial proxy service continues to process the request pipeline of the called service. Similarly, when invoking a business service, the proxy service thread is also used to send the request. Because the HTTP transport uses the asynchronous capabilities of WebLogic Server, the response of the business service is processed by a different thread.

For discussions about threading, it is important to remember that Service Bus runs on top of WebLogic Server. At times, Service Bus depends on WebLogic Server HTTP, JMS, and core engines for executing different types of requests. In some cases Service Bus hands over part of the processing to WebLogic Server. WebLogic Server executes it by using threads available in the WebLogic Server Self-Tuning thread pool. Service Bus documentation does not cover scenarios in which Service Bus depends on WebLogic Server and its Self-Tuning thread pool for processing.

As an example, in the case of a service call out, even it is a synchronous call within the Service Bus layer, responsibility to read the response from the backend lies with WebLogic Socket Muxer threads. After these Muxer threads get a response from the backend system, they notify the Service Bus thread, which is blocked and waiting for a response.

It is strongly recommended that you ensure that all Service Bus proxy and business services are configured to execute using threads from their respective Work Managers . Work Manager minimum and maximum constraints should be set based on the maximum load. This ensures that proxy and business services should not depend on the WebLogic Server execute thread pool for execution and WebLogic Server has enough free or idle threads to process any ad-hoc requests from Service Bus without leading to stuck thread problems.

Pipeline Actions

There are three pipeline actions that specifically affect threading: route actions, publish actions, and service callout actions.

Route Action

By default, the HTTP transport uses asynchronous features of WebLogic Server to prevent thread blocking while waiting for a business service response. For the execution of a route action, once the thread finishes sending the request, it returns to the pool where it is then used to process other work. When a response is returned from the external service, a second thread is scheduled to process it. This behavior can be modified by using a route options action and setting the QoS to **Exactly Once**.

Publish Action

A publish action is a one-way send. It provides the means of invoking an external service but without receiving a response. This is often used to provide notification of an event, such as for logging or auditing. By default, no feedback of whether the call was successful or not is returned to the pipeline thread. For both of the above actions, setting the QoS to **Exactly Once** forces the request thread to block until a response is received. This allows the request thread to notify the caller of an error immediately, without callback. This behavior is also useful when attempting to throttle the number of threads simultaneously processing a proxy service.

Service Callout Action

A service callout is implemented as a synchronous blocking call. Its design intention is to provide the ability to invoke an external service to enrich a request message prior to routing the request to the target service. While the callout is awaiting a response, the request pipeline thread blocks until a response thread notifies it that the response is ready and processing can continue.

Work Managers

WebLogic Server uses a self-tuning thread pool for executing all application-related work. The pool size is managed by an increment manager which adds or removes threads to the pool when it deems it necessary. The number of active threads will never exceed 400. As requests enter the server, a scheduler manages the order in which the requests are executed. When the number of requests exceeds the number of available threads, they are queued and then executed as threads return to the pool and become available. Work Managers indicate the type of work and priority of a request to the scheduler.

You create and configure Work Managers using the WebLogic Server Administration Console. This section describes Work Manager concepts that are key to Service Bus optimization. For more information about Work Managers, see "Using Work Managers to Optimize Scheduled Work" in *Administering Server Environments for Oracle WebLogic Server*.

Work Manager Configuration

Two key properties when configuring a Work Manager are **Max Thread Constraints** and **Min Thread Constraints**. A maximum thread constraint limits the number of concurrent threads executing a type of request by restricting the scheduler from executing more than the configured number at one time. However, the thread pool is shared among all Work Managers, so there is no guarantee the maximum number of threads will be available for processing at any given time.

A minimum thread constraint guarantees a minimum number of threads for processing. If sufficient threads are not available in the thread pool to process up to the minimum number, the scheduler uses standby threads to satisfy the minimum. Standby threads are not counted as part of the maximum number of 400 threads in the pool. When a thread is executing a request associated with a Work Manager containing a minimum thread constraint, the Work Manager first checks the queue for another request associated with the same constraint and executes it (instead of returning to the free pool). For this reason, use minimum thread constraints judiciously. Over-use can cause resource starvation of the default Work Manager, leading to unpredictable results.

Work Manager Priority

By default, Work Managers have equal priority with the scheduler, with a share value of 50. When the scheduler attempts to execute waiting requests, it ensures that requests associated with each Work Manager are given an equal number of thread resources (assuming an equal number of waiting requests).

From within Service Bus, a Work Manager is associated with a service by specifying a *dispatch policy*. If you have a simple proxy service that routes to a business service, you can assign different dispatch policies to each service so the scheduler recognizes that new requests are different work from responses received. Subsequently, each Work Manager is scheduled evenly when work requests (either a new incoming request message for a proxy service or a response message for a business service) are added to the queue. This becomes vital when a business service is invoked with a blocking call, which is the case with a service callout.

Once a thread is processing under the designation of a specific Work Manager, it continues to do so until it is returned to the pool. Therefore, when invoking a business service or a proxy service from within a pipeline, the currently executing thread continues processing under the Work Manager of the initial proxy service. The Work Manager specified for the service being called is ignored in this case.

Designating Work Managers

The Work Manager (dispatch policy) configuration for a business service should depend on how the business service is invoked. If a proxy service invokes the business service using a service callout, a publish action, or routing with exactly-once QoS (as described in [Pipeline Actions](#)), consider using different Work Managers for the proxy service and the business service instead of using the same for both. For the business service Work Manager, configure the Min Thread Constraint property to a small number (1-3) to guarantee an available thread.

